Neurocomputing ■ (■■■■) ■■■–■■■

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An efficient algorithm for distributed density-based outlier detection on big data

Mei Bai *, Xite Wang, Junchang Xin, Guoren Wang

College of Information Science and Engineering, Northeastern University, Liaoning, Shenyang 110819, China

ARTICLE INFO

Article history:
Received 13 February 2015
Received in revised form
27 April 2015
Accepted 22 May 2015

Keywords:
Density-based outlier
Local outlier factor
Distributed algorithm

ABSTRACT

The outlier detection is a popular issue in the area of data management and multimedia analysis, and it can be used in many applications such as detection of noisy images, credit card fraud detection, network intrusion detection. The density-based outlier is an important definition of outlier, whose target is to compute a Local Outlier Factor (LOF) for each tuple in a data set to represent the degree of this tuple to be an outlier. It shows several significant advantages comparing with other existing definitions. This paper focuses on the problem of distributed density-based outlier detection for large-scale data. First, we propose a Gird-Based Partition algorithm (GBP) as a data preparation method. GBP first splits the data set into several grids, and then allocates these grids to the datanodes in a distributed environment. Second, we propose a Distributed LOF Computing method (DLC) for detecting density-based outliers in parallel, which only needs a small amount of network communications. At last, the efficiency and effectiveness of the proposed approaches are verified through a series of simulation experiments.

1. Introduction

The outlier detection plays an important role in the area of data management and multimedia analysis, such as detection of noisy images and credit card fraud detection. According to Hawkins [1], "An outlier is an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism". Thus far, there have been a large amount of studies aiming at outlier detection, and various kinds of definitions have been proposed, *e.g.*, DB-outlier [2], top-n outlier [3] and density-based outlier [4].

There exist lots of excellent algorithms for the outlier detection. However, most of them only focus on centralized environments. With the increasing amount of data, the processing efficiency of these algorithms becomes limited and cannot meet users' requirements. For instance, in the area of electronic commerce, we consider the users' trade information as a data set, and the abnormal trade records as outliers. Then the techniques of the outlier detection can help us to find the theft of user accounts and avoid the property damage. For many online shopping websites (*e.g.*, eBay and Amazon), huge amount of trade information is generated every day. It takes hours even days if we use traditional centralized algorithms to compute outliers. The time-effectiveness cannot be guaranteed. In this case, economic losses cannot be avoided. Therefore, it is quite necessary to design a parallel algorithm that can use multiple machines to accelerate the outlier computing.

This paper focuses on the problem of the outlier detection in distributed environments, and we follow the definition of density-based outlier [4]. Specifically, in a data set, for each tuple p, we calculate its local outlier factor (LOF) that represents the degree of p to be an outlier. LOF can reflect the isolation situation of p with respect to (w.r.t.) its surrounding neighbors. We will give the formalized description in Section 3. Comparing with other definitions of outliers, the density-based outlier has two advantages. (a) In other existing approaches [2,3], the outlier detection is considered as a binary problem (either an object in the data set is an outlier or not). Instead, for the density-based outlier, they tend to assign each object a degree of being an outlier, and they show it is more meaningful in many complex scenarios. (b) They point out that in many real-world data sets, whether a tuple t can be an outlier or not depends on its surrounding neighborhood (only the tuples which are closed to p). In this situation, LOF achieves better expressiveness than other existing approaches.

There exists only one study [5] focusing on the same problem with our paper. In [5], they adopt a master–slave architecture for distributed computing. Each slave node calculates its neighborhood set (it is the matrix that contains all the local tuples and their respective neighborhood) and sends it to the master node. The master node collects all the partial neighborhood sets and calculates LOFs of all the tuples. Clearly, this approach is not suitable for distributed outlier detection on large-scale data, because a large

* Corresponding author.
E-mail address: baimei861221@163.com (M. Bai).

http://dx.doi.org/10.1016/j.neucom.2015.05.135
0925-2312/

number of tuples are aggregated to the master node, and lots of calculations are needed to obtain the result. The master node becomes the bottleneck when the data scale is large.

In this paper, to detect density-based outliers in distributed environments efficiently, we propose several practical techniques, which are summarized as follows:

1. We propose the Gird-Based Partition algorithm (GBP) for data preprocessing. The algorithm first splits the whole data set into several grids, and then allocates these grids to the datanodes in a distributed environment. Using GBP, we can balance the workload on each datanode and reduce the network overhead while computing outliers.
2. We propose the Distributed LOF Computing method (DLC) for detecting density-based outliers in parallel, which includes two portions. First, based on the characters of LOF, we classify the tuples in a grid into two categories: grid-local tuples and cross-grid tuples. The grid-local tuples can be processed locally, and the network communications are required only for the cross-grid tuples. Then, we design a tailored method to minimize the number of tuples that need to be transmitted across the network.
3. We evaluate the performance of the proposed approaches through a series of simulation experiments. The experimental results show that the density-based outliers can be computed in parallel efficiently using GBP and DLC. The performance of our methods can meet the requirements of practical applications.

The rest of this paper is organized as follows. In Section 2, we briefly review the related work. Section 3 states the problem of density-based outlier detection in a distributed environment. Section 4 describes the details of GBP and DLC. Section 5 presents the experimental results. Finally, we conclude this paper in Section 6.

2. Related work

We summarize the existing definitions of outliers and related computing methods in Section 2.1. Then the previous approaches of distributed outlier computing are described in Section 2.2.

2.1. Definitions of outliers

The concept of outlier was presented by Hawkins [1] in 1980. In the recent decades, it has attracted the attention of many scholars, and several formal definitions of outliers were proposed.

Several earlier studies [1,6,7] focus on statistic-based approaches to detect outliers, where the tuples are modeled as a distribution and outliers are the tuples who show significant deviations from the assumed distribution. However, for high dimension data the statistic-based techniques are unable to build an appropriate model, which leads to performance degradation. Some non-statistical (model-free) approaches are proposed and they do not rely on the assumed data distribution. Knox and Ng [2] proposed the distance-based (DB) outlier. Given two parameters k, r, the neighborhood of a tuple p are the tuples whose distances to p are smaller than or equal to r, and p is determined to be a DB outlier if the number of its neighborhood is smaller than a given threshold k. They also presented a nested-loop (NL) method to compute DB outliers. Ramaswamy et al. [3] pointed out that Edwin's method lacks the ranking information for outliers. Thus they proposed a new definition, called D_n^k outlier, and proposed some efficient algorithms to compute outliers using clustering techniques.

