2D-Clustering through Approximation Method using Geometric Calculation

Jolan Philippe
School of Informatics, Computing and Cyber Systems
Northern Arizona University, Flagstaff, AZ, U.S.A.
jp2589@nau.edu

Michael Gowanlock
School of Informatics, Computing and Cyber Systems
Northern Arizona University, Flagstaff, AZ, U.S.A.
michael.gowanlock@nau.edu

***Abstract*—Recently, the size of the data largely increased. Consequently, this increase in dataset size substantially increases the time-to-solution of many well-known data analysis algorithms and tools. Data analysis is a large-scale science and includes challenges to existing architectures. In particular, one data analysis tool is clustering, which finds groups of similar data points in a feature space. Different approach already exists. As a concrete example, detecting galaxy clusters or classifying stars are challenges in astronomy.**

Given that clustering is computationally expensive, we explore an approximate clustering solution, where we will examine the trade-off between clustering accuracy and algorithm performance. To achieve this, we index the dataset in a grid of constant size-defined cells. Each cell is considered as a cluster entity. From points in a cell, we calculate the probability of being in the same cluster of the neighbor cells.

***Index Terms*—DBSCAN, Parallel Clustering, Approximation, Large datasets, Overlapping area.**

I. Introduction

The constant increasing size of data challenges the developers to improve their computational techniques [1]. This "data revolution" is due to continually improved hardware which is more precise and efficient. These data are exploited using computational algorithms in many scientific fields. The term computational science describes scientific and engineering inquiry in which the computer plays an essential role (e.g., computational biology [2]), even if some fields are more mature than others. Make clusters from a large set of data in a reasonable amount of time remains a hard problem to solve. Make a group of similar data is used in many scientific applications, and many different approaches already exist, depending on the type of data, and the wished precision [3].

Many domains use Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [4]. The performance of the DBSCAN algorithm has been improved thanks to different techniques such as parallel implementation based on indexing techniques [5], [6]. The points are clustered using two parameters: (i) ϵ, an arbitrary distance used to find the neighborhood of a point, and (ii) $minPts$, the minimum number of points within ϵ-distance to create a cluster. The most common approach is to compare the distance points to points to define which ones can be in the same cluster. This method gives exact results, but the cost of computation is expensive. There already exists different optimizations that considerably reduce the cost of calculation. This paper studies a new reduction of calculation time with an approximation of a solution for creating clusters with geospatial data in two dimensions. This approach is based on the intersection of the ϵ-area around points of cells, and their neighbors. This paper makes the following contribution.

- We propose formulas to calculate the intersection area of a circle and cells of a grid, based on circle segment and trigonometry theory;
- Using the defined equations, we present an approximation of DBSCAN. The clusters are represented by cells and not by points anymore.

The organization of the paper is as follow. In Section II, we describe the DBSCAN algorithm, with related works such as already existing optimizations. Section III presents the indexing technique avoiding useless calculation. The overlapping area concept and the relevant decision are introduced in Section IV. Section V gives an overview of the algorithm used to solve the problem. Finally, Section VI and Section VII presents future works and the conclusion of the study.

II. Background

A. The DBSCAN Algorithm

The DBSCAN algorithm aims to cluster points defined in arbitrary dimensions. The algorithm uses two parameters: a distance ϵ and a minimum number of points $minPts$ which have to be within a ϵ radius to be considered as a cluster. The entry parameters are then an estimation of the point density of the clusters. The main idea is, from a point in a database, look for its ϵ-neighborhood, and check that there are at least $minPts$ points. If there is, the point is a member of

a cluster. To find every point of this cluster, we record the ϵ-neighborhood point-by-point from the ones which are already in the same cluster. If a point doesn't have $minPts$ ϵ-neighbors, it is considered as noise. The distance calculation can be made in different ways, but it used to be the Euclidian natural distance:

$$dist_{euclid}(A, B) = \sqrt{\sum_{i=1}^{dim}(x_{Ai} - x_{Bi})^2}$$

where x_{Ai} (resp. x_{Bi}) is the value of the point A (resp. B) for the i^{th} dimension. An outline of the expression of DBSCAN [4] is presented in Algorithm 1. The algorithm take several variables as input: (i) the dataset D of points to be clustered; (ii) the distance ϵ to find neighborhood of a point; (iii) the minimum number of points, $minPts$, to consider a cluster; and (iv) a function to calculate distance between points.

