Spatial Clustering in Large Databases Using Packed X-tree

Grace L. Samson[#1], Joan Lu[*2]

#Department of Informatics, University of uddersfield, United Kingdom
1Zenhev2@gmail.com
* Department of Informatics, university of Huddersfield, United Kingdom
2j.lu@hud.ac.uk

Abstract

In this paper, we are proposing a new algorithm that improves the performance of the DBSCAN clustering algorithm using a packed X-tree. The proposed algorithm does not require the minpoints and eps values. We have extensively described how the system is achieved and we have also proposed a new effective method for finding the k- nearest neighbours of spatial objects in a large database. The study shows that the proposed method is very efficient and will greatly accelerate the operations of density based clustering in large dataset as against the existing methods.

Keywords: *Clustering; Spatial Clustering; Nearest Neighbour; DBSCAN; Big Data*

1. Introduction

The study in [1] reveals that the major challenges militating against effective management of large spatial datasets is storage utilization and computational complexity (both of which are characterised by the size of spatial big data which now tends to exceeds the capacity of commonly used spatial computing systems owing to their volume, variety and velocity). Spatial database systems incorporate space in database systems, they support non-traditional data types and more complex queries, therefore in order to optimise such systems for efficient information processing and retrieval, appropriate techniques must be adopted to facilitate the construction of suitable index structures. Though a single spatial data contains observations with locations and identify features and positions of objects on the earth's surface and they present us a framework for putting our observations on the map [2], large spatial datasets stems from scientific activities these days that tends to generate large databases which always come in a scale nearing terabyte of data size. Several access methods have been proposed to optimise spatial database systems for efficient information processing and retrieval especially in a large multidimensional spatial dataset environment using appropriate techniques to facilitate the construction of suitable index structure for these database systems. This paper describes the design of an effective system for spatial query processing for clustering of data in large datasets.

2. Related Works

[16] Suggested that one of the ways to improve the speed performance of the DBSCAN algorithm could be through the implementation of an indexing structure that can support spatial data access method and as such speeds up the neighbourhood finding operation for the DBSCAN clustering algorithm. This notion has attracted so much interest and has given rise to variation of the clustering algorithm which main aim is to build a DBSCAN algorithm using different indexing structure that can implement spatial operations.

PDBSCAN was suggested by [17], the algorithm applied a distributed R*-tree for partitioning the dataset among many computer nodes. Distributed R*-trees partition data but they replicate the entire index on each node. [18] Suggested implementing the DBSCAN algorithm based on the PP-R-tree. This variation increases in performance if the database is initially stored on an r-tree indexing structure. [19] Also proposed the P-DBSCAN, a novel parallel version of the existing DBSCAN algorithm applied in a distributed environment by implementing a priority R-tree. In [20], the K-dimensional tree (also known as the kd-tree) was applied to solve the problem of database size especially in the case where the size becomes so large (which is one of the limitations of the existing DBSCAN algorithm). Their algorithm applies a k-distance graph method to automatically calculate the values of Eps and Minpoints. [21] Applied the extended CUDA-DClust algorithm which is a block tree indexing structure to extend the functionality of the existing DBSCAN. Their DBSCAN clustering algorithm version (Mr. SCAN) is designed to handle extreme cases in density based clustering using a hybrid parallel tree-based implementation to combine a network of GPGPU-equipped nodes with an MRNet tree-based distribution network. The algorithm (Mr. SCAN) effectively partitions the point space and optimizes DBSCAN's computation over dense data regions in other to overcome the problems encountered by previous implementations. In [22] the kd-tree was also used to implement the DBSCAN algorithm. The package DBSCAN as they call it is a fast reimplementation of several density-based algorithms of the DBSCAN family for spatial data. Theses includes OPTICS (ordering points to identify the clustering structure), and the LOF (local outlier factor) clustering algorithms. The application of the kd-tree data structure was to accelerate a faster k- nearest neighbour search. The SR-tree based DBSCAN was also proposed by [23]. Their variation of the DBSCAN algorithm they claim performs optimally for determining the Eps-neighbourhood of an object (which is one of the most difficult task in running the existing DBSCAN algorithm). In other words according to them, to determine the Eps-neighbourhood of a given spatial object, its encapsulating region has to be determined, and the tree has to be traversed from the children of the object to the leaves [23].

