XDBSCAN: A Fast Parallel Density-based Clustering Algorithm Using Graphics Processing Units

Wilson A. Urdaneta V.

October 2, 2012

School of Computer Science and Software Engineering,

The University of Western Australia,

Crawley WA 6009, Australia

Abstract

Cluster analysis (clustering) is the process of grouping sets of objects with similar characteristics into classes, a process commonly used in different disciplines for data mining. This work investigates the use of CUDA technology to parallelize the DBSCAN algorithm, a density-based clustering technique. The capabilities and benefits of this relatively new parallel architecture for General-Purpose computation on GPU (GPGPU) are explored, tested, and used progressively to develop a massively parallel density-based clustering algorithm, using as main indicators computing time and scalability. This work shows that it is possible to speed-up the DBSCAN algorithm by up to two orders of magnitude (20X-140X) using a GPGPU. We propose a new parallel density-based algorithm, XDBSCAN for eXtreme DBSCAN, that performs faster, especially for large data sets and high dimensional data.

Keywords: Algorithms, Performance, Density-based clustering, DBSCAN, Graphics Processing Unit (GPU), CUDA

CR classification: H.2.8 [Information Systems]: Database Applications - Data mining; I.3.1 [Computer Graphics]: Hardware Architecture - Graphics processors

1 Introduction

A new era has emerged in information technology and science, one driven by data, where data-intensive problems outpace computing capacity, affecting and challenging both scientific research [2, 3] and private sector [3]. Data mining, an emerging field of Computer Science, provides the tools and techniques to assist in the analysis of large amounts of data for knowledge discovery, and it

is being research extensively as a way to deal with the information overload issue [11, 14, 13, 7, 9]. Data mining can be defined as the process of analysing large data sets to find useful patterns that can be used to obtain information and knowledge. The Density-based spatial clustering of applications with noise (DBSCAN) algorithm is a data mining technique commonly used for spatial data clustering. It is capable of finding arbitrarily-shaped clusters and can handle noise. Additionally, DBSCAN easy to use and requires only two input parameters, and contrary to other algorithms such as K-means the number of clusters does not have to be known in advance [10].

It is accepted that 20% to 30% of the time and effort of a data mining project is used for data understanding, while 50% to 70% is expended on data exploration and analysis [8]; consequently, performance-related issues are among the main challenges in data mining. In the last years, a significant amount of investigations have focused on the use of Massively Parallel Architectures to optimise existing algorithms. This is the case of the Compute Unified Device Architecture (CUDA), a relatively new technology, created by NVIDIA, for developing massively parallel applications that increase computing performance by using the power of Graphic Processor Units (GPUs). The research focus of this work is the use of Massively Parallel Architectures to optimise data mining techniques. More specifically, this work investigates the use of CUDA technology to parallelise the DBSCAN algorithm and shows that it is possible to speed-up the DBSCAN algorithm by up to two orders of magnitude (20X-140X) using a GPGPU. We propose a new parallel density-based algorithm, XDBSCAN for eXtreme DBSCAN, that performs faster, especially for large data sets and high dimensional data.

The rest of the paper is organised as follows. Section 2 surveys the existing work in density-based clustering, specially when using GPUs. Section 3 briefly introduces the CUDA architecture. Section 4 presents formally the DBSCAN algorithm. Section 5 explains XDBSCAN, our novel CUDA-based density clustering algorithm. Section 6 presents the experimental results. Finally, section 7 concludes with a summary and provides some recommendations for future research.

2 Previous Work

Since the introduction of DBSCAN, several studies have been done to improve, extend, and adapt the algorithm in several ways, and it is critical to ascertain this work for its further optimisation. This section presents the most popular modifications made on this algorithm. In the literature, two major modifications to DBSCAN are identified: changes made to improve efficacy, and changes made to optimise the efficiency of the algorithm. One major modification to DBSCAN w.r.t. efficacy is the introduced by [15] with GDBSCAN, an algorithm that generalises the notion of point density and therefore can cluster points objects and spatially extended objects, considering both spatial and non-spatial attributes. Another work is the introduced by [1] with the Ordering Points To Identify the

Clustering Structure (OPTICS) algorithm. While DBSCAN identify clusters using a single density input, OPTICS is a hierarchical representation of the intrinsic structure of the data that makes it able to detect meaningful clusters in data of varying density.