Algorithm 1 The DBSCAN Algorithm

```
1:  procedure DBSCAN(D, ϵ, minPts, dist)
2:      C ← 0
3:      for each points P ∈ D do
4:          if label(P) ≠ undefined then
5:              continue
6:          end if
7:          N ← NeighborSearch(D,ϵ,P,dist)
8:          if |N| < minPts then
9:              label(P) ← Noise
10:             continue
11:         end if
12:         C ← C + 1
13:         label(P) ← C
14:         S ← N\{P}
15:         for each points Q ∈ S do
16:             if label(Q) = Noise then
17:                 label(Q) ← C
18:             end if
19:             if label(Q) ≠ undefined then
20:                 continue
21:             end if
22:             label(Q) ← C
23:             N ← NeighborSearch(D,ϵ,Q,dist)
24:             if |N| ≥ minPts then
25:                 S ← S ∪ N
26:             end if
27:         end for
28:     end for
29: end procedure
```

To ensure better performance for the research neighbors, an index I of the points used to be created before the execution of DBSCAN. It is then passed as an argument of the function and is used in $NeighborSearch$. The algorithm examines all points P in D that have not been visited yet ($label(P) \neq undefined$). The neighbors of P are stored in a set N. If N is large enough ($|N| \geq minPts$), P is considering as starting a new cluster. P is a noisy element otherwise. All the neighbors are explored. Two cases can happen. A neighbor Q is either already in a cluster, and then we don't treat it, or Q is now a member of the same cluster than P. If Q is not noise, and has enough neighbor to form a cluster, its neighbors are also members of the same cluster. The output of the algorithm is a label for each point of the dataset. It is either $Noise$ or the id of a cluster.

B. Related Work

Many approaches have addressed improvement of DBSCAN performance[7], [8], [9], [10], [11]. These optimizations are made using parallelism. In [11], a MapReduce[12] implementation of DBSCAN is proposed. The program is split into two parts. First, small local clusters are made using split data distributed on the nodes. A reduction of these results are made in a second part to get bigger clusters. The clusters are first merged and then relabeled to obtain a final result. It is common to make several DBSCAN execution in science fields (e.g., space physics and aeronomy). The execution variants differ by their input parameters. It appears that several results can be reused during the variant computations. [8] presents very good optimizations for DBSCAN based on the use of commonalities on multithreading programs. Most of the optimizations are based on GPU computation to take advantage of the GPU architecture. However, [7] presents a grid-based hybrid approach of DBSCAN, using both GPUs in conjunction with multicore CPU. Two GPU kernels are used. The first one is used to compute the ϵ-neighborhood of the points without using shared memory. HYBRID-DBSCAN takes advantage of the shared memory on the GPU to page the cells, before making distance calculations. With CUDA-DClust [9], Böhm et al. take advantage of the extremely high parallelism of the GPU, and its low cost of memory transfer. The presented algorithm starts by creating chains of points in parallel on the GPU. The algorithm keeps track of collisions, that is two chains belong to the same cluster. The chains are finally merged into clusters based on the collisions. This approach is very similar than Yaobin et al.'s in [11] This main idea is reused in the Mr.Scan implementation [10], an algorithm which performs kernel optimizations by reducing host-GPU interaction. Another optimization for DBSCAN implementation based on GPU calculation is discussed in [13]. An index of the data is generally used on these different approaches to reduce the time of computation, and help to remove useless calculation. On a grid-based algorithm approach, the points are stored into cells. The points from a cell are compared to the ones in the neighbor cells. The simple approach

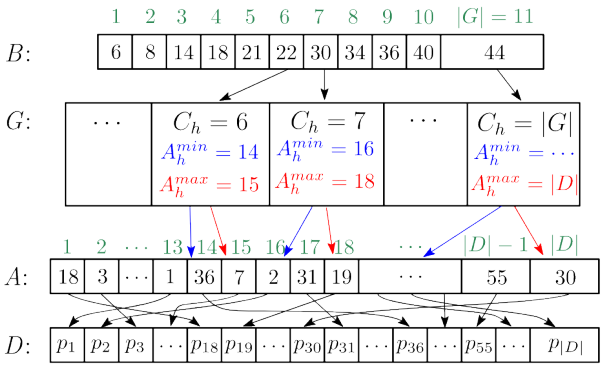

Fig. 1. Indexing of points from a dataset D. A is the lookup array to D, G the index array and B the lookup array of G

is to compare each cell with all of its neighbors. [14] presents an optimization of these comparisons, by non-duplicating the computation, based on the symmetry of the calculation. Gan and Tao proposed an approximate sequential solution [15], [16], faster to compute than DBSCAN, and with outstanding accuracy.