Another model-free approach is the density-based outlier [4] that is adopted in our paper. Instead of directly determining whether a tuple is an outlier or not, a *local outlier factor* (LOF) that represents the *degree* of this tuple to be an outlier is assigned to each tuple. The LOF of a tuple p is computed by analyzing its local neighborhood. Comparing with other definitions, the density-based outlier shows several significant advantages that have been illustrated in Section 1.

Some studies aim at improving the computational efficiency for the outlier detection. Bay and Schwabacher [8] proposed a nested loop algorithm using randomization and a simple pruning rule, and they showed that the algorithm has near linear time performance on many large real data sets. Angiulli and Fassetti [9] proposed DOLPHIN. Through maintaining a small portion of data in the main memory, the entire data are required to be scanned twice to calculate outliers. Furthermore, numbers of indexing techniques (*e.g.*, R-tree [10]) are employed to accelerate the computing speed. Recently, there also emerge some outlier detection algorithms for special purposes, such as uncertain data [11], streaming data [12], and high dimensional data [13].

2.2. Outlier detection in distributed environments

Faced with the large-scale data, it takes a long time for outlier detection if we still use the centralized algorithms, and the efficiency is not satisfactory in most cases. Therefore, some researchers start to utilize multiple machines to speed up the calculation, and several methods [14–16] for distributed outlier detection were proposed.

Otey et al. [17] proposed an outlier definition for the data with mixed attributes, and designed a distributed method to detect outliers. They considered the data sets which contain a mixture of categorical and continuous attributes. The computation process includes two steps. In the first step, they designed the anomaly score function and computed the locally anomaly scores for all the tuples. In the second step, a global schema is used to recalculate the anomaly scores of tuples whose locally scores are larger than the threshold. Finally, the outliers were chosen according to their anomaly scores.

Angiulli et al. [18] proposed a method for top-n outlier detection in distributed environments. In the first iteration, each salve node randomly selects n tuples and transfers them to the master node. In the master node, the neighborhoods of these tuples are calculated and the top-n tuples are chosen to filter out the local tuples in the salve nodes. In the second iteration, another n tuples in each salve node are randomly chosen and transferred to the master node. The master node recalculates the top-n tuples and uses them to prune local tuples. The process is repeated until all the local tuples are pruned. This method is complex and needed multiple iterations. Furthermore, most of the computations occur on the master node which would be a bottleneck for large-scale data.

Lozano and Acufia [5] proposed a distributed algorithm to compute density-based outliers, which targets at the same issue in our paper. The proposed algorithm is a parallel version of Breuning's method [4]. However, as we mentioned in Section 1, they adopt a master–slave architecture. In the slave node, the local neighborhood of each tuple is calculated locally. Then all the tuples and their neighborhood are transferred to the master node. The final result is calculated in the master node. Since all the tuples are transferred to the master node, the workload on the master node is quite heavy. Thus, this approach cannot achieve good performance when the data scale is large.

Outlier detection has some applications in the graph-based data management. Specifically, many graph-based models are used as geometric image descriptors [20] to enhance image categorization. Besides, these methods can be used as image high-order potential descriptors of superpixels [21–23]. Further, graph-based descriptors can be used as a general image aesthetic

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

descriptors to improve image aesthetics ranking, photo retargeting and cropping [24–26].

3. Problem statement

In this section, we first give the formal definition of the density-based outlier. Then we briefly describe the frame of density-based outlier detection in distributed environments. The symbols used in this paper are summarized in Table 1.

3.1. The density-based outlier

Given a data set P in d-dimensional space (the size of P is $|P|$), a tuple p is denoted as $p=\langle p[1], p[2], \ldots, p[d]\rangle$. To simplify the description, each dimension value of a tuple is normalized into $[0, 1]$ in the rest of this paper. The distance between two tuples p, q is

$$dis(p,q)=\sqrt{\sum_{i=1}^{d}(p[i]-q[i])^2} \quad (1)$$

Next, we give some basic definitions before describing the LOF of a tuple.

Definition 1. (k-distance of a tuple) Given a positive integer k, the k-distance of a tuple o (denoted as $dis_k(o)$) is the distance between o and a tuple p such that:

(a) there exist at least k tuples q that $dis(q,o)\le dis(p,o)$, and
(b) there exist at most $k-1$ tuples q' that $dis(q',o)<dis(p,o)$.

Definition 2 (*k-distance neighborhood of a tuple*). Given a positive integer k, the k-distance neighborhood of a tuple o is the set of tuples whose distances from o are smaller than or equal to the k-distance of o. $Neigh_k(o)=\{p\,|\,dis(p,o)\le dis_k(o)$ and $p\ne o\}$.

Note that, given a tuple o, the number of tuples in $Neigh_k(o)$ may be larger than k when there is more than 1 tuple whose distance from o is equal to $dis_k(x)$. For example, in Fig. 1, given $k=3$, $dis(p_3,o)=dis(p_4,o)$, and $Neigh_3(o)$ contains 4 tuples, including p_1, p_2, p_3 and p_4.

Definition 3 (*reachability distance of a tuple p w.r.t. o*). For a given parameter k, the reachability distance of a tuple p w.r.t. o is defined as

$$Rdis_k(p,o)=\max\{dis_k(o), dis(p,o)\}$$

Fig. 1 illustrates the original intention of reachability distance. Given a tuple o, if a tuple is so close to o that this tuple is a k-distance neighbor of o (*e.g.*, p_1), the reachability distance of this tuple w.r.t. o is equal to the k-distance of o. Otherwise, if a tuple is far away from o (*e.g.*, p_5), the reachability distance is the actual distance between this tuple and o.