In terms of efficiency, the literature reveals the use of three major optimisation techniques: sampling techniques, to reduce the number of points being processed during the expansion process; parallel techniques, which speedup run time by using concurrent processes to minimise workload ; and partitioning techniques, which divide workload among different hosts.

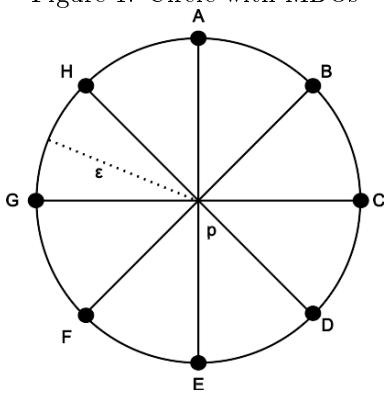
2.1 Sampling techniques

These are the most common optimisation techniques found in the literature. They basically optimise DBSCAN by reducing the number of points that will be selected for expansion during cluster formation. Literature ascertain more than 10 different sampling techniques, which seem to be popular because of their simplicity and efficacy. Sampling techniques use the concept of Marked Boundary Objects (MBOs) [5], this is a set of objects that delimits a core point when its n-dimensional space is divided equally in n quadrants. Generally, for a point of dimension d, there are up to $(3^d - 1)$ MBOs and 2^d quadrants. For instance, consider a core point p with radius Eps and a minimum of points $MinPts$ in a two dimensional space. A circle can be drawn with radius Eps and center at point p. By drawing the coordinate axis and two diameters line with angles of 45^o and 135^o to the x-axis, the circle is divided into eight sections, and marked by eight points known as MBOs (see figure 1). These points are used to drive the expansion process by selecting for expansion only those core points closest to MBOs, without selecting the same object twice; therefore, for the two dimensional case the total number of seeds selected is less or equal than eight, and consequently reducing the number of core points selected for expansion. The major drawback of sampling techniques is the accuracy of the results that are only approximate solutions to the real results.

2.2 Parallel and partitioning techniques

Parallel techniques try to use concurrent processes to accelerate execution time; while, partitioning techniques divide the work load among different hosts; therefore, depending upon the implementation, these techniques may overlap. Where all partitioning techniques can run in parallel, only some parallel techniques can partition the data. Among the most popular studies in this area are: the PDBSCAN algorithm [17], it divides the input into several partitions, and distribute them, so DBSCAN can be run concurrently among different computers. The final results are merged; [1] presents a parallel implementation of DBSCAN. They use a Parallel Programming Environment (PPE) as main tool to parallelise DBSCAN. The final work results in a Master-Slave parallel implementation of DBSCAN; [18] presents a distributed implementation of DBSCAN, this uses

Figure 1: Circle with MBOs



sorting as main technique for dividing the workload. Finally, [4] presents a massively parallel implementation of DBSCAN using GPUs, this implementation is not scalable and the workload cannot be partitioned among different hosts or GPUs.

2.3 Our work

Our work improves by providing an efficient algorithm that does not require sampling techniques for optimisation and therefore yielding exact results to those computed by DBSCAN. Also, our work provides a massively parallel, scalable and partition-able algorithm that take full advantage of GPUs to reduce computing time.

3 CUDA Architecture

During the last few years, GPUs have evolved into powerful and affordable devices; that is, they increasingly offer a programming environment that facilitates the implementation of applications that take advantage of their parallel architecture for algorithm optimisation. Additionally, they are becoming ubiquitous in computer systems, found from laptops to super computers. This is the case of CUDA, NVIDIA's massively parallel architecture, that provides a complete suite of tools for implementing massively parallel applications on GPUs.

