**M.Sc. Data and Web Science 2022-2023**

**Technologies for Big Data Analytics**

“Scalable Processing of Dominance-Based Queries”

Ippokratis Kotsanis  
 Dept of Computer Science   
 Aristotle University of Thessaloniki  
 Thessaloniki, Greece  
 ippo.kotsanis@gmail.com

Filitsa-Ioanna Kouskouveli  
 Dept of Computer Science   
 Aristotle University of Thessaloniki  
 Thessaloniki, Greece  
 filiannakou@gmail.com

Charalampos Moutafidis  
 Dept of Computer Science   
 Aristotle University of Thessaloniki  
 Thessaloniki, Greece  
 charismoutafidis@gmail.com

**ABSTRACT**

In this project, we worked with multi-dimensional data and given a potentially large set of d-dimensional points, where each point is represented as a d-dimensional vector, we had to detect interesting points. The project is based on the concept of dominance. We say that a point p dominates another point q, when p is as good as q in all dimensions and it is strictly better at least in one dimension. It is assumed small values are preferable.

For instance, the point p(1, 2) dominates q(3, 4) since 1 < 3 and 2 < 4. Likewise, p(1, 2) dominates q(1, 3) since, although they have the same x coordinate, the y coordinate of p is smaller than the one of q. There were three different tasks needed to be completed:

Task1: To return the set of points that are not dominated, given a set of d-dimensional points. This set of points is also known as the skyline set.

Task2: To return the k points with the highest dominance score, given a set of d-dimensional points. The dominance score of a point p is defined as the total number of points dominated by p.

Task3. To return the k points from the skyline with the highest dominance score, given a set of d-dimensional points.

∗Scalable Processing of Dominance-Based Queries.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

*Ethnikis Antistasis,16. February, 2023, Kalamaria, Thessaloniki, Greece GR*

© 2023 Copyright held by the owner/author(s). 978-1-4503-0000-0/18/06...$15.00

After defining the number of data points, that would comprise each dataset, along with each dataset’s dimensions, we generated different datasets. Each dataset’s points would follow one of four distinct distributions. The four distributions were uniform, normal, correlated and anti-correlated. The first idea to solve the problem was inspired by this paper [1]. The concept was to sort the dataset in non-ascending order of sum of point’s dimension values. The thought was abandoned and we selected the most reliable method, the Brute Force method to solve these three tasks. In conclussion, the main comparison in performance was covered by two parts. The first one was to compare performance via the number of dataset’s data points and the second one was via the number of threads used to complete each task.

1. **INTRODUCTION**

**1.1 Problem**

The problem consists of three main tasks. In the first task, given a d-dimensional dataset, a point p is said to dominate another point q if it is better than or equal to q in all dimensions and better than q in at least one. A skyline is a subset of points in the dataset that are not dominated by any other points. Skyline queries, which return the skyline points, are useful in many decision making applications involving high dimensional datasets.

The second task is discribed by finding the k points with the highest dominance score. The dominance score of a point p is defined as the total number of points dominated by p.

In the third task the requirement was to find the k points from the skyline, of a given set of d-dimensional points, that have the highest dominance score.

An example is shown in Figure 1. The dominating region of the top-1 point is shown gray. Any point that falls inside this region is dominated by top-1. Based on the Figure the top-1 point has a domination score of 8, since it dominates 8 other points.

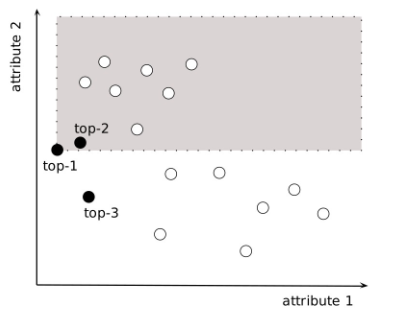


Figure 1. Example of a top-3 dominating query.

**1.2 Basic requirements**

Given a dataset of d-dimensional points we should implement scalable and efficient algorithms to solve the aforementioned tasks. Our algorithm was implemented in the Scala programming language and we also used Apache Spark. Note: the parameter d (number of dimensions) depends on the dataset, whereas the parameter k is user-defined and must be given as an input to the algorithm - program (for Task 2 and Task 3).

In general, the point coordinates are double numbers, so coordinate values are treated as doubles. We provide results for different values of k, different dimensionalities, different data distributions and different data cardinalities.

Examples of data distributions in the 2-d space are given in Figure 2.1 and Figure 2.2.

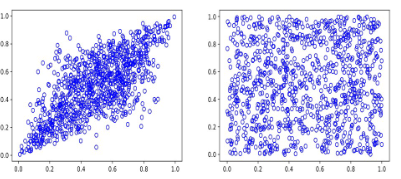


Figure 2.1. Different distributions for 2-d (from left to right): correlated, uniform.