3. How the DBSCAN Clustering Algorithm Works

As described by [24], the DBSCAN clustering algorithm is optimal, due to its efficiency in performance when used within spatial databases which otherwise can result to irregularly shaped clusters. The algorithm is density based as such scales well in finding similarity between densely populated spatial object. It is known to perform well in clustering data without prior knowledge of the number of clusters it contains and it also performs optimally in filtering noise from a dataset. The most important feature of the algorithm is that its generalised version GDBSCAN [25] can cluster point objects and spatially extended objects (based on spatial and non-spatial attributes).

Despite all these abilities, the DBSCAN has some major limitation which includes high time consumption for finding neighbourhood of a given data point [16], performance degeneration with increase in dataset size [20]. The DBSCAN algorithm clusters data points based on density. Its idea of density is based on two parameters (Eps and Minpoints). The algorithm operates by finding the Eps-neighbourhood of each given point. The Eps- neighbourhood of a point p is that set of points that are located around the Eps-distance of the point p. p is marked core point if there are at least minpoints points in its Eps-neighbourhood. Any other points could be classified as non-core points.

Again the non-core points has two distinct classification (that is a border point or a noise point). Border points are a non-core point that contains at least one core point in its Eps-neighbourhood, while noise points do not. In the DBSCAN algorithm, clusters are formed by the set of core and border points that are reachable from a particular core point. When the algorithm finds an unvisited core point it considers it a new cluster and then forms a new cluster on it using its Eps-neighbourhood. These clusters could be expanded by finding the Eps-neighbourhood of each point, which is classified in the cluster until all points that are reachable from the first core point are found.

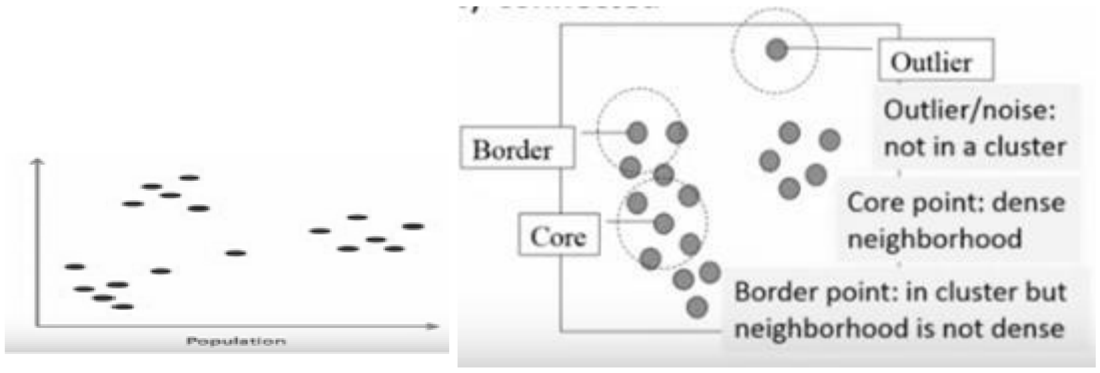
Population
Outlier
Outlier/noise: not in a cluster
Border
Core point: dense neighborhood
Core
Border point: in cluster but neighborhood is not dense

Figure 1: DBSCAN algorithm core and border points

This means that the performance of the algorithm greatly depends on the value of ***Eps-neighbourhoods*** that is chosen (see figure 1).