III. Indexing Method

Since the datasets are extensive, it is convenient to make the data more accessible. We outline in this Section an indexing method to facilitate the data access and remove useless calculation. The indexing is similar than in [7] and [14] solutions. A grid index is appropriate to distance calculation. By making cells with ϵ side squares, we are sure that all the points within ϵ-distance are contained in the direct neighborhood of the current cell. This approach is similar to the method used for trajectory calculations[17]. On a two-dimensional dataset, the points from the points dataset D are defined by two coordinates: x and y. The two-dimension grid is represented with several arrays. An example is given in Figure 1.

The array A is a lookup array of the dataset D, respecting the spatial position of the elements. In other words, the points are sorted to make them near other spatially close elements. A linearized id defines each cell. The array B contains the id of the non-empty cells of the grid. Array G, which has the same length of B is a lookup array giving information about the contained points of the cells. For each cell, contained at the position h in B, two values are defined: A^h_{min} and A^h_{max}. These values define the position in A of the points contained in the cell h.

For example, considering the example in Figure 1, the cell with the linearized id 22, at the 6^{th} position, contains the points from 14 to 15 in A, that is the points p_{26} and p_7.

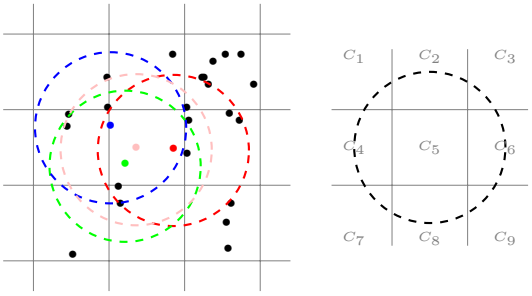

Fig. 2. Example of overlapping areas from points of cell with the cells of the neighborhood

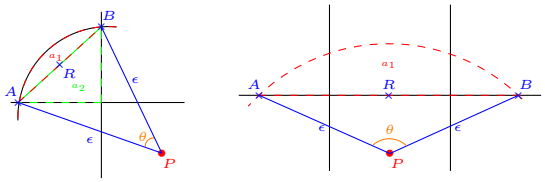

Fig. 3. Two cases of overlapping area calculation

IV. Overlapping Area

The main idea with our approach is to estimate, from a point p, the number of points within a distance ϵ. Knowing the density of a cell, we need to calculate the area contained in the ϵ-distance of the point. That is, for a cell, approximate the number of points which could be in the same cluster of p. Because the grid is constructed with ϵ side cells, we can consider every point of a cell, members of the same cluster.

The decision of merging the cluster of a given cell, and the one of a neighbor cell which contains n_c point objects are made with the following formula:

$$d = \frac{A_o}{A_c} * n_c * p$$

with A_o, the overlapping area of a circle of radius ϵ from a point and the cell, and A_c, the area of a cell, defined by ϵ^2. Because the distribution of points is not necessarily uniform, we need a probabilist constant p to make our decision more or less strict. If there exists a point within the current analyzed cell such that $d \geq minPts$, the neighbor cell is considered in the same cluster. Figure 2 shows an example of the possible overlapping areas from points within a cell.

