Lab: Efficient sampling using Quad-tree and Kd-tree

Auke-Dirk Pietersma

December 19, 2022

1 The Problem

Working with large datasets can introduce a multitude of perfomance issues. In this lab the student will program an efficient querying technique using Kd-trees and furthermore implement a sampling strategy through Quad-trees.

A common problem amongst applications that visualize geographical data is how to minimize the amount of data that is being send over thet internet yet still providing enough detail for the user. For example, displaying all the houses of the city of Groningen onto a map on your phone might not be the best idea if you are trying to navigate through a particular area.

One solution, on which we will focus during this lab, is data subsampling. The most straight forward solution would be to return only the frist n elements as seen in figure(2).

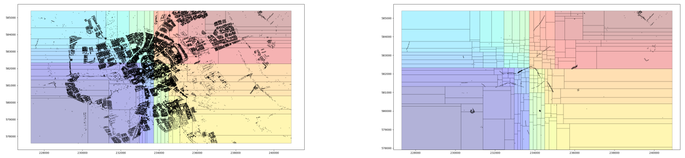

Figure 1: All houses

Figure 2: First n houses

Figure 3: The City of Groningen

Figure(2) clearly not does illustratie the space and shape of the city as portrayed in figure(1). The student will address this particular issue throughout this lab.

The main idea is to sample the situation as ‘as evenly‘as possible. Where the sample size, which we will later address as quad level/depth, can be chosen by the user. The roles that *depth* and *level* have will be come apparent throughout the lab.

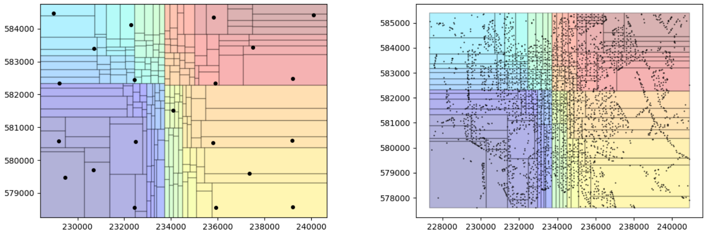

Figure 4: depth 4 and level 3

Figure 5: depth 8 and level 7

Figure 6: Sampling using Quadtree

Figures(4,5) demonstrate the impact Quad-tree sampling can have on the spatial distribution of your sampled data.

2 Lab

The student needs to improve on the provided code such that sampling through Quadtrees is possible. In the final program there should be two variables controlling the final sampling:

- (Quadtree) quadtree : add a QuadTree with certain depth
- (Plotter) quadlevel: at which level the points should be sampled.

In order to achieve the above you will first need to implement the following:

KDTree	def closest(self, point, sidx = si.StorageIndex())
QuadTree	def recurse(self,bbox, depth)

This lab is designed to be run from within a docker environment, therefore all plots generated will be written to the current working directory (output.png).

3 The role of each class

3.1 Documentation

The classes Database,QuadTree, KDTree, BoundingBox come with full documentation and tests. The documentation can be viewed in code, or through ipython: All the classes contain a main entry point such that they can be tested/viewed independantly.

```
python kdtree.py
{'index': 1, 'depth': 0, 'partition': 5, 'axis': 0}
{'index': 2, 'depth': 1, 'partition': 4, 'axis': 1}
{'index': 3, 'depth': 1, 'partition': 2, 'axis': 1}
{'index': 4, 'depth': 2, 'elements': array([1, 2]), 'axis': 0}
..
```

4 Explaining the steps

4.1 Read all the classes documentation

Before starting to implement the missing functions take some time to read through the classes: Database,QuadTree, KDTree, BoundingBox.

4.2 Implementing the QuadTree

The QuadTree is a recursive datastructure, where in each new iteration, the current BoundingBox needs to be split into 4 equal sized BoundingBoxes. These newly formed boxes should fill the exact space as their predecessor.
To test your QuadTree:

```
python3 plot_kdtree.py --quadtree 4 --quadshow True
```

and you should see an 8x8 chessboard pattern.

4.3 Implementing the KDTree

The KDTree is used to partion the datapoints into several BoundingBoxes, after which the tree can be queried to obtain a collection of these datapoints. The range-query, find all BoundingBoxes/Leaf-nodes that intersect with a given BoundingBox and return their datapoint collection, is already implemented. However the closest function is still left for the student. The closest function will traverse through the tree similar to that of the rquery instead it operates an a single point $[x, y]$, and can thus only traverse left or right. After reaching the leafnode, closest should return all the datapoints within the leafnode.

Verify your changes by testing:

```
python plot_kdtree.py --closest "3 2"
```

4.4 Using the QuadTree depth to subsample the KDTree

In the final section we will need to add/update the field ‘quad‘ for every record inside our database. This field represents whether or not a record should be sampled. For example: if we wish to sample all records up to quad-level 4, then we simply check whether or not a record his ‘quad‘ field is lower or equal to 4 (The higher the level the more data).
The student needs to add code in ‘plot_kdtree.py‘ at:

#To be implemented by the student

The following steps are needed:

- initially set the quad field for all records to the max quad-level
- iterate through all levels of the quadtree in reverse order. (maxdepth to 0)
- for every box inside that level obtain its centroid.
- use this centroid to obtain the list of closest points inside the KDTree
- find the closest point inside that list
- update the ‘quad‘ field corresponding to that point to the current quad-level

5 Final result

Executing the final program with differen values of ‘quadtree‘ and ‘quadlevel‘:

```
python3 plot_kdtree.py
        --filename ../data/addressen_groningen/adressen_groningen.shp
        --bbox-depth 8
        --max-depth 9
        --plot kdtree-bb
        --quadtree 8
        --quadlevel 4
```

should result in sampling as in figures(4,5).