A. Existing DBSCAN Algorithm

The algorithm takes as an input:
*A set of **points P** in space **(2d).***
*A neighbourhood **N** and a neighbourhood value **eps (see figure 1).***
*And a parameter **minpts,** which determine when a cluster*
can be taken as dense.
***i**. The algorithm starts with an arbitrary unvisited starting point.*
***ii.** Then extracts the neighbourhood of that point using the*
***eps** value (and making sure).*
*// all points within the **eps** distance are in the same neighbourhood.*
***iii**. Clustering process begins when enough points*
*(**minpts**) are found around the neighbourhood with a distance not more than **eps** between each points.*
//For all points that belongs to the cluster (including its
***eps neighbourhood**), repeat **steps** 3 through 5*
***iv.** Then new unvisited points are extracted and processed (this might lead to the discovery of further clusters or even noise).*
***v.** The process terminates only when all points are visited.*

Directly Density Reachable. A point **p** is directly density reachable from another point **q**, if **p** is within the ***eps*** (Figure 2a) neighbourhood of **q** and **q** is a core point (core because it has at least ***minpts*** within its neighbourhood – see Figure 2b).

4. Clustering

Clustering according to [4] is described as a data mining technique that groups data into meaningful subclasses, known as clusters. The procedure is applied to optimize the identification of the differences between subclasses in a large database class. Different types of clustering methods according to [3], exist including: hierarchical,partition, density based method and grid based method. K-means, K-medoids, BIRCH, DBSCAN, STING, Wave- Cluster, etc. are several clustering algorithms. DBSCAN is an effective density based method clustering algorithm for spatial database systems which can detect noise and outlier, cluster arbitrary shaped point dataset and do not require the number of clusters a priori. Notwithstanding the algorithm deteriorate in performance when data size becomes too large and may not perform optimally if the wrong values are chosen for minpoints and eps (which are two vital components of the algorithm).

A. Index structure for accelerated clustering algorithms:

Basically, as a general approach any n-dimensional data structure can be used for indexing the data in a spatial database, such as binary search trees and B-trees, R-trees, X-trees e.t.c. [7]. In most cases the R-tree based structures are used ([9]; [10]; [11]; [12]; [7]; [6]; [13]) and they are constructed using coordinates of the objects minimum bounding rectangles – MBRs – (covering or containing the spatial objects under consideration) as input.

Clustering algorithm for a spatial database can easily be enhanced for fast nearest neighbour search if they are indexed, because the indexes serve as good substitutions for poor performance caused by dimensionality [5]. Spatial index structures like the R-trees [11] are normally used in a spatial database management system to speed up the processing of queries such as region queries or nearest neighbour queries. When a SDBMs is indexed by an R– tree, then the R–tree nodes helps to accelerate the search operations [8] For instance, the branch and bound paradigm for nearest neighbour search proposed by [6], is a method that uses the two technique (mindist and minmaxdist) to order the nearest neighbour search, the mindist measures the minimum distance between a query point q and another point p, while the minmaxdist measures the minimum of the maximum of all distances from q to any vertex or face of the rectangle containing p, with these measures the lower and upper bound of the real distance between q and p is gotten and used for the calculate the nearest neighbour of all the objects in the index structure space. A modified version of the [6] method was proposed by [13], the new method avoided the use of the minmaxdist and applied only the mindist measure redundant search elimination for highly correlated data. Their method shows better performance in terms lesser number of disk accesses for individual query operation but it is limited by a poor computational strength. The X-tree proposed by [15] has the following properties which guarantees a better performance compared to the rest: 1) the super-nodes and overlap minimal split provide higher speed up for point and nearest neighbour queries. 2) With the increase in X-Tree search time which grows in a logarithmic manner with the database size, the tree structure scales well for very large database sizes. 3) Though the CPU-time of the X-tree is higher than that of the TV-tree, R*-tree and some others (because the nearest neighbour queries require sorting on the min-max distance [15], it is still better than that of an R-Tree.

For databases indexed using any of these hierarchical data structures (the R-tree and its variants including the X-tree), nearest neighbour search algorithms can decide to prune a branch of the tree if it identified that the region (feature space) that they represent do no promise any object belonging to the nearest neighbour of the query object processed [14].

