INFS7205

Advanced Techniques for High Dimensional Data

Student ID: s45748590

Student Name: Guangyu ZHANG

Table of content

Abstract

In this report, I am going to demonstrate the result of my experiments on the Ranked Skyline Query using Block Nested Loop (BNL) and Branch and Bound Skyline Algorithm (B&B). And I am going to explain what is ranked skyline query and why using B&B to do ranked skyline is important. I'm going to explain how I implement the algorithm and do the test. I will compare these two algorithms performance with different data dimension, data size and data correlation. Finally, I will draw a conclusion based on what I find.

Problem explanation and motivation

Skyline query is widely used today. Many decision support applications use skyline query to make their decisions, for example, we can find an appropriate hotel in terms of its price and distance by a skyline query.

The basic idea of a skyline query is to return the points that aren't dominated by other points in the data base, which means none of the other points in the data base has a smaller number in all dimensions than a skyline point.

Ranked skyline is one of the skyline query family. The outputs of a ranked skyline points are ranked based on a user defined function. So, it can demonstrate the result points according to user preference.

The original approach of skyline query is Block Nested Loop (BNL) algorithm which needs to analyze all point in the data set which takes a long time to compute when the data set is extremely large. While Bound Skyline Algorithm (B&B) can filter the rectangles by using r-tree and reduce the computing time. With B&B, we are able to produce results instantly and continuously, while BNL can only present result after all the data points are compared. So, B&B is critical for online/progressive algorithm. [1]

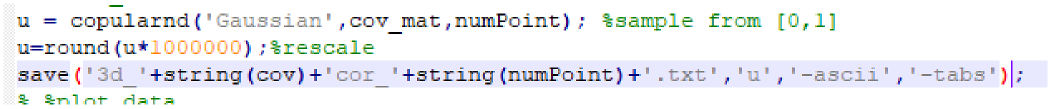
In this report, I am going to demonstrate the result of my experiment on the Ranked Skyline Query using Block Nested Loop (BNL) and Branch and Bound Skyline Algorithm (B&B). And I am going to compare these two algorithms on the ranked skyline query.

Implementation

In this project, I choose Java to implement the core algorithms as Java has plenty of packages and inbuild classes which will make the implementation more convenient. The generation of the data and the plots of the result is produced by Matlab, because it is good at data analysis.

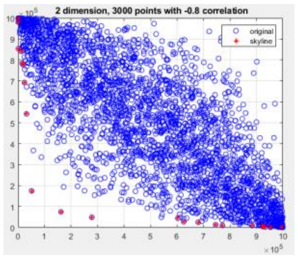
Data generation

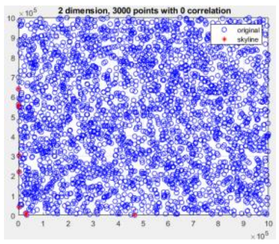
I generate data points with 2,3,4 dimension respectively. For each dimension I generate data set with different size and correlation (correlated, independent, anti-correlated). All the data sets have a uniform marginal distribution, so the analysis won't be influenced by the variance.

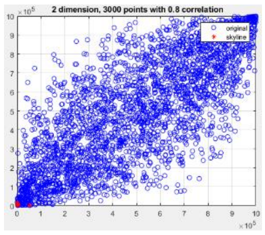


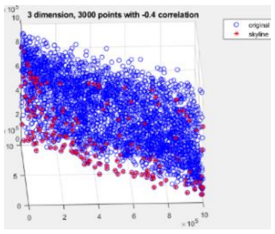
```
u = copularnd('Gaussian',cov_mat,numPoint); %sample from [0,1]
u=round(u*1000000);%rescale
save('3d_'+string(cov)+'cor_'+string(numPoint)+'.txt','u','-ascii','-tabs');
% %plot data
```

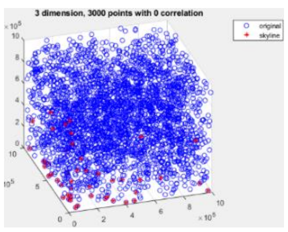
To have a better understanding of the data I use, I plot six plots to demonstrate how the data points look like visually,

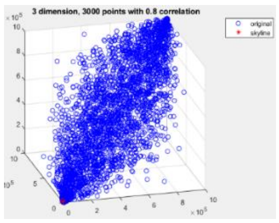










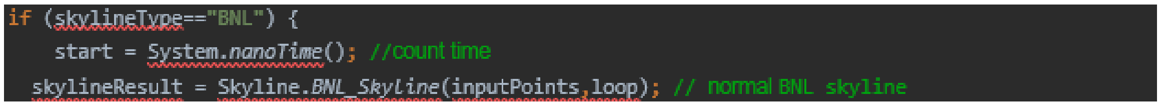


The above graphs show how data distribute visually when the dimension equals 2 and 3. The red points are the skyline points. The first row of graphs shows the distribution of 2d data when correlation equals -0.8, 0, 0.8. The second row of graphs shows the distribution of 3d data when correlation equals -0.4, 0 , 0.8.