A typical CUDA-capable GPU has a set of streaming multiprocessors (SMs) and each SM has a number of streaming processors (SPs) that are capable of execute the same instruction concurrently, which allows the massively parallelisation of tasks. GPUs usually have available large amounts of high-speed RAM that is used to support all computing operations. Finally, GPUs cannot communicate directly with the CPU and the communication is achieved through specialised API functions that transfer the information from/to the CPU using the PCI-Express Bus. As an example, the GeForce GTX 550 Ti has 24 SMs,

each with 8 SPs for a total of 192 SPs (cores), 1024MB of RAM with a memory bandwidth of 98.4GB/sec and a maximum computational power of 691.2 GFLOP/sec.

4 The DBSCAN Algorithm

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm, proposed by Ester et al. in 1996 [10], is a clustering technique that uses the notion of point density to find clusters. The idea is to take advantage of the fact that the number of points within clusters is substantially higher than outside of clusters [5] and, therefore, form clusters by finding contiguous high-density areas; whereas, low-density areas indicate either the start of a new cluster or the presence of noise points. DBSCAN requires only two input parameters, the radius search (Eps) and the minimum number of points of the radius search (MinPts). The DBSCAN algorithm has the following formal definitions:

- **Eps-neigbourhood** is the neighbourhood of an object **p** within a given radius **Eps**.
- **MinPts** is the minimum number of objects that an object **p** must have in its **Eps-neigbourhood** to be a **core point**.
- An object **p** is **density-reachable** from an object **q** wrt. **Eps** and **MinPts** in the set of objects **D**, if there is a chain of objects p_1,... ,p_n, p_1 = q, p_n= p such that p_i belongs to **D** and p_{i+1} is directly **density-reachable** from p_i wrt. **Eps** and **MinPts**.

DBSCAN uses the **Eps-neigbourhood** of each point to search for clusters. If the **Eps-neigbourhood** of a point **p** is greater or equal than **MinPts**, a new cluster is created with **p** as core point. Then, DBSCAN tries to expand the cluster by collecting iteratively directly **density-reachable** objects from **p**, which may require the merging with other clusters. The algorithm finishes when all points have been added to a cluster or labeled as noise [13]. The most computational demanding methods of DBSCAN are: the expansion method that does the *core point expansion* and the regionQuery method that does the *neighbour search*.

There are several positive factors that have made DBSCAN a popular choice for spatial-data cluster analysis. First, it is widely known for its effectiveness to detect unknown number of clusters of arbitrary shape such as spherical, linear, or elongated. Second, it is robust to outliers, being able to detect with precision noisy points. Third, compared with other algorithms such as CLARANS, DBSCAN is efficient and can handle relatively large data sets. It has a complexity of $O(n^2)$ that can be easily reduced to O(nlog(n)) using spatial indexing structures such as R-tees or kd-tree to assist in the neighbour search. Finally, the concepts behind DBSCAN are very intuitive and easy to extend; DBSCAN requires only two input parameters that can be easily estimated (Eps and MinPts).

5 XDBSCAN

As with most programming-related activities, developing a new optimised version of an existing algorithm can be difficult. For this research work, the major challenge was to use properly CUDA's parallel architecture to develop a massively parallel implementation of DBSCAN. That is, the challenge was not only to adapt DBSCAN to CUDA, but also to find the adaptation that best utilises all parallel resources. Thus, assuring that the new CUDA-based DBSCAN is faster than the sequential version, and it is the faster parallel implementation that can be developed in the current CUDA architecture.

Two versions of XDBSCAN were implemented (XDBSCAN1 and XDBSCAN2). These versions modify and adapt the vanilla version of DBSCAN to exploit more effectively the resources of NVIDIA's parallel architecture and therefore optimise further the computing time. They use the GPU to accelerate the *neighbour search* and *core point expansion* methods, on which DBSCAN heavily relies for the cluster finding process.

5.1 Parallel Neighbour Search

During the development of XDBSCAN two different techniques were explored to improve the neighbour search process. The first technique (used by XDBSCAN1) uses brute force to compute all distances and determine the neighbours of each point. In this technique, the number of threads launched is equal to the number of distances to be computed. For instance, the number of distances to compute with a data set of size N is N^2; therefore, $N^2$is the number of threads launched. Once computed all distances, core nodes are determined by counting the number of distances less than **Eps** for each point. If the number of distances are greater or equal to **MinPts** then the current point is a core point. Both, distances and core points are passed as input parameters to the parallel expansion process.