Definition 4 (*local reachability density of a tuple*). For a given parameter k, the local reachability density of a tuple o is defined as

$$LRD_k(o)=\frac{|Neigh_k(o)|}{\sum_{p\in Neigh_k(o)} Rdis_k(o,p)}$$

Clearly, $LRD_k(o)$ is the inverse of the average of reachability distances of o w.r.t. the tuples in $Neigh_k(o)$. Next, based on the definitions above, we give the formal description of the local outlier factor of a tuple.

Table 1
Symbols in this paper.

Symbol	Signification
$dis(p,q)$	The distance between tuples p and q
$dis_k(p)$	The actual k-distance of p in P
$Ldis_k(p)$	The local k-distance of p in the grid where p belongs
$Neigh_k(p)$	The k-distance neighborhood of p
$Rdis_k(p,q)$	The reachability distance of p w.r.t. q
$LRD_k(p)$	The local reachability density of p
$LOF_k(p)$	The local outlier factor of p
$g_{x_1,x_2,\ldots,x_d}$	The grid at the x_i-th position for dimension i ($i\in[1,d]$)
$D_i(p,g)$	The distance between tuple p and grid g on the i-th dimension
$dis(p,g)$	The distance between tuple p and grid g

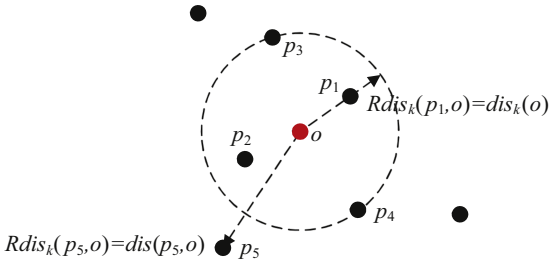

Fig. 1. k-distance neighborhood and reachability distance, for $k=3$.

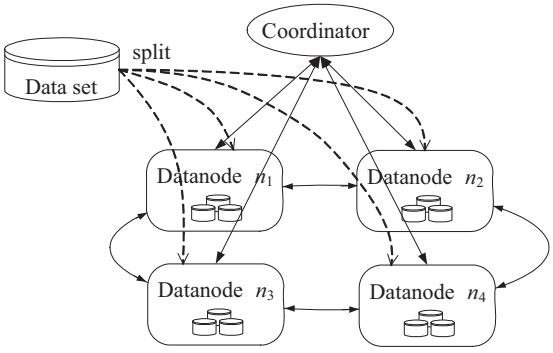

Fig. 2. Computation frame.

Definition 5 (*local outlier factor of a tuple*). Given an integer k, the local outlier factor of a tuple o is:

$$LOF_k(o)=\frac{\sum_{p\in Neigh_k(o)}\frac{LRD_k(p)}{LRD_k(o)}}{|Neigh_k(o)|}$$

Intuitively, the local outlier factor of o is the average of the ratio of the local reachability densities of o's k-distance neighbors and that of o. A large LOF $LOF_k(o)$ means o has a great degree to be an outlier. As [4] describes, *the target of the density-based outlier* is to give each tuple a degree of being an outlier using LOF, instead of determining whether a tuple is an outlier or not directly. For a concise description, we omit the subscript k from *Neigh*, *Rdis*, *LRD* and *LOF*, if there is no confusion.

3.2. Computation frame

As Fig. 2 shows, we adopt a common distributed environment that consists of a coordinator and a number of datanodes (The set of datanodes is denoted as $N=\{n_1, n_2, \ldots, n_{|N|}\}$). The coordinator is responsible for the overall scheduling. Each datanode stores a portion of the data set P, and it is the executor in outlier

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

computing. Note that most of the existing algorithms for outlier detection adopt a master-slave architecture. Lots of computations are performed on the master node. In contrast, the coordinator in our frame only takes charge of the scheduling and all the actual computations are allocated to the datanodes. Thus the coordinator would not be a bottleneck if the data scale is large.

Intuitively, the actual work for density-based outlier detection is to compute the LOF of each tuple for a given integer k. First, in the preprocessing phase, we use the proposed GBP algorithm to split the data set P into several subsets and assign them to the datanodes. Then, the DLC algorithm for the density-based outlier detection includes two main steps: (a) Each datanode processes the local tuples. LOFs of some tuples can be computed directly in the local nodes. (b) Through a few necessary network communications, we output LOFs of the rest of the tuples.

4. The description grid-based partition algorithm

4.1. Grid-based Partition

In the GBP algorithm, we first split the whole d-dimensional space into several isometrical grids. Taking into account that grid-based partition has limitations in the high-dimensional data, we use the following method to divide the whole dataset. Specifically, if the dimensionality d of the data set P is small (*e.g.*, $d \le l$), for each dimension, we cut it into several equal segments (the number of segments is denoted by s). Then, the whole space is split into s^d grids. Otherwise, if the dimensionality is large $d > l$, we randomly select l dimensions and divide these selected dimensions into s equal segments. The whole space is split into s^l grids. Note that l is a small value that we can execute grid partition in at most l dimensions. We set $l = 5$ in this paper. According to user's requirements, l can be adjusted. We set s as the smallest number that satisfies $s^d \ge |N|$ (or $s^5 \ge |N|$), where $|N|$ is the number of datanodes. Let $g_{x_1,x_2,\ldots,x_d}$ be the gird that is at the x_i-th position for dimension i ($i \in [1,d], d \le 5$, $x_i \in [1,s]$). Then the adjacent grids of $g_{x_1,x_2,\ldots,x_d}$ (denoted as $Nei(g_{x_1,x_2,\ldots,x_d})$) can be computed using the following equation:

$$Nei(g_{x_1,x_2,\ldots,x_d}) = \{g_{y_1,y_2,\ldots,y_d} \mid 1 \le x_i - 1 \le y_i \le s \le x_i + 1, g_{y_1,y_2,\ldots,y_d} \ne g_{x_1,x_2,\ldots,x_d}\}, \quad (2)$$

where $x_i, y_i \in [1,s]$ because each dimension is divided into s segments. If $x_i = 1$, $y_i \in [1,2]$. If $x_i = s$, $y_i \in [s-1,s]$. As Fig. 3 shows, there is a set P with 120 tuples in 2-dimensional space, and the number of datanodes is 10. Thus we set $s = 4$ and split the space into 16 (4^2) grids. For gird $g_{2,2}$, according to Eq. (2), all the adjacent grids can be computed, which are highlighted in grey in Fig. 3. For grid $g_{4,4}$, its adjacent grids are $g_{3,3}, g_{3,4}, h_{4,3}$.

