**Homework 2: Implementation of KDTree and Octree**

**This assignment includes three tasks:**

(1) implement the construction and search of KDTree/Octree, including KNN search method and radius search method.

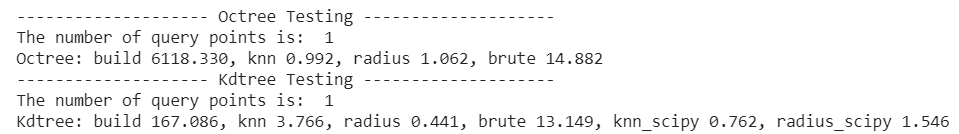
(2) compare running time of KDTree, Octree, Numpy brute-force search, and scipy.spatial.KDTee.

(3) optimize the implementation of KDTree/Octree; or tune the parameters to improve running time.

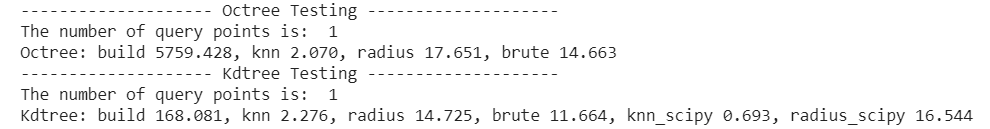
**Experimental results:**

To investigate the effect of the number of query points on running time for a specific search algorithm, different query point amounts are selected, which are 1, 10, 100, 1000, 5000, 10000, respectively. The results are shown below.

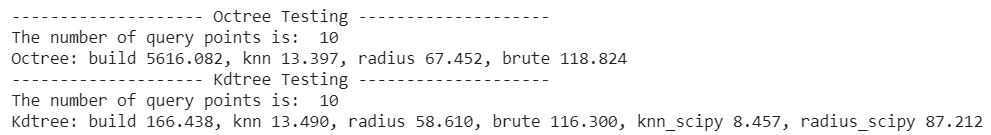
**N = 1 (first data point in point cloud data)**:



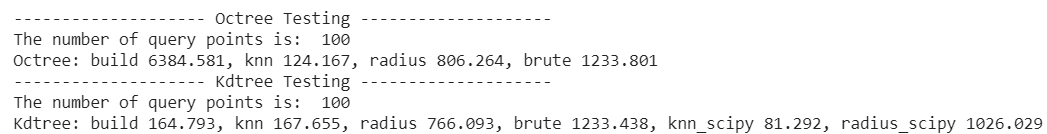
**N = 1 (randomly selected)**:



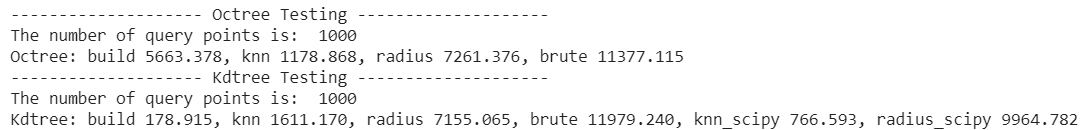
**N = 10 (randomly selected)**:



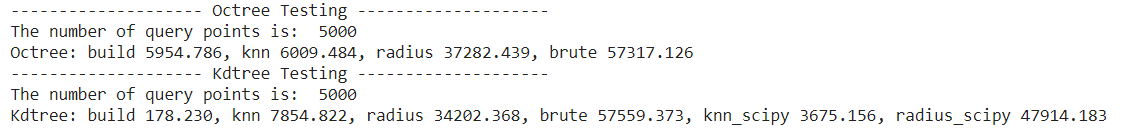
**N = 100 (randomly selected)**:



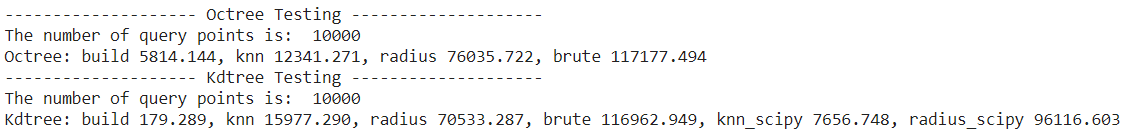
**N = 1000 (randomly selected)**:



**N = 5000 (randomly selected)**:



**N = 10000 (randomly selected)**:

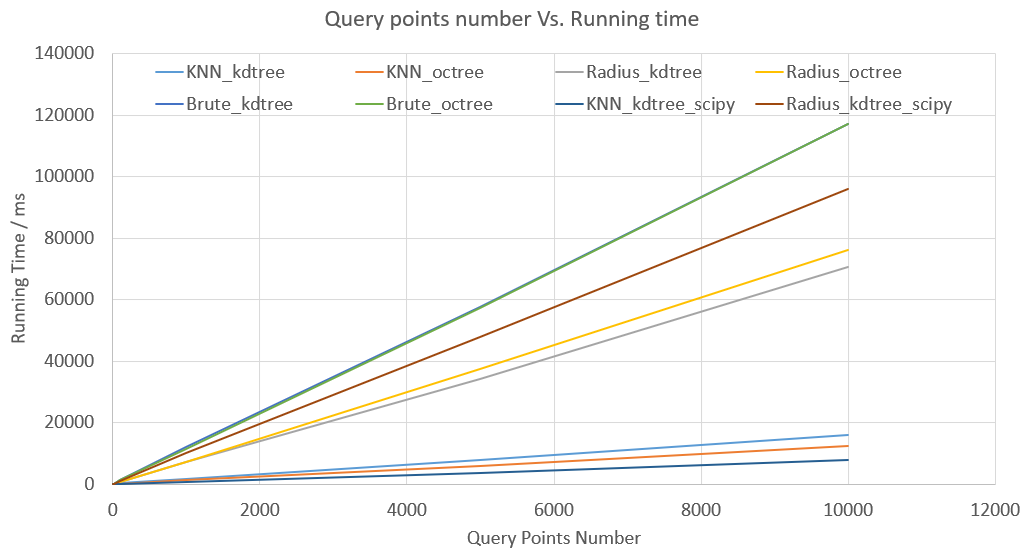


The results are summarized in the following table.

Table 1 Performance of different search algorithms with different query points number

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N=1 | | N=10 | | N=100 | |
| **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** |
| Build tree | 168.081 | 5759.428 | 166.438 | 5616.082 | 164.793 | 6384.581 |
| KNN search | 2.276 | 2.070 | 13.490 | 13.397 | 167.655 | 124.167 |
| Radius search | 14.725 | 17.651 | 58.610 | 67.452 | 766.093 | 806.264 |
| Brute force | 11.664 | 14.663 | 116.3 | 118.824 | 1233.438 | 1233.801 |
| KNN\_scipy | 0.693 | - | 8.457 | - | 81.292 | - |
| Radius\_scipy | 16.544 | - | 87.212 | - | 1026.029 | - |
|  | **N=1000** | | **N=5000** | | **N=10000** | |
|  | **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** |
| Build tree | 178.915 | 5663.378 | 178.23 | 5954.786 | 179.289 | 5814.144 |
| KNN search | 1611.170 | 1178.868 | 7854.822 | 6009.484 | 15977.29 | 12341.271 |
| Radius search | 7155.065 | 7261.376 | 34202.368 | 37282.439 | 70533.287 | 76035.722 |
| Brute force | 11979.240 | 11377.115 | 57559.373 | 57317.126 | 116962.949 | 117177.494 |
| KNN\_scipy | 766.593 | - | 3675.156 | - | 7656.748 | - |
| Radius\_scipy | 9964.782 | - | 47914.183 | - | 96116.603 | - |

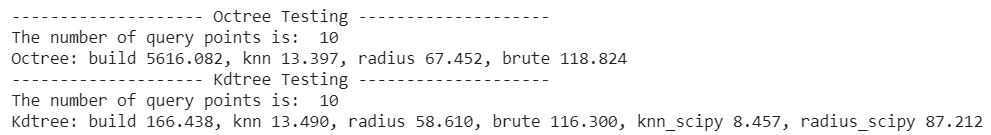
The dependency of running time on the number of query points is plotted as follows.



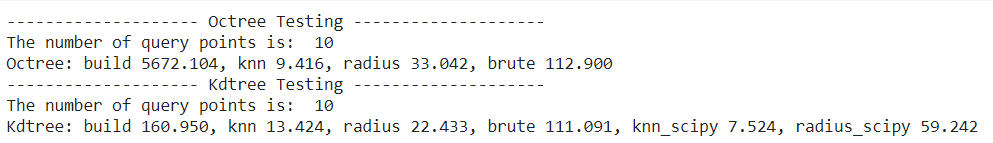
As shown in the figure, it is obvious that brute force search is the worst one among all the methods. With the increase of query points, the query speed becomes slower (higher running time) when compared to other algorithms. It can be observed that the query speed is slower in radius search than that in KNN search for the results from KDTree, Octree, and Scipy library. Radius search in KDtree outperforms the search in Octree and Scipy library. In contrast, KNN search performs better in Octree than that in KDTree. It is expected that the result in Scipy library is better than that in KDTree or Octree because the splitting method in current implementation is simple and needs to be optimized.

Normally, the query speeds from radius search and KNN search should be close. But this is not case as shown in the figure. To figure out how the big difference occurs, various radius is utilized to explore the effect of radius on the running time of KNN search and radius search. At a fixed query point number (N = 10), different radii are selected, which are 1, 0.5, 0.1, 0.05, 0.01, respectively. The results are shown below.