The second technique (used by XDBSCAN2) reduces further the computational time. Instead of using one thread to compute each neighbour distance (N^2 threads), a reduced number of threads that is equal to the size of the data set (N) are launched to compute the number of neighbour points of each point in the data set, resulting an array that contains the number of neighbours of each point (see figure 2). Then, the elements in the array with a number of points less than **MinPts** are deleted so the array can be used as a core-point array, useful for driving the parallel expansion process.

5.2 Parallel Core Point Expansion

The seed point expansion was accelerated by assigning one CUDA thread to each element of a core point. Then, each thread checks whether its assigned element is a core point itself, if so then the thread will proceed to the expansion of this element. That is, multiple threads expand a core cluster from multiple ends (see figure 3).

Figure 2: Parallel count of neighbour nodes

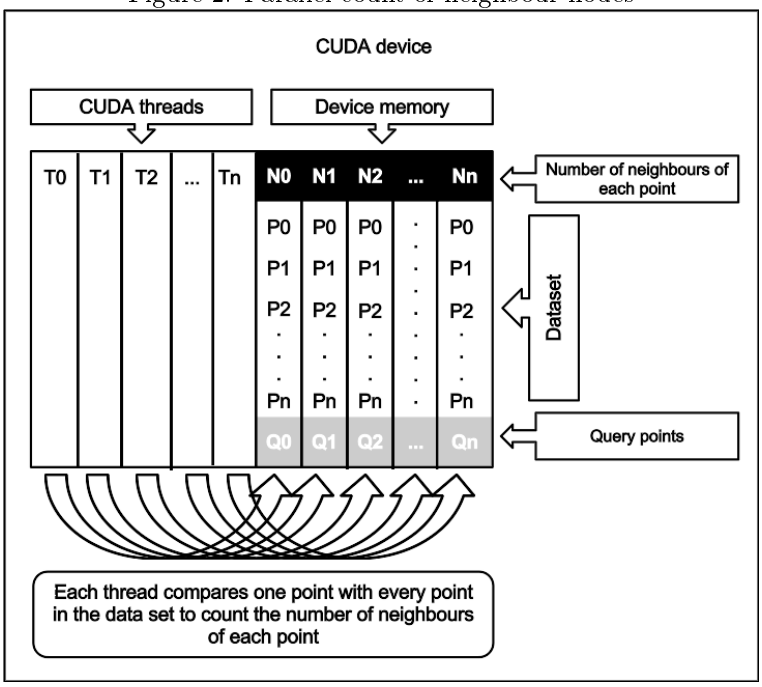


Figure 3: Parallel core point expansion

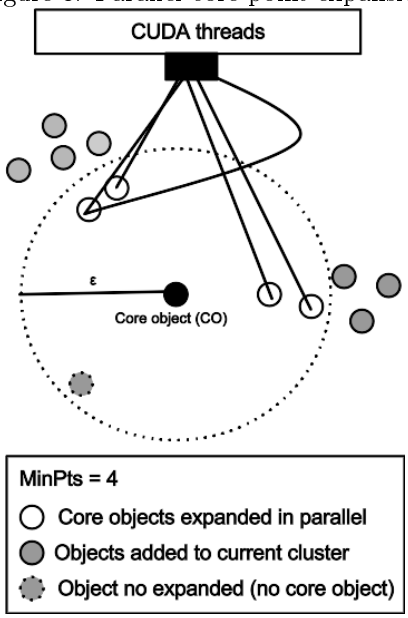


Table 1: Experiment 1 (runtime)

Algorithm/Data size	10k	20k	50k	100k	250k	500k	1m
Dimension: 2, Eps: 0.65, MinPts: 4							
XDBSCAN1	0.28	0.60	2.38	6.15	26.15	104.15	321.23
XDBSCAN2	0.22	0.54	2.02	4.89	18.59	79.48	224.69
vDBSCAN	3.46	13.90	82.07	329.89	2076.54	8117.57	32549.01
iDBSCAN	0.22	2.46	37.05	275.84	1205.28	11201.97	25228.32