Next, we need to allocate these grids to the datanodes. In order to accelerate the computations of density-based outliers, we design an allocation method considering the following two factors. (a) Intuitively, to acquire high parallelism, we need to *make the number of tuples on each datanode almost the same* (balance the workload). (b) According to Definition 5, we have to compute the k-distance neighborhood for each tuple. Thus it can reduce the network overhead if we *allocate the adjacent grids to the same datanodes*. The details of the method are described in Algorithm 1.

Algorithm 1. Gird allocation.

input : The grid set G, The datanode set N
output: Allocation plan

1 Sort the grids in G according to the number of tuples in a grid in the descending order.
2 **for** *each grid* g *in* G **do**
3 **if** *there exist datanodes with no grid* **then**
4 Arbitrarily choose a datanode with no grid and allocate g to it;
5 **else**
6 $\varepsilon \leftarrow$ the average number of tuples per datanode;
7 Initialize a datanode set N';
8 **for** *each datanode* n_i *in* N **do**
9 **if** *the number of tuples on* n_i *is not large than* ε **then**
10 Insert n_i into N';
11 $n \leftarrow$ choose the datanode in N' with the largest number of grids that are adjacent to g;
12 Allocate g to n;

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

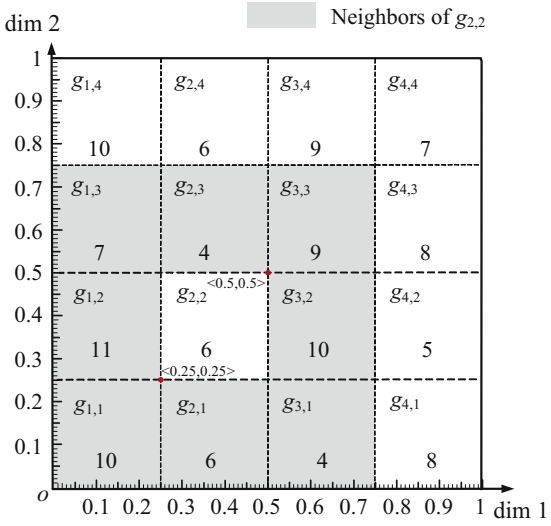

Fig. 3. Example of grids ($|N| = 10$, $|P| = 120$).

Table 2
Allocation process.

Sequence number	Grid ID	Number of tuples	Allocated datanode	Average number of tuples
1	$g_{1,2}$	11	n_1	–
2	$g_{1,1}$	10	n_2	–
3	$g_{3,2}$	10	n_3	–
4	$g_{1,4}$	10	n_4	–
5	$g_{3,4}$	9	n_5	–
6	$g_{3,3}$	9	n_6	–
7	$g_{4,3}$	8	n_7	–
8	$g_{4,1}$	8	n_8	–
9	$g_{4,4}$	7	n_9	–
10	$g_{1,3}$	7	n_{10}	8.9
11	$g_{2,2}$	6	n_{10}	9.5
12	$g_{2,4}$	6	n_5	10.1
13	$g_{2,1}$	6	n_2	10.7
14	$g_{4,2}$	5	n_8	11.2
15	$g_{3,1}$	4	n_3	11.6
16	$g_{2,3}$	4	n_6	–

In Algorithm 1, we first sort the grids according to the number of tuples in a grid in descending order (line 1). Then, for each grid g, if there exists at least one datanode with no grid, we randomly choose a datanode with no grid and allocate g to it (lines 2–4). Otherwise, we compute the average number of tuples in a datanode (denoted as ε) and initialize a datanode set N' (lines 5–7). For each datanode n_i in N, we insert it into N' if the number of tuples on n_i is smaller than or equal to ε (lines 8–10). Then, we choose the datanode n in N' with the largest number of grids that are adjacent to g and allocate g to n (lines 11 and 12).

For example, in Fig. 3, we have split the space into 16 grids and need to allocate these grids to 10 datanodes. The related number of tuples is shown at the bottom of each grid. First, we sort the grids according to the number of tuples in a grid, and the result is shown in Table 2. Then, for each of the first 10 grids, we randomly allocate it to a datanode with no grid. After that, totally 89 tuples have been allocated, and the average number of tuples per datanode is 8.9. When allocating the 11th grid $g_{2,2}$, there are 4 datanodes whose numbers of tuples are not larger than 8.9, including n_7, n_8, n_9, n_{10}. We choose n_{10} because $g_{1,3}$ is adjacent to $g_{2,2}$. Using the same method, we allocate all the grids to the corresponding datanodes.

Thus far, the process of GBP has been introduced. Clearly, we try to allocate adjacent grids to the same datanodes (lines 11 and 12 in Algorithm 1), which can reduce the network overhead. Theorem 1 illustrates another advantage of GBP.

Theorem 1 (*workload balancing on datanodes*). *Let g be the grid with the maximum number of tuples, and m_g be the number of tuples in g. Then, after GBP, the number of tuples on each datanode is smaller than $m_g + |P|/|N|$.*

Proof. Clearly, after the first allocation, the theorem is correct. Suppose the number of tuples on each datanode is smaller than $m_g + |P|/|N|$ after the i-th allocation. Then, for the $(i+1)$-th allocation, the average number of tuples per datanode is smaller than $|P|/|N|$, thus the number of tuples on the chosen datanode must be smaller than $|P|/|N|$ according to lines 9 and 10 in Algorithm 1. Meanwhile, the number of tuples in the grid that is awaiting to be allocated is not larger than m_g. Therefore, the theorem is still correct after the $(i+1)$-th allocation. □

4.2. Distributed LOF computing

In Section 4.1, we have split the data set into several grids and allocated them to the corresponding datanodes. Next, we need to compute the LOF for each tuple in parallel. By analyzing the definitions in Section 3, it is clear that LRD (Definition 4) is the premise of LOF. In order to calculate LRDs effectively, we have to compute the k-distances and k-distance neighborhoods for all the tuples first, which is the core content of this section.