B. Problems of existing DBSCAN:

Clustering algorithms according to [24], must satisfy the three basic requirements of a) have a basic domain specific knowledge to be able to determine the input parameters, b) discover clusters with arbitrary shape, and c) have a good efficiency on large databases. Despite the efficiency of the existing DBSCAN algorithm, it is well known to possess some major limitations, which include high time consumption for finding neighbourhood (eps) of a given data point [16], performance degeneration with increase in dataset size [20]. The DBSCAN algorithm clusters data points based on density and the underlying idea of density is based on the two parameters (Eps and Minpts). Though the existing method and its so many (R- tree indexed) variants performs well in discovering clusters with arbitrary shape, they do not scale well in terms of acquiring enough domain specific knowledge to be able to determine the input parameters Eps and Minpts, which is attested by the fact that users decision of these parameters (which is the mode of entry for the parameters) has huge undesirable effect on the algorithms behaviour. More so, in the aspect of measuring the algorithm’s efficiency on large databases, index structures like the R-trees [11] are normally used in a spatial database management system to speed up the processing of queries such as region queries or nearest neighbour queries. When the SDBSs is indexed by an R–tree (or any other indexing structure), then the tree nodes helps to accelerate the search operations [8]. Never-the-less, the basic limitations of the existing DBSCAN algorithm is compounded by the fact that the R-tree and its variant [15] are not adequate for large high-dimensional data sets as the index structures supports high overlap of the bounding boxes in the directory, which increases with growing dimension. The problem with this is that most large spatial databases are often represented using high- dimension feature vectors, thus because feature spaces most often tend to contain multiple instances of similar objects, then the database built using such a feature space is bound to be clustered and then if the database is indexed with an R-tree there would be cases of redundant search of rectangles due to the high overlap between MBRs of the R-tree nodes. According to [8] several new index structures (including the A-tree, VA-tree and the X- tree) have been proposed that outperforms the R-tree for indexing high dimensional data but most of them show degraded performance as dimension increases [5], [15]; [8]. Thus based on these premises we propose an improved DBSCAN algorithm that is accelerated using an adjusted X-tree (aX-tree) and scalable for large datasets. The main objective of the proposed algorithm is to save users the stress of predicting the appropriate value for the main input parameters (Eps and Minpts) for the algorithm, by equipping the new algorithm with the intelligence of fast nearest neighbour (E) computation for eps and the assumption of m (the maximum capacity of an aX-tree node) as the Minpts.

5. Proposed Method

The proposed system is an improvement on the work we described in [26] and [27]. However, we have made new suggestions to enhance the performance of the proposed algorithm. The bulk-loading algorithm in subsection A of this section uses the pre-processing technique introduced in the [26]. However, the aX-DBSCAN algorithm in subsection B, uses a different procedure (from the one we proposed [26]) for calculating and finding the nearest neighbour of a given object in space for effective clustering purpose. This section explains in detail the proposed system, how it works and what makes it different from the existing system.

The clustering procedure starts when the aX-tree is loaded into memory (as outlined in Sub-section A), but this follows after the objects have been sorted and partitioned on the X-axis of the space (2D in this case) using range partitioning. The sorted rectangles are grouped into J = S/m and the space is sliced into square root (J), from here the procedure in subsection A begins. This process is fully described in [26] [27]. After indexing the database using the aX-tree, the distance between the leave nodes are computed, sorted and stored in other to determine the value of eps (E) which is an alternative to the eps value in the original DBSCAN algorithm proposed in [24].

Given:

*A point or a spatial database **S** {assuming that records consists of real number values}*
Let
***S** be the total points*
***k** =an arbitrary number*
***m** is the maximum capacity of an X-tree node*
A. Bulk Loading the X-Tree (aX-tree)

Start:

```
Step 1. Load the collection of the MBRs (in groups of m)
of sorted spatial objects from a temporary file.
Step 2. Create leaf nodes i.e. the base level (L = 0)
While S
/* i.e. while pre-processed sets of objects > 0*/
Step 3. Create a new aX-tree node,
Step 4. Allocate the subsequent m rectangles to this node
/* During node creation avoid split that cause overlapping, by extending one super-node in the current level (only for leave level)
Step 5. Create nodes at higher level (L + 1)
While (nodes at level L > 1)
Step 6. Sort nodes at level L ≥0 on ascending creation time
Repeat steps 2
Step 7. Return Root
```