The calculation of A_o, the overlapping area, depends on the cell position. We consider two types of cell: the corners, and the centered cells. In Figure 2, C_1, C_3, C_7, C_9 are corners, while C_2, C_4, C_6, C_8 are centered cells. Figure 3 presents two simple cases when these overlapping areas are defined. They are not necessarily defined, depending on the reference point. If one of the coordinates corresponds to a limit of the cell, the opposite neighbor wont be covered by its circle. In both cases, considering A=(x_A, y_A), B=(x_B, y_B) and

P=(x_P, y_P), R is defined by
$R = (x_R, y_R) = (\frac{(x_A+x_B)}{2}, \frac{(y_A+y_B)}{2})$. We consider h the distance from P to R. Then, the area a_1 is equal to

$$a_1 = \int_{h\ sin(\theta/2)}^{-h\ sin(\theta/2)} \int_{\sqrt{h^2-x^2}}^{h\ cos(\theta/2)} dy\ dx$$
$$\text{with } \theta = 2 * arccos(\frac{h^2}{\epsilon})$$

which can be simplified to

$$a_1 = \frac{1}{2}h^2(\theta - sin(\theta))$$

The area a_2 is calculate using the triangle area formula. Finally, the overlapping area of a corner is simply defined by $a_1 + a_2$.

Calculate the overlapping area of a centered cell is equivalent to calculate the area of global side and remove the two corner areas. For example, using Figure 2, calling A_i the overlapping area within C_i, we have the following result

$$A_2 = overlapping((C_1 \cup C_2 \cup C_3)) - A_1 - A_3.$$

V. Algorithm Overview

A. Algorithm

To design an approximation for DBSCAN, we used the indexing method combined with the overlapping area presented in Sections III and IV. The specific entry parameters of the algorithm are the grid resulting from the indexing method, and a probabilistic factor for the decision taking.

We start by making a set for each cell, to contain the linear ids of the cells which will be in the same cluster. To make clusters, every non-empty cell is considered globally. The goal is to merge cells that have enough points (greater than $minPts$), within an ϵ distance from a point, in the same cluster. If two cells appear to be in the same group, they have to merge their cluster. That is, all the cells already defined as being in the same cluster than the current analyzed one, must update their set of ids by adding the new one. At the end of the execution, each cell has a set of several ids. To identify the different clusters, the maximum linear id of each set is kept. This choice is totally arbitrary. It could be the minimum one.

An outline of the algorithm is presented in Algorithm 2. The variables taken as input are: (i) the index G as a grid; (ii) a probabilistic $factor$ for the decisions; (iii) the distance ϵ to find neighborhood of a point; (iv) the minimum number of points $minPts$. We can notice that we don't need to calculate distances anymore, then the $dist$ parameters from Algorithm 1 has been removed.

To be able to compare different solutions, the cluster ids must be standardized. For C clusters, we distribute new ids. The new ids are integers in [0;C].

Algorithm 2 Approximation of DBSCAN Algorithm

```
1:  procedure APPROXIMATE_DBSCAN(G, factor, ϵ, minPts)
2:      for each non-empty cell C ∈ G do
3:          cluster(C) ← {C}
4:      end for
5:      for each non-empty cell C ∈ G do
6:          N ← neighbors(C)
7:          for each point P ∈ C do
8:              A ← overlapping_arround(P)
9:              for i in 0 .. |A|-1 do
10:                 np ← N[i].nb_points * A[i]/(ϵ²) * factor
11:                 if np ≥ minPts then
12:                     cluster(C) ← cluster(C)
                                     ∪ {N[i]};
13:                     for each cell C_n ∈ cluster(C) do
14:                         cluster(C_n) ← cluster(C_n)
                                           ∪ {N[i]};
15:                     end for
16:                 end if
17:             end for
18:         end for
19:     end for
20:     for each non-empty cell C ∈ G do
21:         cluster(C) ← max_id(cluster(C))
22:     end for
23: end procedure
```

B. Optimizations

In the implementation of the approximation of DBSCAN using overlapping method, we made several optimizations. Because we are testing every points of a cell to calculate overlapping area with neighbors, it is possible to find several time that a cell is in the same cluster. In addition, there is no chance that an empty cell has enough point to be clustered with the same current analyzed one. Using a boolean arrays, we defined what calculation to do, and then which ones can be removed. Thanks to the index, we only iterate on non-empty cells. The second loop can also consider less cell to calculate. Considering that to start a cluster, a cell must be dense enough, we consider that the points in a cell which verifies $\frac{\pi}{4} * p * n_c < minPts$, with $\frac{\pi}{4} * 100$ the percentage of covered area of a circle of radius $\epsilon/2$ and a ϵ side square, p the probabilistic factor and n_c the number of points in the cell, are outliers. The cells containing outliers are not calculated.