Recalling Definition 1, in simple terms, the k-distance of a tuple p is the smallest distance dis that there exist at least k tuples

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

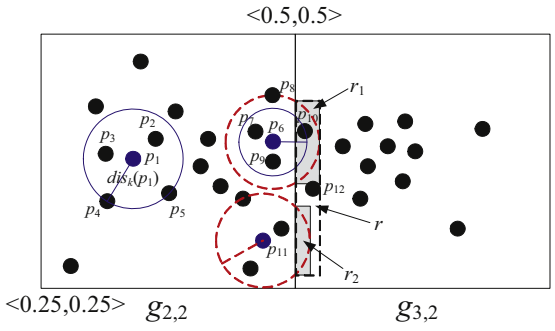

Fig. 4. Example of DLC ($k=3$).

whose distances from p are smaller than or equal to *dis*. Although Breunig et al. [4] do not give the detailed method for computing the k-distance, we can see that the k-distance is similar to the k-Nearest Neighbor problem (kNN). The method for answering kNN queries [19] can be used to compute k-distances. Specifically, we can build a spatial index (*e.g.*, R-tree) for the data in a grid. Note that when the dimensionality is high (when $d>l$, in this paper, we set $l=5$), we have chosen l dimensions to divide the grids. Thus we use the remaining $d-l$ dimensions to build the R-tree. If $d-l>l$, we randomly select l dimensions in these $d-l$ dimensions to establish the R-tree. For a tuple p, we simply perform a range query w.r.t p, then the *local* k-distance and k-distance neighborhood for p are obtained. The query processing details can be found in [3], and Fig. 4 shows an example.

Fig. 4 shows a specific view for the girds $g_{2,2}$ and $g_{3,2}$ in Fig. 3. For grid $g_{2,2}$, we build an R-tree index and perform a kNN query w.r.t p_1 where $k=3$, and tuples p_2, p_3, p_4 are returned. The local k-distance of p_1 is equal to the distance between p_1 and p_4. p_5 and p_4 have the same distance from p_1, thus the k-distance neighborhood of p_1 is $\{p_2, p_3, p_4, p_5\}$.

However, in distributed environments, the situation is more complex. We cannot compute the *actual* k-distances of all the tuples easily. For example in Fig. 4, considering the tuples in grid $g_{2,2}$, the local k-distance neighborhood of p_1 is identical to its actual k-distance neighborhood in P. However, p_6 shows a complex situation. Its local k-distance neighborhood in $g_{2,2}$ is $\{p_7, p_8, p_9\}$. Whereas, the actual k-distance neighborhood in P is $\{p_7, p_9, p_{10}\}$, which cannot be computed unless p_{10} is transmitted to gird $g_{2,2}$.

Based on the observation above, we classify the tuples in a grid into 2 categories. The formal description is as follows.

Definition 6 (*grid-local tuple and cross-grid tuple*). Given a tuple p in a grid g, we use g's minimum point g.min and maximum point g.max to represent g, and use $Ldis_k(p)$ to denote the local k-distance of p in g. If $\forall i \in [i,d], p[i]-Ldis_k(p) \geq g.\min[i]$ and $p[i]+Ldis_k(p) < g.\max[i]$, p is a grid-local tuple. Otherwise, p is a cross-grid tuple.

For example in Fig. 4, $g_{2,2}$'s minimum point is $\langle 0.25, 0.25\rangle$ and maximum point is $\langle 0.5, 0.5\rangle$. The local k-distance of p_6 is $dis(p_6, p_8)$, and $dis(p_6, p_8)+p_6[1] > 0.5$. Thus, p_6 is a cross-grid tuple. Note that in the high-dimensional space, there exist some dimensions which are not divided. Then for a dimension j which is not divided, any grid g_i, the value of g_i's minimum point in dimension j is 0 ($g_i.\min[j]=0$). And the value of g_i's maximum point in dimension j is 1 ($g_i.\max[j]=1$).

Obviously, for a grid-local tuple, the local and actual k-distances are the same. However, for a cross-grid tuple, there may exist its k-distance neighbors in the other grids. The situation is more complex. Next, we first introduce some basic notions, and then give the method to deal with the cross-grid tuples effectively.

Table 3
Parameters in real data sets.

Dataset	Number of tuples	Measurable attributes
Shuttle	43,500	9
Census	32,561	14
Covtype	581,012	54

Given a tuple p and a grid g', the minimum distance between p and g' on the i-th dimension is

$$D_i(p,g') = \begin{cases} p[i]-g'.\max[i] & \text{if } p[i] > g'.\max[i] \\ g'.\min[i]-p[i] & \text{if } p[i] < g'.\min[i] \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Thus, the minimum distance between p and g' is $dis(p,g') = \sqrt{\sum_{i\in[i,d]}(D_i(p,g'))^2}$. Clearly, it is possible that there is a k-distance neighbor of p in g' if $dis(p,g') < Ldis_k(p)$.

Theorem 2. *Given a tuple p and a grid g', p is not in g' and $dis(p,g') < Ldis_k(p)$. Then we assert that for any neighbor q of p in g', $\forall i \in [i,d], (q[i]-p[i])^2 \leq (Ldis_k(p))^2 - \sum_{j\in[i,d], j\neq i}(D_i(p,g'))^2$.*

Proof. Suppose there is a neighbor q' of p in g' and $(q'[i]-p[i])^2 > (Ldis_k(p))^2 - \sum_{j\in[i,d],j\neq i}(D_i(p,g'))^2$. According to Eq. (3), $\forall j \in [i,d]$ *and* $j \neq i$, the distance between p and q' on the j-th dimension is not smaller than $D_i(p,g')$. Thus, $dis_k(p,q') > Ldis_k(p)$, and q' is not a k-distance neighbor of p. The assumption fails, and the theorem is proved.