B. Our clustering (aX-DBSCAN) algorithm

```
Start from root
Locate ax-tree leaf node with lowest x-value
→ L = 1
→ While L ≥ 1
1. Determine Φ > 0
/* Φ is the parameter that determines which
cluster is dense*/
/* value of Φ = (at least) m +1 so as to avoid clusters that has only one object */
2. Find a neighbourhood value (E > 0)
/* the value of E (nearest neighbour to an
object s∈ S) outlined in section D.
```

```
3. Clustering
i.   Find $B_i = \{s \in S: d(s_i, s) \leq E\}$
// (d = distance between s and si for
all i = 1, 2....m)
ii.  If $|B_i| \leq \Phi$, THEN
iii. REJECT Bi
// Bi is an outlier
ELSE
iv.  Find the relationship $(B_i \cup B_j \neq \emptyset)$
v.   Repeat iv until there is no more union
```

C. How the proposed system works

Initially, we create temporary file that stores all the calculated distance between the points using the distance function:

Distance *(**p**, **q**)* $\geq$ *distance (MBR (**p**), MBR (**q**)) (for any point or object* E_i *in the tree, i = 1….. **S**).*

Assumption 1:

For each **node v,** the distance between any two points in that node say S_i **and** $S_j = 0$
Proof:

The distance between all the elements of **kth partition in figure 2b** is equal considering the fact that they are all located in one internal MBR (*r*) bucket (as compared to other MBRs) based on their spatial occupancy. Therefore, since Φ ***= (at least) m +1*** and ***m*** is the maximum capacity of an aX-tree node, it means all the objects in V_j automatically falls into the same cluster. Thus: ***distance*** (S_i, S_j) ***in*** $V_j = 0; i \neq j;$ i, $j \geq 1$. On the other hand for super-node s in figure 2b therefore whether s forms a valid cluster depends on the value of Bi.

Other assumptions

a. The distance to the k-nearest neighbour (calculated by the distance function) represents eps neighbour, therefore the eps-neighbourhood (E) are those points whose distance falls within k-th distance. (I.e. eps- neighbourhood = points around k-nearest neighbours

b. Using the aX-tree, the edges of the MBR containing the points in the space represents their nearby neighbours. Therefore we calculate the distance between the query MBR and the MBRs that share edges with within k-distance

c. The choice of $\Phi = m$ in the aX-DBSCAN algorithm is based on the fact that all elements are packed into nodes in all levels of the tree equally, therefore setting the value of the minimum objects in the nodes to be equal. As such core points or core objects are those elements having up to minimum points in their immediate neighbourhood. Therefore we have assumed $\Phi = m$ to match the requirement of minpoints as in the original DBSCAN algorithm

Furthermore, Super-nodes are created only at the leaf using our packing algorithm as described in subsection A. Super nodes are formed when the total number of entries in a given directory node exceeds m but cannot be packed into a new node without destroying proximity.

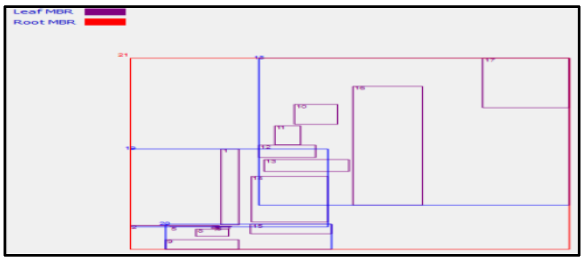

Figure 2a: aX-tree structure - packed rectangles.