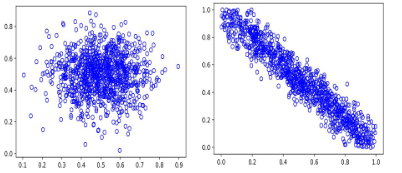


Figure 2.2. Different distributions for 2-d (from left to right): normal, anti-correlated.

**1.3 Data distribution - manipulation**

In order to create different datasets, to later examine them in Tasks 1, 2 and 3, Python language was employed. More analytically, *pandas* and *numpy* libraries were utilized to produce the datasets, while *matplotlib.pyplot* module was utilized to check a datasets’ compliance with a (specific) distribution in two and three dimensions.

Datasets of uniform, normal, correlated and anti-correlated distributions were generated, for each one of the following dimensions: 2, 3, 4, 10 and 20 and for a number of: *1,000*, *10,000*, *25,000*, *50,000*, *100,000*, *250,000*, *500,000* and *1,000,000* data points, respectively. Hence, for each one of the abovementioned distributions there are forty different datasets. It should be mentioned that all coordinates, irrespectively of distribution, are doubles, rounded in to four digits.

As far as it concerns the uniform distribution, the lower boundary of the output interval was set to 0 and the upper boundary of the output interval to 1. On top of that, it is worth noting the mean of all normal distribution datasets was set to 0 and standard deviation to 1. In the direction of generating correlated and anti-correlated distribution datasets, not one but d-means are required. In addition to the d-means, a covariance matrix is necessary to be constructed. The mean vector of d-dimensions includes d means, each one of which was set to 0. Covariance matrices are randomly formed, however, *np.random.seed()* was set to 1, to have “reproducible” results.

**2. METHODS**

The project was implemented using the Scala programming language, combined with the Spark framework. The first part of the implementation was the initialization of the Spark instance. For simplicity purposes, we deployed a Spark instance that was initialized on runtime and run on the fly.

For our experiments, we run our implementation with different Spark configurations, ranging from 1 to 16 threads. At this point, it should be pointed out the machine -in which the test was run- has an 8-core Intel Core i9, 2.3GHz. The second part of the implementation was the parsing of the previously created data as well as their transformation, in the interest of fitting the needs of the application.

Another important thing to disclose, here, would be the function called *i\_dominates\_j*. This function checks if the first data point parameter dominates the second one.

Having implemented all the above, the needed data are in the desirable form, so we proceed with the implementation of Task 1.

For Task 1, we, initially, created a cartesian product of the data points and then eliminated the main diagonal entries. Afterwards, we run the *i\_dominates\_j* function for every entry (i, j) and we created a new map in the form of (i, flag) -where flag is 0 if i does not dominate j or 1 if i dominates j-. Finally, we reduced the result by key and kept the entries of data points not dominated by any other data point, aka skyline.

In our solution for Task 2 we created a cartesian product of the data points and eliminated the main diagonal entries, as in Task 1. Subsequently, we run the *i\_dominates\_j* function for every entry (i, j) and we created a new map in the form of (i, flag) -where flag is 0 if i does not dominate j or 1 if i dominates j-. However, after reducing the result by key, we kept all the results. In contrast to Task 1, we are interested in dominance score of each one of the data points, for Task 2.

For Task 3, we combined the results of Task 1 and Task 2, by grabbing the sorted result of Task 2 and keeping the k first points that also existed in the skyline.

For every Task assigned, we computed the time required for execution, conductive to produce measurable results.

**3. RESULTS**

The execution time results for the experiments were collected in a data sheet to draw useful results. We run the above tasks from 1 to 16 threads, for data points from 1.000 to 1.000.000, and for data points with dimensions from 2 to 20.

The first useful piece of information we extracted from the results proved the more dimensions the dataset has, the more time the software takes to compute the result.

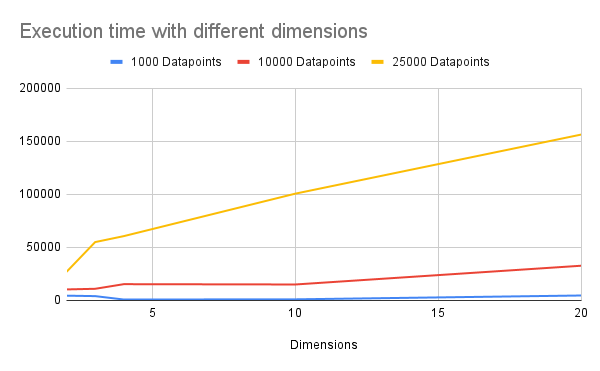


Figure 3. Execution time difference chart, comparing different dimensions. The above execution was done using 16 threads.

The second useful finding confirmed that with more threads, the execution time of every task was proportionally decreased. A meaningful observation tuned out to be that for very small datasets, for example 1.000 data points, the execution time of the program for 16 cores was actually higher than the execution time of the program for 2 cores. Thus, we can conclude Spark execution adds a noticeable overhead when the problem is trivial and easy to solve.