Based on Theorem 2, for a cross-grid tuple p and a grid g' (p is not in g'), if $dis(p,g') < Ldis_k(p)$, the *maximum distance* between p and its neighbor in grid g' on the i-th dimension, denoted by $NeiDis_i(p,g')$, can be computed using the equation below:

$$NeiDis_i(p,g') = \sqrt{(Ldis_k(p))^2 - \sum_{j\in[i,d],j\neq i}(D_i(p,g'))^2} \quad (4)$$

Next, we construct a minimum rectangle r to cover all the neighbors of p in g', which is represented by its minimum point r.min and maximum point r.max. The detailed method is as follows: (a) If $p[i] > g'.\max[i]$,

$$\begin{cases} r.\min[i] = p[i] - NeiDis_i(p,g') \\ r.\max[i] = g'.\max[i] \end{cases} \quad (5)$$

(b) If $p[i] < g'.\min[i]$,

$$\begin{cases} r.\min[i] = g'.\min[i] \\ r.\max[i] = NeiDis_i(p,g') + p[i] \end{cases} \quad (6)$$

(c) Otherwise,

$$\begin{cases} r.\min[i] = \max\{p[i] - NeiDis_i(p,g'), g'.\min[i]\} \\ r.\max[i] = \min\{NeiDis_i(p,g') + p[i], g'.\max[i]\} \end{cases} \quad (7)$$

For a cross-grid tuple p in g, we simply perform a kNN query w.r.t p and get the local k-distance $Ldis_k(p)$. For a grid g' that $dis(p,g') < Ldis_k(p)$, we use Eqs. (5)–(7) to construct a minimum rectangle r that covers all the neighbors of p in g'. For each cross-grid tuple in g who possibly has a neighbor in g', there is a related minimum rectangle. We construct an integrated rectangle in g' that covers all the minimum rectangles. The tuples in this integrated rectangle need to be sent to g. In grid g, with the help of these received tuples, we can easily compute the actual k-distances and k-distance neighborhoods for the cross-grid tuples. Algorithm 2 formally illustrates the process.

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

Algorithm 2. The DLC algorithm.

```
input : Integer k
output: LOF
1  for each tuple p in grid g do
2      Compute the local k-distance Ldis_k(p);
3      if p is a cross-grid tuple then
4          for each adjacent grid g' of g do
5              if dis(p, g') < Ldis_k(p) then
6                  Compute the related minimum rectangle in g';
7  for each adjacent grid g' of g do
8      Generate the integrated rectangle for g;
9      Send the tuples in the integrated rectangle to g;
10 for each cross-grid tuple p in g do
11     Compute the actual k-distance and k-distance neighborhood;
12 Compute LOFs for all the tuples in g;
```

For example, in Fig. 4, p_6 is a cross-grid tuple. $dis(p_6, g_{3,2}) < Ldis_k(p_6)$, thus it is possible that there is a neighbor of p_6 in $g_{3,2}$. Using Eqs. (5)–(7), we can obtain a minimum rectangle r_1 that covers all the neighbors of p_6 in $g_{3,2}$. Similarly, we obtain a minimum rectangle r_2 for p_{11}. Then, the integrated rectangle r is generated to cover r_1 and r_2. Finally, tuples p_{10}, p_{12} in r are sent to $g_{2,2}$, and we can compute the actual k-distances and k-distance neighborhoods of p_6 and p_{11}.

Thus far, we have introduced the methods to compute the k-distances for grid-local tuples and cross-grid tuples. Then, based on the definitions in Section 3, we can calculate the LRD and LOF of each tuple in parallel. The whole process of distributed density-based outlier detection is completed.

5. Experimental evaluation

We implement our proposed approaches using JAVA programming language, and evaluate the performance in a cluster where each node (coordinator or datanode) has a Intel Core i3 2100 @ 3.1 GHz CPU, 8G main memory and 500GB hard disk. The method in [4,5] is used to determine the density-based outliers. Specifically, we set the upper bound $upper = 20$, the lower bound $lower = 10$. For each $k \in [10, 20]$, we compute the related LOFs of all the tuples. Then, for each tuple p, the final LOF value $LOF_{final}(p) = \mathrm{Max}\{LOF_k(p) \mid k \in [lower, upper]\}$. The top-10 tuples with the largest final LOF values are the outliers. In the experiments, we compare our methods with PLOFA proposed by Lozano and Acufia [5]. The main performance indicators that we concern are the runtime and data transmission quantity (it is measured by the number of tuples transmitted across the network).

Table 4
Experimental results on real data sets.

Approaches	Runtime (s)	Transmission quantity
Shuttle ours(GBP+DLC)	5	1327
PLOFA	26	870,221
Census ours(GBP+DLC)	11	1027
PLOFA	63	652,372
Covtype ours(GBP+DLC)	217	15,731
PLOFA	1791	11,623,194

5.1. Results on real data set

In this section, we evaluate the performance using 3 real data sets, Shuttle, Census and Covtype (in Table 3), which are obtained available on the Machine Learning database repository at UCI ⟨http://archive.ics.uci.edu/ml/⟩ [27]. The cluster in this experiment is constituted by 1 coordinator and 3 datanodes. The results are shown in Table 4.

As Table 4 illustrates, our methods (GBP+DLC) show some significant advantages. The processing time is much smaller than that of PLOFA. Comparing with PLOFA (In PLOFA, each tuple and its respective neighborhood needs to be transmitted to the master node), we only need to transmit a small number of data across the network, because (a) In the data partition stage, GBP allocates the adjacent grids to the same datanodes. (b) In DLC, the tuples are classified into 2 categories and the network communication is required only for the cross-grid tuples.

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

5.2. Results on synthetic data sets

Due the limited size of the real data set, we generate various synthetic data sets to further analyze the performance of our approaches. Specifically, for a data set with α tuples, we randomly generate $\alpha/1000$ clustering center points. Thus the average number of tuples in a cluster is 1000. In each cluster, the tuples follow a normal distribution. The parameter settings in the experiments are summarized in Table 5.

Fig. 5(a) shows the impact of data scale on processing time. With the increase of data scale, the runtime for both PLOFA and ours becomes longer. Our approaches show better performance than the previous one. Fig. 5(b) describes the impact of data scale on transmission quantity. In PLOFA, the datanodes need to transmit the k-distance neighborhood for each tuple to the master node. The network overhead is huge, thus the performance of PLOFA is limited when processing larger-scale data. In contrast, our methods can save a lot of network resources (the detailed reason has been described in Section 5.1).