BNL with ranked skyline query

As BNL does't support ranked skyline query directly, the implementation of a ranked skyline query using BNL algorithm need two steps. The first step is using BNL to extract all the skyline points into a list. The second step is sorting these skyline points in the order of user preference function. [2]

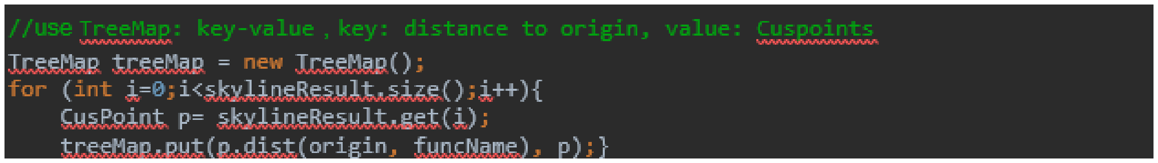
For the BNL step Figure 1, I use an arrayList to store the candidates of the result. I iterate the whole data set. In each iteration, if a point is not dominated by any point in the candidate list, it is added to the candidate list. Otherwise, it is discarded. [3] If we assume there are m skyline points, and the total data point size is n, then the time complexity will be O(m*n). The result of this step is a sequence of the skyline data points with no order.



```
if (skylineType=="BNL") {
    start = System.nanoTime(); //count time
  skylineResult = Skyline.BNL_Skyline(inputPoints,loop); // normal BNL skyline
```

Figure 1

For the sorting step after BNL Figure 2, I used treeMap in Java as it can sort key-value pairs by the key and has an O(log(n)) time complexity. The key I set is the distance from a point to the origin calculated by user preference function. The result of this step is a sequence skyline data points sorted by user defined function.



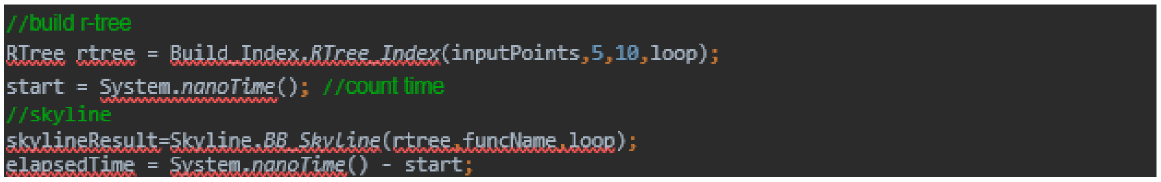
```
//use TreeMap: key-value , key: distance to origin, value: Cuspoints
TreeMap treeMap = new TreeMap();
for (int i=0;i<skylineResult.size();i++){
    CusPoint p= skylineResult.get(i);
    treeMap.put(p.dist(origin, funcName), p);}
```

Figure 2

This method is time consuming because BNL (the 1st step) needs to analyze every point in the data set. Furthermore, the sorting (the 2nd step) is also time consuming when the data size is large. Another drawback is that only after all the data points are analyzed, can we get the result.

BB with ranked skyline query

B&B also needs two steps to process Figure 3. The first step is building a r-tree index, the second is using B&B to find skyline points. B&B step can do skyline and rank the skyline points according to user preference function at the time. In the test, I set MinEntry and MaxEntry of the r-tree as 5 and 10 as I find it has the best performance after several tests. Furthermore, during the test, only the time used in the second step will counted, because we only focus on the query processing instead of the build of the index.



```
//build r-tree
RTree rtree = Build_Index.RTree_Index(inputPoints,5,10,loop);
start = System.nanoTime(); //count time
//skyline
skylineResult=Skyline.BB_Skyline(rtree,funcName,loop);
elapsedTime = System.nanoTime() - start;
```

Figure 3

To retrieve skyline points using r-tree, I use a heap to stores nodes in the r-tree sorted by preference function. There is a result list which store all the skyline points in preference order. If the first element in the heap is dominated by a point in the result list, then it is discarded. Otherwise, if it is a point then this point is a skyline point and is stored into a result list, or if it is a rectangle then its children will enter the r-tree. [1]