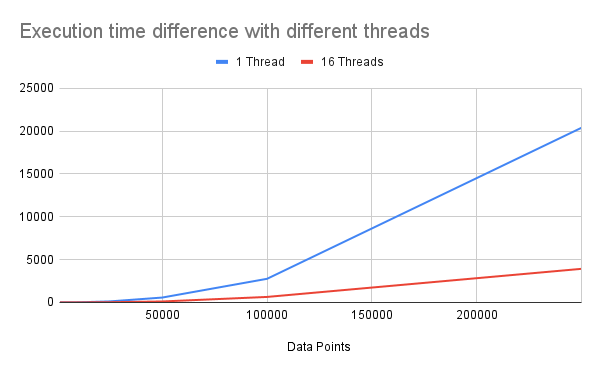


Figure 4. Execution time difference chart, comparing different thread number used for the execution.

We did not see any noticeable difference in runtime when considering different dataset correlations, as the approach is exhaustive.

**4. CONCLUSIONS**

Given a multi-dimensional dataset, we attempt to identify interesting points, hence, points which are not dominated by any other or have a very high dominance score. Small values are considered better. Primarily, we obtain data points which compose the skyline set. Then, we calculate the dominance score for each one of dataset’s data point. At last, we are interested to locate the k skyline data points with the highest dominance score.

The examined datasets follow different distributions, have different size and different dimensions. Nonetheless, taking into account our approach was brute force, any differences in execution time and completion time can be attributed to factors as:

1. datasets size in terms of rows’ number and columns’ numbers
2. number of threads used to calculate the results

It is worth observing that Spark’s overhead is not to be overlooked in cases of not computationally demanding problems.

**5. KEYWORDS**

Dominance, Scala, Spark, Skyline, Data points, Scalable, Dimensions, Threads, Distribution, Queries.

**6. CODE REFERENCES**

[**https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html#sphx-glr-gallery-mplot3d-scatter3d-py**](https://matplotlib.org/stable/gallery/mplot3d/scatter3d.html#sphx-glr-gallery-mplot3d-scatter3d-py)

[**https://stackoverflow.com/questions/35316728/does-setting-numpy-arrays-to-none-free-memory**](https://stackoverflow.com/questions/35316728/does-setting-numpy-arrays-to-none-free-memory)

[**https://sparkbyexamples.com/numpy/how-to-use-numpy-random-uniform-in-python/**](https://sparkbyexamples.com/numpy/how-to-use-numpy-random-uniform-in-python/)

[**https://chris35wills.github.io/courses/PythonPackages\_matplotlib/matplotlib\_scatter/**](https://chris35wills.github.io/courses/PythonPackages_matplotlib/matplotlib_scatter/)

[**https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html**](https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html)

[**https://www.digitalocean.com/community/tutorials/python-convert-numpy-array-to-list**](https://www.digitalocean.com/community/tutorials/python-convert-numpy-array-to-list)

[**https://www.geeksforgeeks.org/numpy-random-rand-python/**](https://www.geeksforgeeks.org/numpy-random-rand-python/)

[**https://stackoverflow.com/questions/619335/a-simple-algorithm-for-generating-positive-semidefinite-matrices**](https://stackoverflow.com/questions/619335/a-simple-algorithm-for-generating-positive-semidefinite-matrices)

[**https://stackoverflow.com/questions/16024677/generate-correlated-data-in-python-3-3**](https://stackoverflow.com/questions/16024677/generate-correlated-data-in-python-3-3)

[**https://stackoverflow.com/questions/4174941/how-to-sort-a-list-of-lists-by-a-specific-index-of-the-inner-list**](https://stackoverflow.com/questions/4174941/how-to-sort-a-list-of-lists-by-a-specific-index-of-the-inner-list)

[**https://stackoverflow.com/questions/3724551/uniqueness-for-list-of-lists**](https://stackoverflow.com/questions/3724551/uniqueness-for-list-of-lists)

[**https://stackoverflow.com/questions/30346356/how-to-sort-list-of-lists-according-to-length-of-sublists**](https://stackoverflow.com/questions/30346356/how-to-sort-list-of-lists-according-to-length-of-sublists)

[**https://realpython.com/python-sort/**](https://realpython.com/python-sort/)

**REFERENCES**

[1] Finding k-Dominant Skylines in High Dimensional Space, 5-6,

https://www.comp.nus.edu.sg/~atung/publication/k\_dominant.pdf

[2] F.N.M Surname, Article Title, https://www.acm.org/publications/proceedings-template.

[3] F.N.M Surname and F.N.M Surname, 2018 Article Title, The title of book two (2nd. ed.). Publisher Name, City, State, Country.

[4] Ian Editor (Ed.). 2018. *The title of book two* (2nd. ed.). University of XXX Press, City, Chapter 100. DOI: <http://dx.doi.org/10.1000/0-000-00000-0>.