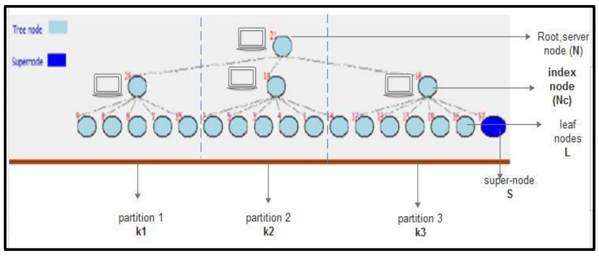

Figure 2b: aX-tree structure - the tree nodes

Implementing the DBSCAN algorithm based on indexing the database on an aX-tree has a lot of benefits. Closely related to the work in [6] which was modified by [13], our work has taken a much easier approach for KNN calculation by limiting the number of distance calculation to only involve the nodes and not the objects itself at the first stage. Starting from the root (S), the leftmost leave node (L) on the tree is located and the clustering process begins, MBRs (r) which do not fall within k-th distance of the leftmost r is pruned and then clustering is performed within the objects in the remaining r which passed the distance test. At the end of the first stage clustering process, the distance is computed between the last r in the current neighbourhood and the children of the next consecutive L +1 level of the tree and then continues until the last node is processed. Using this method, clustering progresses easily by cutting off all nodes not within the limit of the K-Nearest Neighbour, thereby making the procedure a bit faster for very large database. The idea behind this is that since similar objects are sorted and packed together in the same bucket (partitions in figure 2b) in a recursive manner up to the root then, objects which do not fall into a particular partition/group do not belong to that eps neighbourhood. The formula below calculates the minimum distance between a query MBR and other MBRs:

Given two leave node of the tree structure: MBRs rI, rJ, ***Dist*** (rI, rJ) = sum (n)| rI - rJ $|^2$ (where **i ≠ j and n** is the different dimension).

*D. How we find eps (**the** nearest neighbours **E**)*

For finding the closest neighbours considering the coordinates of the lower-left and upper-right corner of the rectangle given as (XAi; XAj; YAi, YAj) in figure 3, we are presented with the following scenario.

Case 1 The objects intersects and **Case 2** The two objects are classified as adjacent because they share either a common edge or a vertex. Therefore for any two rectangle (*MBRs*) say A or B in *n* dimension, we have:
rA = {XAi, YAi} *{ i > 0 ≤ n}*
rB = {XBi, YBi}

CASE 1:
*XA1 < XB2 and XA2 > XB1 and YA1 >YB2 and YA2 < YB1 // i.e**. rA intercepts rB = TRUE THEN***
Return
Dist (rA, rB) = 0

Else
CASE 2:

XB1 > XA2 or XB2 < XA1 or YB1 > YA2 or YB2 < YA1//
*i.e**. rA intercepts rB = FALSE***
Dist (rA, rB) > 0
→ XB1 > XA2, XB2 < XA1 = Dist (rD, rE)
Or YB1 < YA2 or YB2 > YA1 = Dist (rC, rD)

Finding the closest rectangle becomes easy because the databased is indexed with packed X-tree, where rectangles are already packed into buckets (***MBRs***). Based on this advantage and because these MBRs are axis aligned and packed in a sorted order, all we need to do therefore is: having known the *min and max* values of the XY coordinates *(i.e. XA1, YA1, XA2, YA2)* for ***rectangle A* in figure 3**,

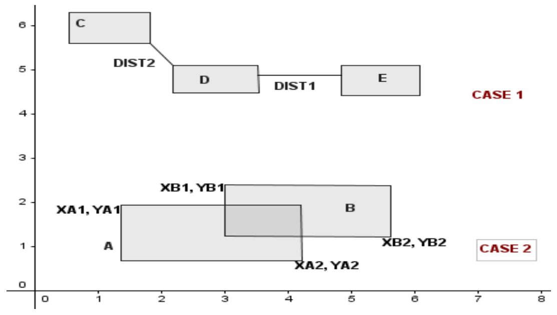

Figure 3: relationship rectangles (nodes)

Thus when the left and right edges of the rectangles are *different i.e. XB1 > XA2, XB2 < XA1 we compute Dist (rD, rE).*
*Else when **case** → YB1 < YA2 or YB2 > YA1 then compute = Dist (rC, rD).*

Based on the number of boundary points around each MBR (which depends on the dimension), we carried out a series of test. For each vertex in the adjacent (or neighbouring) polygon, we compute the distance to each vertex in the query polygons and eventually, we find the closest (using their Euclidean distance ***Dist***).