C. Parallelization

In our algorithm, cells are computed one-by-one, independently. To get a more efficient implementation of the algorithm, the calculation of the neighbors cluster belonging are made in parallel. Using OpenMP, the second loop of the algorithm is parallelized. The shared

index structures is only read and does not need to create a critical section. However, the cluster set of each cell is shared must be protected.

VI. Future work

The next step of the study, is the accuracy of the algorithm. By comparing the approximate solution clusters and an exact solution, the accuracy of our implementation will can be evaluated. By evaluating the correspondence score, defined by a value in [0;1]. The objective is to obtain a correspondence score closed to 1 by an approximation. By testing different value of the probabilistic factor p, and different dataset, we will be able to obtain a better approximation. GPU calculation is way much faster than CPU calculation. Even if we don't calculate as many points than in an exact solution, the time reduction provided by GPU computing is not negligible. Finding a balance between accuracy and performances is the final goal of the project. The solution provided by Gan and Tao [15], [16] outperformed traditional DBSCANimplementation, and their results have almost a perfect accuracy. We aim to use parallel computation to get better performances than already existing solution, without reducing accuracy results.

VII. Conclusion

In this paper, we have presented a grid-based DBSCANalgorithm to cluster points in a dataset. By exploring neighbors physical proximity and density from points, we proposed an approximation of the clustering, with defining outliers, also called noise. To conclude this project, we still need to evaluate the worth of our solution. By examining the results and the performances, we want to be able to get an acceptable solution for clustering.

Acknowledgment This research was supported by Michael Gowanlock who gave the opportunity to work on something new. In addition, we would also like to show our gratitude to Benoît Gallet for providing his expertise on this project.

References

[1] A. Szalay, "Extreme data-intensive scientific computing," *Computing in Science Engineering*, vol. 13, no. 6, pp. 34–41, Nov 2011.

[2] B. Lindsay, "Unmet needs for analyzing biological big data: A survey of 704 nsf principal investigators," Oct 2017.

[3] H. Kaur and J. Singh, "Survey of cluster analysis and its various aspects," pp. 353–363, Oct 2015.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231.

[5] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, Jun. 1984. [Online]. Available: http://doi.acm.org/10.1145/971697.602266

[6] C. Xiaoyun, M. Yufang, Z. Yan, and W. Ping, "Gmdbscan: Multi-density dbscan cluster based on grid," in *2008 IEEE International Conference on e-Business Engineering*, Oct 2008, pp. 780–783.

[7] M. Gowanlock, C. M. Rude, D. M. Blair, J. D. Li, and V. Pankratius, "Clustering throughput optimization on the gpu," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2017, pp. 832–841.

[8] M. Gowanlock, D. M. Blair, and V. Pankratius, "Exploiting variant-based parallelism for data mining of space weather phenomena," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 760–769.

[9] C. Böhm, R. Noll, C. Plant, and B. Wackersreuther, "Density-based clustering using graphics processors," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09. New York, NY, USA: ACM, 2009, pp. 661–670. [Online]. Available: http://doi.acm.org/10.1145/1645953.1646038

[10] B. Welton, E. Samanas, and B. P. Miller, "Mr. scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov 2013, pp. 1–11.

[11] He, H. Tan, W. Luo, S. Feng, and J. Fan, "Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 83–99, Feb 2014. [Online]. Available: https://doi.org/10.1007/s11704-013-3158-3

[12] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[13] G. Andrade, G. Ramos, D. Madeira, R. Sachetto, R. Ferreira, and L. Rocha, "G-dbscan: A gpu accelerated algorithm for density-based clustering," *Procedia Computer Science*, vol. 18, pp. 369 – 378, 2013, 2013 International Conference on Computational Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913003438

[14] M. Gowanlock and B. Karsin, "GPU accelerated self-join for the distance similarity metric," *CoRR*, vol. abs/1803.04120, 2018. [Online]. Available: http://arxiv.org/abs/1803.04120

[15] J. Gan and Y. Tao, "DBSCAN revisited: Mis-claim, un-fixability, and approximation," in *SIGMOD Conference*. ACM, 2015, pp. 519–530.

[16] ——, "On the hardness and approximation of euclidean DBSCAN," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 14:1–14:45, 2017.

[17] M. Gowanlock and H. Casanova, "Distance threshold similarity searches: Efficient trajectory indexing on the gpu," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2533–2545, Sept 2016.