**CS 615 Skyline Queries in Databases**

**Assignment 1**

**By Swapnil Mhamane (15111044)**

**Objective:**

To understand basics of non-indexed skyline queries through the implementation for BNL and SFS algorithm with simulation of disk behaviour.

**Overview:**

BNL, acronym for *Block Nested Loop* algorithm basically maintains in memory bounded windows for storing probable skyline tuple. Here objects or tuple are read from file or disk one by one tested against the tuples in window for dominance. If new tuple is dominated by any of tuple in window it is pruned. Also all tuples in window which are dominated by new tuple are pruned from window. When windows is full probable skyline tuples not dominated by any other tuple in window are moved to temporary file. At the end of input file we run algorithm on this temporary file until we find list of all skyline points in window. Time stamping is used to make space in window while processing temporary file.

SFS, acronym for *Sort filter Skyline* is extension to BNL algorithm where the data set passed to BNL algorithm is pre-processed. In pre-process step is sorted based on some monotonic preference function. Here in implementation we have used entropy function.

**Comparison between BNL and SFS:**

* As mentioned earlier SFS is an extension to BNL algorithm.
* SFS has additional overhead of sorted input data set based some monotonic preference function.
* Since such sorting make sure that upcoming data objects either get dominated by previous objects or are skyline objects but does not dominate previously processed object in window There are less chances of dummy skyline points are get stored in window or temporary files.
* As sorting in SFS reduce the number of dummy skyline points, it shows reduction in object to object comparison and so the running time.
* Basically SFS shows some kind of optimization over BNL with some additional overhead of monotonic preference function computation and sorting.

**Steps to Run attached program:**

On cmd run the following command

>java –jar Skyline.jar “dataSetFilePath” “QueryFilePath” Y

Where third argument is Y/N Boolean indicating whether to print skyline tuple to output file.

* Let’s analyse the result of two algorithms on some sample input data sets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Parameters** | **Smaller Data Set** | **Correlated**  **Data Set** | **Anti-Correlated**  **Data Set** | **Random**  **Data Set** | **Larger Data set** |
|  | **Window Size** | **2** | **100** | **100** | **100** | **50** |
|  | **No. of Objects** | **4** | **1000** | **1000** | **1000** | **3000** |
|  | **No. of Skyline Points** | **2** | **13** | **705** | **166** | **207** |
| **BNL** | **No. of Comparisons** | 4 | 1603 | 315909 | 34442 | 115177 |
| **Running Time** | 62 ms | 2016 ms | 4890 ms | 2438 ms | 5520ms |
| **SFS** | **No. of Comparisons** | 3 | 1101 | 265134 | 17201 | 31174 |
| **Running Time** | 516 ms | 1950 ms | 3032 ms | 2000 ms | 4484 ms |

**Observations:**

* **Kind of Data:**
  + As we can observe from the result table that as the correlation between data set goes on decreasing no. of skyline points, for same data set length, goes on increasing, so is the no. of object to object comparisons and execution time.
  + Since for correlated data set probability of objects getting dominated by other correlated object increases, it results in lesser no. of skyline objects.
* **No. of Comparisons:** 
  + Pre-sorting in SFS algorithm helps in *reduction of number of object to object comparison.*
  + Since this sorting *reduce* the possibility of *dummy skyline*s and that is independent of what kind of data and how much length of data set you are processing.
  + It doesn’t matter whether the data is anti-correlated, correlated or some random set, SFS dominates well on this factor.
* **Running Time:** 
  + Total execution time for finding all skyline points in data set using BNL algorithm takes longer time than SFS algorithm in general for sufficiently large data source.
  + But for *smaller data* sets additional *overhead in sorting* input lets SFS take much longer to execute than BNL.
* **Window Size:**
  + As we go on increasing the window size, scope of in memory processing increases and so the time running time goes on decreasing.

**Conclusion:**

So in for larger data sets we can say in that SFS algorithm dominates the BNL algorithm. Although there is overhead of sorting data set in SFS but it impacts is reduced by marginal reduction in no. of comparison and total execution time.