Given any two (2) MBRs in n-dimensional Euclidean space rA and rB, let rA = (Xa, Ya), and rB = (Xb, Yb), be their upper right and lower left corners respectively, then ***Dist*** *(rA, rB) = min {min {dist (r_a, r_b)}}, (1)*
//∀ $r_a \in rA$; $r_b \in rB$

(I.e. for any point/object ra in rA, find the least distance to any point/object in rB; then find and store all the smallest distance in r_a)

What is the value of **K**

After these distances are calculated and sorted in an ascending order then.

1. We create a buffer for the most k closest neighbours.
2. Based on this, we prune the MBRs based on the distance of the furthest from k

Following the above description, the next stage follows the steps below, where the distance calculation is based on the distances between the nodes i.e. Euclidean distance between nodes i through N

Following the above description, the next stage follows the steps below, where the distance calculation is based on the distances between the nodes i.e. Euclidean distance between nodes i through N.

6. Performing our Clustering Algorithm

Starting from the root node
→Take the leftmost internal node, search all its child until you get to the leaf ***(i)***
While on the level → leaf
→ Start from the leftmost node
→Find ***E (k*** *-neighbourhood as described above)*
Begin clustering
→Find all directly reachable nodes (from the
aX-tree*) as section V (D)*
→ Follow the rest of the Clustering (aX- DBSCAN AGORITHM) until the last leave node
(r) in the current neighbourhood
→ Move to the children of the next consecutive ***i***
+1 level of the tree and then continues until the last node in the tree is reached.

7. Conclusion

DBASCAN algorithm is one algorithm that is well known and used in big data analysis for clustering a large set of spatial data in other to find important information from the data. Studies have shown that the algorithm does not perform optimally when the size of the database begins to grow and that it has a very bad worse case complexity in terms of finding its two most important parameters the ***eps and the minpoints***. **In t**his paper, we examined some of the existing methods that has tried to improve the performance of the algorithm in terms of analysing huge databases and in its time consumption in the process of trying to choose the appropriate ***minpoints and eps*** values. We discovered that apart from other spatial access method which has been proposed to improve the existing DBSCAN clustering algorithm, the R-tree spatial indexing method and its variants has been the most widely used. And so we proposed the use of a different indexing structure which has a higher capability than the R-tree in terms of handling data in high-dimensional data space. The sort based X-tree *(aX-tree)* that we propose, like the original X-tree is focused on creating a spatial indexing method with overlap minimal split but in addition, the new X-tree is pre-sorted and packed thus increasing its capabilities.

References

[1]. G. Samson, J. Lu, and Q, Xu, Large spatial datasets: present challenges, future opportunities, Int'l Conference on Change, Innovation, Informatics and Disruptive Technologies, North

[2]. Al-Naymat, GCG: Mining maximal complete graph patterns from large spatial data. In Computer Systems and Applications (AICCSA), 2013 ACS International Conference on (pp. 1-8). IEEE.

[3]. J. Han, and M. Kamber, Data Mining Concepts and Techniques: Morgan Kaufmann Publishers, San Francisco, CA 2001. pp. 335-391.

[4]. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "*From Data Mining to Knowledge Discovery in Databases"*, AI Magazine, 1996, Vol. 17(3), p. 37.

[5]. P. Berkhin, A survey of clustering data mining techniques. In *grouping multidimensional data* (pp. 25-71). Springer Berlin Heidelberg, 2006).

[6]. N. Roussopoulos, S. Kelley, and F. Vincent, Nearest neighbor queries. In Proceedings of the 1995 ACM-SIGMOD Intl.Conf. on Management of Data, San Jose, CA, June 1995.

[7]. F. Cazals, I. Z. Emiris, Chazal, F., Gärtner, B., Lammersen, C., Giesen, J., and G. Rote, "D2. 1: Handling High-Dimensional Data". *Computational Geometric Learning (CGL) Technical Report No.: CGL-TR-01*, 2013.

[8]. N. Mamoulis, *Spatial Data Management*, 1st ed., Morgan & Claypool Publishers, US 2012

[9]. Z. Song, and N. Roussopoulos, K-nearest neighbor search for moving query point. In *International Symposium on Spatial and Temporal Databases* (pp. 79-96). Springer Berlin Heidelberg, July 2001.

