Data mining

Experimenting on DBSCAN

Team members:
Andrea Armani - 000497267
Dimitrios Tsesmelis - 000493148
Haftamu Hailu - 000472133
Ricardo Holthausen Bermejo - 000497802

Professor:
Mahmoud Sakr

Understanding of the publication

The publication followed in order to implement this parallel version of DBSCAN has been "A faster algorithm for DBSCAN", the Master Thesis by Ade Gunawan. As it states, the task of clustering has been a rather relevant Data Mining field, as several approaches and algorithms have been devised throughout the last decades. Among them we find the DBSCAN algorithm, which stands out due to features such as the ability of finding arbitrarily-shaped clusters, or its robustness to outliers. Notwithstanding the capabilities and advantages of DBSCAN approach, its complexity for a worst-case scenario is $O(n^2)$, which can be prohibitive regarding large datasets. For this reason, several modifications of the original DBSCAN algorithm have been devised, either trying to reduce the number of range queries (IDBSCAN, FDBSCAN) or partitioning the dataset (GF-DBSCAN, GriDBSCAN). Nonetheless, the results obtained from these algorithms are different from those yielded by DBSCAN (regarding the border points) or, when there is no difference regarding the results, the theoretical running time remains the same. Thus, by proposing “a Faster Algorithm for DBSCAN”, the publication followed for the present assignment aims to address this issue.

Implementation

We implemented both algorithms using Python as the programming language and we experimented by running the algorithms in Google Collaboratory (Google Colab). Google Colab is a cloud technology that provides free Jupiter notebooks that are executed on a high-end machine, with the below specs:

- **CPU**: 1 single core hyper threaded i.e(1 core, 2 threads) Xeon Processors @2.3Ghz (No Turbo Boost), 45MB Cache.
- **RAM**: ~12.6 GB Available.
- **GPU** (not needed in our experiments): 1 Tesla K80, having 2496 CUDA cores, compute 3.7, 12GB(11.439GB Usable) GDDR5 VRAM.

Description of the main function:

Google Colab does not save long term sessions. That is why we are using Google Drive to save the datasets that we will use for testing our implementations. As a result, in the first lines of the code we are fetching the dataset from Google Drive to Google Colab environment. Subsequently, we are reading the specified file by calling the method **load_points(points_list_from_file)**. This method returns a list of points.

Description of the functions used for the algorithm:

The main idea of this paper is to reduce the complexity of DBSCAN by firstly partitioning the space, formed by the points to be clustered, in smaller sub-spaces. Then, each point of the sub-spaces is

marked as core point or not and after that the partitions can be merged into clusters. Finally, the algorithm determines the border and the noise points to form the final clusters.

In this work, we are partitioning the space by constructing a grid and assigning each data point to the grid cell it lies in. It is done by the function **constructGrid(points,eps)** that takes as arguments a set of points and a constant epsilon, and it returns a 2d array G, each cell of which containing a list of points.

The next step is to determine which points of the grid are core points. This is done by the method **determine_core_pts(G,eps,minpts)**, the arguments of which are the grid G, the epsilon and the parameter minimum points. This method iterates through all the cells of the grid and for every cell, if the number of points of this cell is > minpts, then all these points are marked as core points. Otherwise, the method further checks whether the points of the current cell have close points to the neighbor cells. To get the neighbor cells of the current cell, we are using the function **neighbour_cells(G,row,box)** that given the grid G, the row and the column of the cell, it returns a list with the neighbor cells.

After having determined the core points, we can start merging the clusters by using the function **merge_cluster(G,cluster_number)**. It iterates over the entire grid and for every cell that is not empty and is core, it tries to merge it with every neighbor cell that is also core and the distance of the 2 closest points of the cells is less than epsilon.

Finally, to determine the border points and the noise points, we are using the method …… that iterates over every p of each cell and marks it as border point of cluster c, if it is not core point and there is at least one core point in the neighbors of p that belongs to cluster c. All the points that are neither core nor border points, are marked as noise points.

Experimental comparison with DBSCAN

User guide