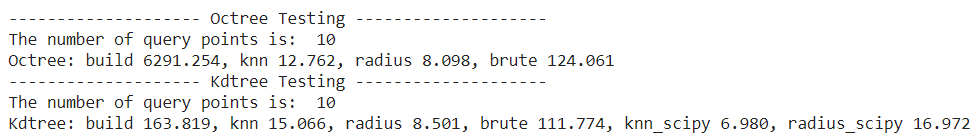
**N=10, R=1:**



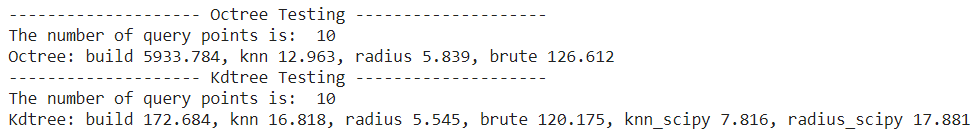
**N=10, R=0.5:**



**N=10, R=0.1:**



**N=10, R=0.05:**

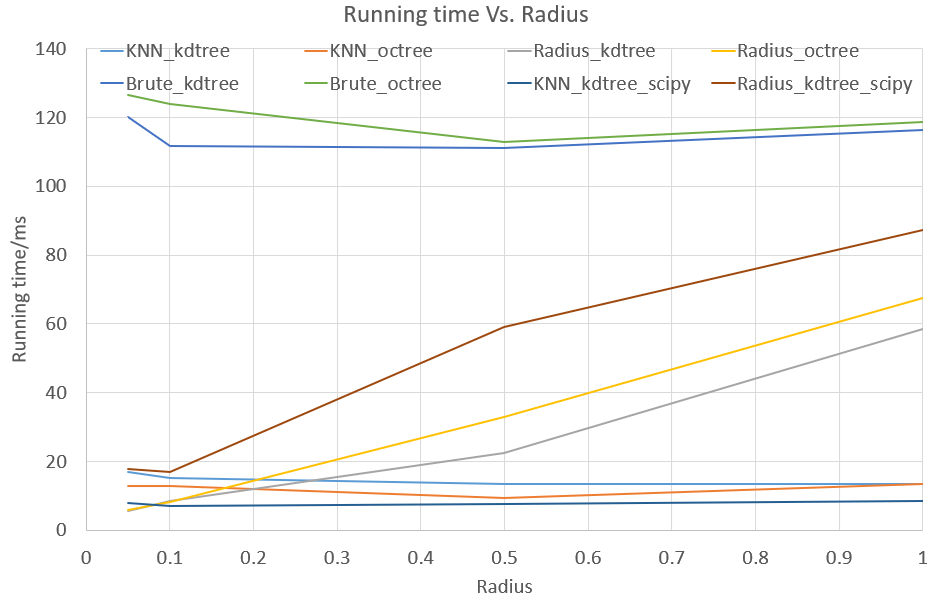


The results are summarized in the Table 2.

Table 2 Performance of different search algorithms with different search radii

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | R=1 | | R=0.5 | | R=0.1 | | R=0.05 | |
| **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** | **Kdtree(ms)** | **Octree(ms)** |
| Build tree | 166.438 | 5616.082 | 160.95 | 5672.104 | 163.819 | 6291.254 | 172.684 | 5933.784 |
| KNN search | 13.490 | 13.397 | 13.424 | 9.416 | 15.066 | 12.762 | 16.818 | 12.963 |
| Radius search | 58.61 | 67.452 | 22.433 | 33.042 | 8.501 | 8.098 | 5.545 | 5.839 |
| Brute force | 116.3 | 118.824 | 111.091 | 112.9 | 111.774 | 124.061 | 120.175 | 126.612 |
| KNN\_scipy | 8.457 | - | 7.524 | - | 6.980 | - | 7.816 | - |
| Radius\_scipy | 87.212 | - | 59.242 | - | 16.972 | - | 17.881 | - |

The dependency of running time on the number of query points is plotted as follows.



As seen from the results, the radius value has a significant effect on the running time. The running time increases as the radius becomes larger. This increasing trend can be observed in the radius search of KDTree, Octree, and KDtree\_scipy. In addition, the radius value has little influence on KNN search and brute force search.

**Conclusions**

Based on above observation, it is obvious that brute force method definitely cannot be used for querying nodes efficiently with the increase of query points. The running time for KNN search is smaller than that for radius search. In terms of the data structure, the construction of Octree (~6000ms) is slower than that of KDTree (~16ms) while the query efficiency of the former is higher than that of the latter, especially when the number of query points is huge. Furthermore, the radius of value has a great effect on the query time which increases as the radius increases.

In a nutshell, when the number of query points is small, both KNN search and radius search in KDTree can be used; when the number of query points is large, it is preferable to leverage the search method in Octree.