6 Experimental Evaluation

All experiments were performed on a PC equipped with these components: Intel(R) Core(TM) i3-2130 @ 3.40Ghz, 8GB DDR3 RAM, and a Gainward NVIDIA GeForce GTX550Ti (192 SP) 1GB GDDR5 RAM. All algorithms were implemented in C/C++ using the CUDA compilation tools version 4.

Synthetic data was used for all the experiments, the data sets can be classified into two categories. A set of data that are relatively small having 3K-10K points, 2 dimensions, 2-20 clusters, and complex shapes of clusters, used to validate the algorithms in terms efficacy. The other data sets are larger, having 10,000-1,000,000 points, 2-500 dimensions, 2-20 clusters, and simple shapes (Gaussian-based clusters), used to test the computing time of the algorithms. All data was generated with a cluster generator developed by [?] at the University of Manchester. The results are bench-marked with two sequential DBSCAN implementations. The first implementation is based on the standard DBSCAN algorithm (vDBSCAN) and the second implementation uses a kd-tree as an indexing structure to improve the performance of DBSCAN (iDBSCAN).

Two major experiments were conducted. First, the run-time of all the algorithms is measured (table 1). then, the impact of highly dimensional data on runtime is assessed (table 2). In overall, XDBSCAN's performance was very satisfactory, optimising up to two orders of magnitude. Figure 4 shows the speedup achieved during runtime of our final implementation of XDBSCAN vs DBSCAN.

6.1 Runtime

All CUDA-based codes outperformed by several factors both vDBSCAN and iDBSCAN; therefore, this section discusses the results of each XDBSCAN implementation with iDBSCAN. XDBSCAN1 outperforms iDBSCAN by a factor of 4X to 107X, which is a huge improvement, and XDBSCAN2 outperformed even further by a factor of 4X to 140X. That is, XDBSCAN2 shows better runtimes, benefiting of an improved neighbour search method that uses more effectively CUDA's parallel architecture. More specifically, XDBSCAN2's neighbour search deals more effectively with problems such as memory access collision and thread synchronisation, and takes advantage of coalesced memory access.

Figure 4: XDBSCAN vs vDBSCAN (runtime)

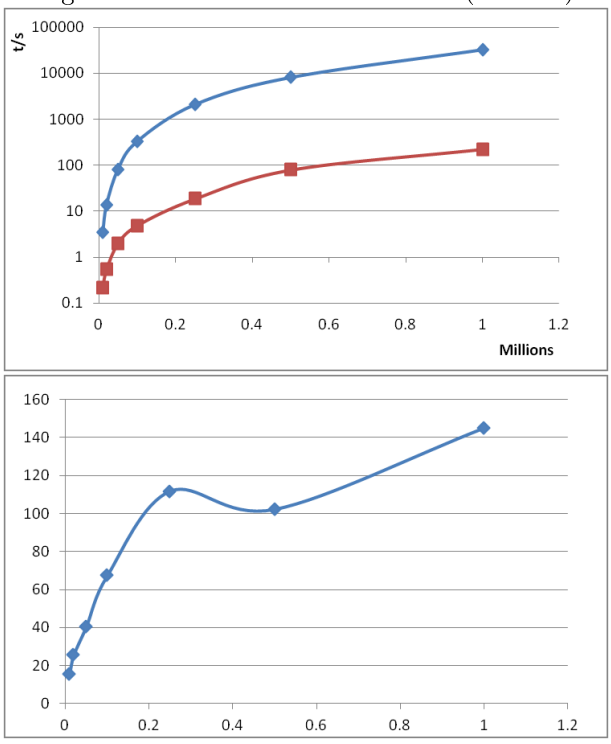


Table 2: Experiment 2, (dimensionality)