[10]. P. Rigaux, M. Scholl, and A. Voisard, Spatial Databases with Application to GIS. *SIGMOD Record*, 2003, *32*(4), pp 111.

[11]. M. Ester, H. P. Kriegel, and J. Sander, Knowledge discovery in spatial databases. In *Mustererkennung 1999* (pp. 1-14). Springer Berlin Heidelberg, 1999.

[12]. K. S. Candan, and M. L. Sapino, *Data management for multimedia retrieval*. Cambridge University Press, 2010.

[13]. J. Kuan, & P. Lewis, Fast k nearest neighbour search for R-tree family. In *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on* (Vol. 2, pp. 924-928). IEEE, 1997, September.

[14]. Y. Ioannidis, H. Marc, J. Scholl, W. Schmidt, M. Florian, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Advances in Database Technology -- EDBT **2006:** 10 International Conference on Extending Database Technology, Proceedings: Springe: Munich, Germany, 26-31 March 2006.

[15]. S. Berchtold, D. A. Keim, and H. P. Kriegel, An index structure for high-dimensional data. *Readings in multimedia computing and networking*, 2001, pp. 451

[16]. M. Szczuka, M. Kryszkiewicz, R. Jensen, and Hu Q. *eds.* Rough Sets and Current Trends in Computing: Springer-verlag Berlin Heidelberg. Proceedings of the 7th International RSCTC Conference (2010), LNAI 6086, pp. 60- 69.

[17]. X. Xu, J. Jager, and H.-P. Kriegel A Fast Parallel Clustering Algorithm for Large Spatial Databases. Data Mining and Knowledge Discovery, 1999, 3(3):263-290.

[18]. A. Amirbekyan, and V. Estivill-Castro, "Privacy preserving DBSCAN for vertically partitioned data". In *Intelligence and Security Informatics* Springer Berlin Heidelberg 2006, pp. 141-153.

[19]. M. Chen, X. Gao, and H. Li, (2010) "Parallel DBSCAN with priority r-tree," in Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. IEEE, 2010, pp. 508-511.

[20]. S. Vijayalaksmi, and M. Punithavalli, "A Fast Approach to Clustering Datasets using DBSCAN and Pruning Algorithms. *International Journal of Computer Applications*, 2012, *60*(14).

[21]. B. Welton, E. Samanas, and B. P. Miller, Mr. Scan: Extreme scale density-based clustering using a tree-based network of gpgpu nodes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis,* 2013, November. pp. 84, ACM.

[22]. M. Hahsler, P. Piekenbrock, S. Arya and D. Mount, Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms, 2016 <online available at> [https://cran.rproject.org/web/packages/dbscan/dbscan.pdf]

[23]. L. Chakrawarty, and P. Gupta, "Applying SR-Tree technique in DBSCAN clustering algorithm" International Journal of Application or Innovation in Engineering & Management (IJAIEM). ISSN 2319 -4847. 3 (1) pp 207 -210, (2014)

[24]. M. Ester, H. P. Kriegel, J. Sander and X. Xu, (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, No. 34, pp. 226-231).

[25]. J. Sander, M. Ester, H – P. Kriegel, and X. Xu, "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications". *Data Mining and Knowledge Discovery* (Berlin: Springer-Verlag), 1998), 2 (2): pp169–194. Doi: 10.1023/A: 1009745219419

[26]. G. L. Samson, J. Lu pax-dbscan: a proposed algorithm for improved clustering. In M. Pankowska, P. Keller, A. Bronder & M. Safian (Eds.), Studia Ekonomiczne - Akademia Ekonomiczna im. Karola Adamieckiego, 2016. 296 (pp. 86 - 121). Katowice, Poland: University of Economics Katowice.

ISSN 2083=8611.http://www.sbc.org.pl/dlibra/publication?id=14232&tab=3

[27]. G, Samson, J. Lu and H. Jazaar, "aX-tree: an Improved Structure for Spatial Data Indexing", 2016, Unpulished