Summary of Paper The Skyline Operator

By S. Xiao Fernández Marín

1 Summary

Skyline is the set of points that are not dominated by other points.

A point dominates another one if it is "as good or better in all dimensions (dimension: each category taken into account for measuring the domination, i.e. price, distance,...) and better in at least one dimension.

In this paper the Skyline operation is integrates into the DB system. This is done by the next statement:

```
SELECT...  FROM...  WHERE...
GROUP BY...  HAVING...
SKYLINE OF [DISTINCT] d_1 [MIN | MAX | DIFF],
..., d_m [MIN | MAX | DIFF]
ORDER BY ...
```

Where $d_1, , , d_m$ is the dimensions of the Skyline. This query can save time as it is not needed to create a nested query and can be combined with some other operations with a little additional cost of computation.

As the Skyline operator explained in this paper is two-dimensional (because is the best solution to approach the performance, as if we had 3 dimensions, it would be unprofitable), there are different algorithms used to achieve the goal.

Sort based: Compare a tuple with its predecessor.

Block nested loops: Reads a set of tuples and keeps in memory a "window" of the tuples that are incompatible. When a new tuple comes, it is compared with all the others of that "window" and the new tuple can be: *Eliminated* if it is dominated by a tuple in the window. *Placed into the window* if it dominates one or more tuples in the window, those tuples are eliminated and the new one is placed in the window. *Placed in a temporary file* if there is not enough space in the window. Later, if it is free space, it becomes part of the window.
A timestamp is assigned to each tuple stored in memory to keep track of the comparisons, so after the output, the tuples with a lower timestamp than the minimum of the temporary file get erased. There are different variants of the algorithm like the self-organizing list (for the window) or replacing tuples in the window (to keep the dominant tuples in the window).

Complexity: Best case=$O(n)$; Worst case=$O(n^2)$. n=number of tuples.

Divide and conquer: From one dimension, compute the median m and get two partitions, One with the tuples better than the median and another one with the other half. Now, recursively do this to both parts until a partition contains a few tuples.
After all the parts have been split, eliminate the tuples that are dominated by some other tuple and later merge two sub-partitions.
Divide and conquer has also had different variants like M-way partitioning (instead of splitting in two, it split it into m partitions) and early skyline (when the data is split into m parts and load as many tuples as the memory can handle).

Complexity: Best case=$O(n(logn)^{d-2})+O(nlogn)$; Worst case=$O(n(logn)^{d-2})+O(nlogn)$. n=number of tuples, d=number of dimensions.

Index based: Using R-trees or nearest neighbours. In this way, we want to find the tuples that are useful and prune the branches that are dominated by those tuples.
The operators can also be pushed through a join so it can be used an index to compute the Skyline so it reduces the size of the result and makes the join cheaper.

After studying the performance in all the algorithms, a summary of an experiment varying the dimensions is done using 10MB of data and a memory buffer of 1MB. Three kinds of DB have been generated: *indep* (all values are independently generated using a uniform distribution), *corr* (points well in one dimension also good in the other dimension) and *anti* (points good in one dimension, bad in the other one).

The Skyline is small the *corr* one, it increases sharply in the *anti* one and it stays in the middle the *indep* one.

We are also shown a table with the running times and amount of disk I/O an algorithm takes to compute in a 1MB buffer. The block nested loop (BNL) algorithm is the best one as the window has space enough to end after just 1 iteration. Being the divide and conquer (D&C) and its variants the worst ones because of their high I/O demands.

In the multi-dimensional view, the BNL is good with small dimensions, where, after that, they are very time-consuming. The D&C are better than the BNL, in particular the D&C-mway partitioning. This is why the system should run this solution or BNL-self-organizing list.

2 Questions not answered by this text

It can also be implementations of the Skyline operator using B-Trees, which can be very useful in small Skylines.

It is a very important part of multiple decision making but it can not be applied in a distributed environment.

3 What has changed since this was written

In parallel computing, there have been new investigations, where first, redundant data is pruned and after that a rotation scheduling plan based on spatial indexes. This ensures that a load of each node is balanced and avoids bottleneck nodes.