Algorithm/Dimension	4	6	10	20	50	100	200
XDBSCAN1	1.85	2.30	4.58	3.46	10.35	65.59	146.31
XDBSCAN2	1.50	2.05	4.44	3.47	21.70	88.98	194.35
vDBSCAN	106.86	116.01	146.64	235.86	509.89	908.08	1702.86
iDBSCAN	166.78	76.03	6.4	489.33	532.58	435.16	238.14

6.2 Dimensionality

Given the results from table 2, it is clear that the data dimension influcences the runtime of the algorithms. In fact, for highly dimensional points XDBSCAN1 performs better than XDBSCAN2 as the latter is affected by non-colasced memory access when computing distances. In overall, XDBSCAN was able to outperform the sequential implementations by a factor of 1X to 90X.

7 Conclusion

We present XDBSCAN, a massively parallel, scallable and partion-able implementation of DBSCAN. XDBSCAN is an efficient algorithm that contributes directly to the need of efficient data mining techniques [6, 14, 12, 18, 16] and proves that it is possible to speed-up the DBSCAN algorithm by up to two orders of magnitude (20X-140X) using a GPGPU. Our work improves by providing an efficient algorithm that does not require sampling techniques for optimisation and yields exact results to those computed by DBSCAN.

As future research work, it would be interesting to assess the scalability of the algorithm on large distributed environments with huge data sets (in the order of TB) by partioning the data among several hosts and GPUs. Also, to add the option of using sampling techniques to seepdup even further the execution time when a margin of error is acceptable in the results.

References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jorg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999.

[2] Gordon Bell, Tony Hey, and Alex Szalay. Beyond the data deluge. *Science Magazine*, 323:1297–1298, 2009.

[3] Francine Berman. Got data?: a guide to data preservation in the information age. *Commun. ACM*, 51(12):50–56, December 2008.

[4] Christian Böhm, Robert Noll, Claudia Plant, and Bianca Wackersreuther. Density-based clustering using graphics processors. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 661–670, New York, NY, USA, 2009. ACM.

[5] B. Borah and D.K. Bhattacharyya. An improved sampling-based dbscan for large spatial databases. In *Intelligent Sensing and Information Processing, 2004. Proceedings of International Conference on*, pages 92 – 96, 2004.

[6] Feng Cao, Anthony K. H. Tung, and Aoying Zhou. Scalable clustering using graphics processors. In *Proceedings of the 7th international conference on Advances in Web-Age Information Management*, WAIM '06, pages 372–384, Berlin, Heidelberg, 2006. Springer-Verlag.

[7] Ming-Syan Chen, Jiawei Han, and Philip S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8:866–883, 1996.

[8] Peter Christen, Markus Hegland, Ole M. Nielsen, Stephen Roberts, Peter Strazdins, and Tatiana Semenova. Efficient data mining: Scripting and scalable parallel algorithms. 2000.

[9] Gianmarco De Francisci Morales. *Big Data and theWeb: Algorithms for Data Intensive Scalable Computing.* PhD thesis, Computer Science and Engineering, IMT Institute for Advanced Studies, 2012.

[10] Martin Ester, Hans peter Kriegel, Jorg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[11] Piatetsky-Shapiro G. Fayyad, U. and P. Smyth. From data mining to knowledge discovery in databases. *American Assosication for Artificial Intelligence*, pages 37–54, 1996.

[12] Naga K. Govindaraju, Brandon Lloyd, Wei Wang, Ming Lin, and Dinesh Manocha. Fast computation of database operations using graphics processors. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[13] Jiawei Han. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[14] Harvey J. Miller. The data avalanche is here. shouldnâ??t we be digging? *Journal of Regional Science*, 50(1):181–201, 2010.

[15] Jorg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. 2(2):169–194+, 1998.

[16] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, 2008.

[17] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Min. Knowl. Discov.*, 3(3):263–290, September 1999.

[18] Aoying Zhou, Shuigeng Zhou, Jing Cao, Ye Fan, and Yunfa Hu. Approaches for scaling dbscan algorithm to large spatial databases. *Journal of Computer Science and Technology*, 15:509–526, 2000. 10.1007/BF02948834.