Fig. 6 illustrates the impact of number of datanodes. We can see that as more datanodes are used, both of our approaches and PLOFA take less time to compute outliers, and ours is always more efficient than PLOFA. As Fig. 6(b) shows, for both of our approaches and PLOFA, the transmission quantity is not sensitive to the variation of number of datanodes.

In Fig. 7, we evaluate the impact of dimensionality. With the increase of dimension, some operations (*e.g.*, computing the distance between two tuples) cost more time, thus the performance of our approaches and PLOFA declines. The transmission quantity changes very slightly with the increase of dimension. Because most of the tuples are grid-local tuples which can be processed in the local datanode.

Table 5
Parameter settings.

Parameters	Default	Variations
Data scale ($\times 10^8$)	1	0.2, 0.5, 1, 2, 5
Number of datanodes	20	10, 15, 20, 25, 30
Dimension	3	3, 5, 10, 15, 20

6. Conclusion

This paper focuses on the problem of density-based outlier detection in distributed environments for large-scale data sets. We first discuss that the existing solution for this problem has several weaknesses and it does not work well when processing large-scale data. Then, we propose our approaches to solve the problem which include 2 algorithms. (a) we propose the Gird-Based Partition algorithm (GBP) for data preparation. The algorithm is used to split the data set into several grids and allocate these grids to the datanodes. (b) We propose the Distributed LOF Computing method (DLC) for distributed density-based outlier computation. The algorithm first classifies the tuples in a grid into two categories: grid-local tuples and cross-grid tuples, and shows that only the computations for the cross-grid tuples require network communications. Then, a tailored method is present to compute outliers for this two types of

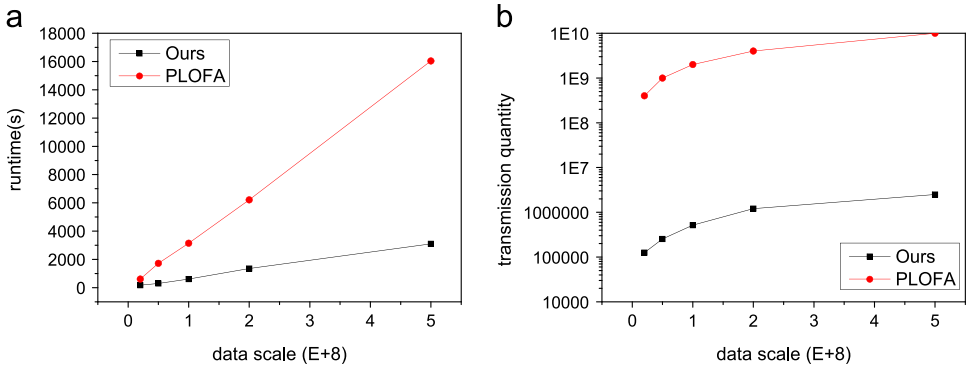

Fig. 5. The influence of data scale.

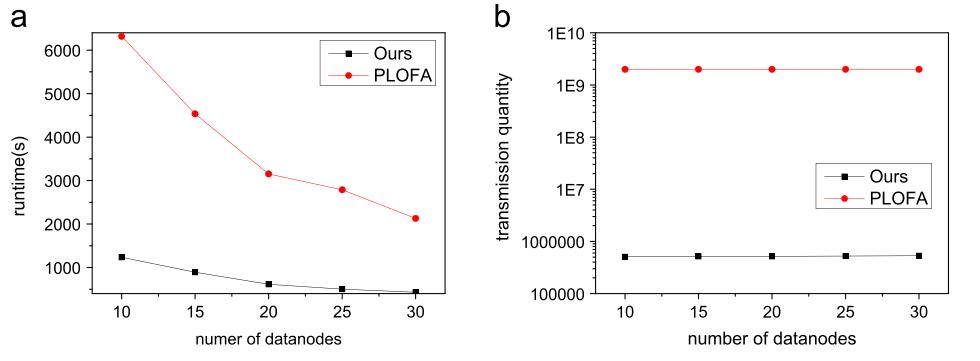

Fig. 6. The influence of number of datanodes.

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135

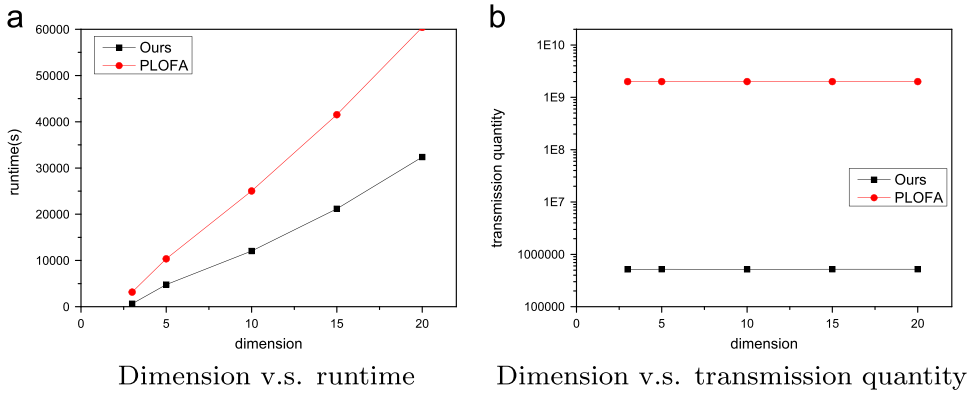

Fig. 7. The influence of dimension.

tuples, which can further save network resources. Finally, we testify the efficiency and effectiveness of the proposed approaches through a series of simulation experiments.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China under Grant nos. 61472069, 61100022, and 61402089; the National High Technology Research and Development Plan (863 Plan) under Grant no. 2012AA011004; and the Fundamental Research Funds for the Central Universities under Grant no. N130404014.