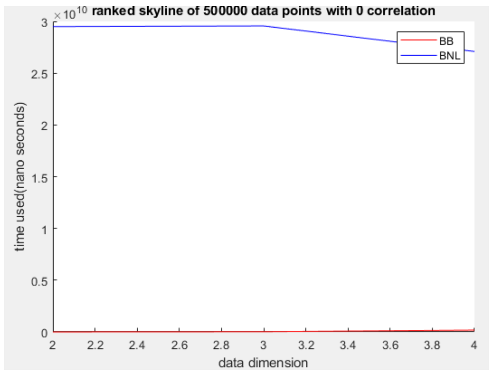
This method is time saving because the B&B can filter the rectangles and reduce the points that need to be analyzed. Furthermore, the result is sorted in the preference order and is progressive.

Test Methodology

In this section I will show how I do the test and the result of my testing.

I use the control variables technique to test BNL and B&B. In every test, I set two of the parameters as control variables and only one parameter is variable. I set data dimensions, data size and data correlation as variable separately.

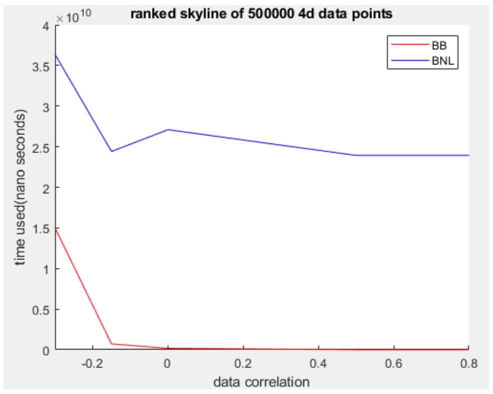
Data dimension as variable



Firstly, I test the performance of B&B and BNL with different data dimension. In the test I set data size and correlation as constant. I set data size as 500,000, correlation as 0 and test the performance of both algorithms when the data dimension is 2,3,4.

We can see that the performance of B&B is more stable and better than that of the BNL.

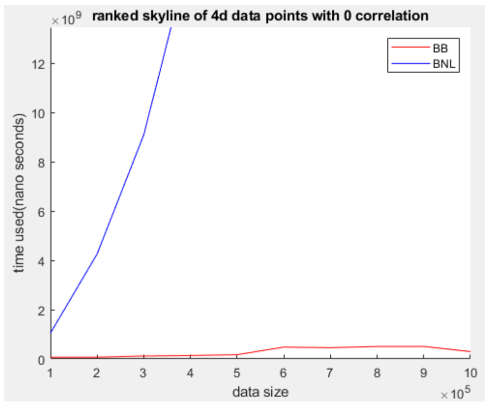
Data size as variable



Secondly, I test the algorithms with different data size. In the test I set data dimension and correlation as constant. I set data dimension as 4, correlation as 0 and test the performance of both algorithms when the data size is from 100,000 to 1000,000.

We can find that as the correlation grows, both algorithms perform better. And B&B performance is better than BNL all the time with different data correlation.

Data correlation as variable



Finally, I test the algorithms with different correlation. In the test I set data dimension and size as constant. I set data size as 500,000, data dimension as 0 and test the performance of both algorithms when the data correlation is -0.5,0,0.8.

We can see that as the data size goes up, the BNL performance increases significantly while B&B is relatively stable.

Through these three graphs we tell that the B&B algorithm always performance better than the BNL. Especially when then data size goes up.

Conclusion

To conclude, when doing ranked skyline query, the performance of the B&B overwhelms the BNL, especially when the data size increases. This is because B&B will filter the rectangles that is dominated by other points, so not all the points need to be analyzed. Furthermore, B&B can rank the skyline points with user preference while BNL can't. If we want do rank skyline query with BNL, we must sort the skyline points after BNL algorithm, which will cost lots of time.

The drawback of the B&B is the storage cost of the r-tree and the time used to build a r-tree. But it is still worth to use B&B, as the performance of B&B is remarkable.

reference

[1] Christos Kalyvas and Theodoros Tzouramanis, A Survey of Skyline Query Processing, Department of Information and Communication Systems Engineering, University of the Aegean, p18, p23

[2] Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, An Optimal and Progressive Algorithm for Skyline Queries, chapter 4.1

[3] Stephan Borzsonyi, Donald Kossmann, Konrad Stocker, The Skyline Operator, p5