References

[1] D.M. Hawkins, Identification of Outliers, Springer, London, USA, 1980.
[2] E.M. Knox, R.T. Ng, Algorithms for mining distance-based outliers in large datasets, in: Proceedings of the International Conference on Very Large Data Bases, 1998, pp. 392–403.
[3] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, ACM SIGMOD Rec. 29 (2) (2000) 427–438.
[4] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, Lof: identifying density-based local outliers, ACM Sigmod Rec. 29 (2) (2000) 93–104.
[5] E. Lozano, E. Acufia, Parallel algorithms for distance-based and density-based outliers, in: the Fifth IEEE International Conference on Data Mining, IEEE, Houston, Texas, (2005) 729–732.
[6] P.J. Rousseeuw, A.M. Leroy, Robust Regression and Outlier Detection, John Wiley & Sons, New York, USA, 2005.
[7] V. Barnett, T. Lewis, Outliers in Statistical Data, Wiley, New York, 1994.
[8] S.D. Bay, M. Schwabacher, Mining distance-based outliers in near linear time with randomization and a simple pruning rule, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Washington DC, USA, 2003, pp. 29–38.
[9] F. Angiulli, F. Fassetti, Very efficient mining of distance-based outliers, in: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, ACM, Lisbon, Portugal, 2007, pp. 791–800.
[10] A. Guttman, A dynamic index structure for spatial searching, in: ACM SIGMOD International Conference on the Management of Data, Boston, MA, (1984) pp. 47–57.
[11] C.C. Aggarwal, S.Y. Philip, Outlier detection with uncertain data, in: SDM, SIAM, San Diego, USA, 2008, pp. 483–493.
[12] M. Kontaki, A. Gounaris, A.N. Papadopoulos, K. Tsichlas, Y. Manolopoulos, Continuous monitoring of distance-based outliers over data streams, in: 2011 IEEE 27th International Conference on Data Engineering (ICDE), IEEE, Hannover, Germany, 2011, pp. 135–146.
[13] C.C. Aggarwal, P.S. Yu, Outlier detection for high dimensional data, ACM Sigmod Rec. 30 (2) (2001) 37–46.
[14] A. Koufakou, J. Secretan, J. Reeder, K. Cardona, M. Georgiopoulos, Fast parallel outlier detection for categorical datasets using mapreduce, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IEEE, Hong Kong, China, 2008, pp. 3298–3304.
[15] Q. He, Y. Ma, Q. Wang, F. Zhuang, Z. Shi, Parallel outlier detection using kd-tree based on mapreduce, in: 2011 IEEE Third International Conference on Cloud Computing Technology and Science, IEEE, Washington DC, USA, 2011, pp. 75–80.
[16] E. Hung, D.W. Cheung, Parallel mining of outliers in large database, Distrib. Parallel Databases 12 (1) (2002) 5–26.
[17] M.E. Otey, A. Ghoting, S. Parthasarathy, Fast distributed outlier detection in mixed-attribute data sets, Data Min. Knowl. Discov. 12 (2–3) (2006) 203–228.
[18] F. Angiulli, S. Basta, S. Lodi, C. Sartori, Distributed strategies for mining outliers in large data sets, IEEE Trans. Knowl. Data Eng. 25 (7) (2013) 1520–1532.
[19] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, ACM Sigmod Rec. 24 (2) (1995) 71–79.
[20] L. Zhang, Y. Han, Y. Yang, M. Song, S. Yan, Q. Tian, Discovering discriminative graphlets for aerial image categories recognition, IEEE Trans. Image Process. (T-IP) 22 (12) (2013) 5071–5084.
[21] L. Zhang, Y. Yang, Y. Gao, Y. Yu, C. Wang, A probabilistic associative model for segmenting weakly supervised images, IEEE Trans. Image Process. 23 (9) (2014) 4150–4159.
[22] L. Zhang, Y. Gao, R. Ji, L. Ke, J. Shen, Representative discovery of structure cues for weakly-supervised image segmentation, IEEE Trans. Multimed. (T-MM) 16 (2) (2014) 470–479.
[23] L. Zhang, M. Song, Z. Liu, X. Liu, J. Bu, C. Chen, Probabilistic graphlet cut: exploring spatial structure cue for weakly supervised image segmentation, in: IEEE Computer Vision and Pattern Recognition (CVPR), Portland, USA, (2013) pp. 1908–1915.
[24] L. Zhang, M. Song, Q. Zhao, X. Liu, J. Bu, C. Chen, Probabilistic graphlet transfer for photo cropping, IEEE Trans. Image Process. (T-IP) 21 (5) (2013) 2887–2897.
[25] L. Zhang, Y. Gao, R. Zimmermann, Q. Tian, X. Li, Fusion of multi-channel local and global structural cues for photo aesthetics evaluation, IEEE Trans. Image Process. (T-IP) 23 (3) (2014) 1419–1429.
[26] L. Zhang, M. Song, X. Liu, J. Bu, C. Chen, Fast multi-view segment graph kernel for object classification, Signal Process. 93 (6) (2013) 1597–1607.
[27] C. Blake, C. Mertz, UCI repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science, ⟨http://www.ics.uci.edu/mlearn/MLRepository.html⟩, 1998.

Mei Bai received B.Sc. and M.Sc. in computer science and technology from the Northeastern University in July 2009 and July 2011, respectively. She is currently a Ph.D. candidate in the Department of Computer Science, Northeastern University. Her research interests include sensory data management and uncertain data management.

Xite Wang is a Ph.D. candidate in College of Information Science & Engineering, Northeastern University, PR China, from where he received his B.S. and M.S. degrees in 2009 and 2011, respectively. His research interests include cloud computing and big-data management.

Junchang Xin received B.Sc., M.Sc., and Ph.D. in computer science and technology from the Northeastern University in July 2002, March 2005, and July 2008, respectively. He is currently an associate professor in the Department of Computer Science, Northeastern University, PR China. His research interests include sensory data management, uncertain data management, cloud computing and machine learning.

Guoren Wang received B.Sc., M.Sc., and Ph.D. from the Department of Computer Science, Northeastern University, P.R. China, in 1988, 1991 and 1996, respectively. Currently, he is a professor in the Department of Computer Science, Northeastern University, PR China. His research interests are XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management.

Please cite this article as: M. Bai, et al., An efficient algorithm for distributed density-based outlier detection on big data, Neurocomputing (2015), http://dx.doi.org/10.1016/j.neucom.2015.05.135