A Minimum Spanning Tree-Inspired Clustering-Based Outlier Detection Technique

Xiaochun Wang[1], Xia Li Wang[2], and D. Mitch Wilkes[3]

[1] School of Electronics and Information, Xi'an Jiaotong University, Xi'an, 710049 China
xiaocchunwang@mail.xjtu.edu.cn
[2] Department of Computer Science, Changan Univeristy, Xi'an, 710061 China
xlwang@chd.edu.cn
[3] School of Engineering, Vanderbilt University, Nashville, TN 37235 USA
mitch.wilkes@vanderbilt.edu

Abstract. Due to its important applications in data mining, many techniques have been developed for outlier detection. In this paper, an efficient three-phase outlier detection technique. First, we modify the famous *k*-means algorithm for an efficient construction of a spanning tree which is very close to a minimum spanning tree of the data set. Second, the longest edges in the obtained spanning tree are removed to form clusters. Based on the intuition that the data points in small clusters may be most likely all outliers, they are selected and regarded as outlier candidates. Finally, density-based outlying factors, LOF, are calculated for potential outlier candidates and accessed to pinpoint the local outliers. Extensive experiments on real and synthetic data sets show that the proposed approach can efficiently identify global as well as local outliers for large-scale datasets with respect to the state-of-the-art methods.

Keywords: distance-based outlier detection, density-based outlier detection, clustering-based outlier detection, minimum spanning tree-based clustering.

1 Introduction

Human species cannot survive without the ability to discover anomalous patterns in observed data that do not conform to the expected normal behavior. In statistics, the anomalous patterns are often referred to as outliers. In a classic definition, "an outlier is an observation that deviates so much from other observations that it aroused suspicions that it is generated by a different mechanism" [1]. Due to its important applications, outlier detection has found immense use in a wide variety of practical domains, such as intrusion detection for cyber-security [2,3], fraud detection for credit cards, insurance and tax [4], early detection of disease outbreaks in the medical field [5], fault detection in sensor networks for monitoring health, traffic, machine status, weather, pollution, surveillance [6], and so on [7,8].

As a result, the task of finding outliers in a data set has long been an area of active research, and different outlier detection techniques, such as distribution-based, depth-based, distance-based, density-based and clustering-based approaches, have been

P. Perner (Ed.): ICDM 2012, LNAI 7377, pp. 209–223, 2012.

developed and no techniques are completely satisfactory for all the situations. Originating from statistics, distribution-based methods study the outlier detection problem in the context of a given distributional model [9,10,11], which is usually not known a priori for modern large databases. As a result, they have limited usefulness. Stemming from Computational Geometry, depth-based methods organize the observations into layers through the computation of *k*-dimensional convex hulls, believing that the shallow layers are more likely to contain outliers than deep ones. But these methods are very computationally expensive for more than a few dimensions [12]. Distance-based methods and density-based methods actually address the detection problems of two different notions of outliers: more globally-oriented outliers (the former) and more locally-oriented outliers (the latter), and that clustering-based approaches obtain outliers as the by-products of clustering, that is, outliers are the data items that reside in the smallest clusters. Other types of outlier detection algorithms have also been developed to look into the problem from different aspects [13,14,15]. However, they are beyond the scope of this study and will not be discussed further here.

In this paper, we propose an in-memory fast outlier detection method which integrates an efficient minimum spanning tree (MST) based clustering algorithm [16] with outlier concepts [10] for modern large high-dimensional datasets. Basically, our CPU efficient MST-inspired outlier detection algorithm has three phases. In the first phase, a spanning tree very close to an MST is constructed. In the second phase, the longest edges in the spanning tree are identified and removed to form clusters as the standard MST-based clustering algorithms do, and the data points in those small clusters are selected as outlier candidates. In the third phase, the algorithm assigns an LOF to a small number of outlier candidates discovered in the second phase so as to utilize the density-based outlier detection technique to selectively mine the local outliers. Our contributions include:

- Extensive experimental evaluation on both synthetic and real datasets demonstrates the meaningfulness of our approach as a very efficient way for the detection of global and local outliers.
- The proposed method combines the advantages of distanced-based, density-based and clustering-based outlier detection techniques to give a better intuition to view such techniques.
- Compared to the state-of-art distance-based algorithms which need some parameters to be provided by the user, the proposed outlier detection methods overcomes this limitation, thus proving to be an effective solution in real applications where a completely unsupervised method is desirable.
- As far as local outliers are concerned, LOF is computationally expensive. However, our algorithm checks distance ranking and only calculates LOF when distances are distributed evenly, that is, when there is not very much density differences. By this way, only a relatively small computation is sufficient for preserving the good quality of the detection results.

The rest of the paper is organized as follows. In Section 2, we review some existing work on distance-based, density-based and clustering-based outlier detection

algorithms. We next present our proposed approach in Section 3. In Section 4, an empirical study is conducted to evaluate the performance of our algorithm with respect to some state-of-the-art outlier detection algorithms. Finally, conclusions are made and future work is indicated in Section 5.

2 Related Work

There are three parts of the unsupervised outlier detection literature that are related to our study: distance-based outlier detection, density-based outlier detection and clustering-based outlier detection.

2.1 Distance-Based Outlier Detection

Proposed by Knorr and Ng, distance-based outlier detection methods provide a good way to detect outliers residing in relatively sparse regions. Given a distance measure on a feature space, the notion of outliers studied by Knorr and Ng is defined as: "An object O in a dataset T is a distance-based outlier, denoted by DB(p, D)-outlier, if at least a fraction p of the objects in T lies greater than distance D from O, where the term DB(p, D)-outlier is a shorthand notation for a Distance-Based outlier (detected using parameters p and D)" [11]. Beginning with this work, various versions of distance-based outlier definition have been developed. Three popular ones are:

1. Given a real number d and an integer p, a data item is an outlier if there are fewer than p other data items within distance d [11,12].
2. Given two integers, n and k, outliers are the data items whose distance to their k-th nearest neighbor is among top n largest ones [17].
3. Given two integers, n and k, outliers are the data items whose average distance to their k nearest neighbors is among top n largest ones [18,19].

The first definition does not give a ranking but requires the specification of a distance parameter d, which may involve trial and error to guess an appropriate value [17]. Eliminating this requirement, the second definition only considers the distance to the k-th nearest neighbor and ignores information about closer points. The last definition accounts for the distances to k nearest neighbors and, thus, is slower to calculate than the first two.

In [11], three approaches were proposed for DB(p, D)-outliers: a $O(dN^2)$ block-oriented nested loop algorithm, a $O(N\log N)$ index-based algorithm for low dimensions [10] and cell-based algorithm (with a time complexity increasing exponentially with the dimensionality). However, these algorithms can be used to detect outliers efficiently only for low dimensional data sets. Theoretically, if a (reasonably tight) cutoff threshold can be determined efficiently, for most normal data, a partial search through the database is enough to determine it is not an outlier. Only for data objects that are potential outlier candidates can a full scan through the database be necessary. Based on this idea, distance-based outlier detection algorithms proposed thereafter (ORCA method [20], RBRP method [21], DHCA-based methods [22,23]) have been

focusing on determining the cutoff threshold efficiently and reducing the partial search sufficiently, by randomizing data and/or using some data structures.

2.2 Density-Based Outlier Detection

Distance-based outlier detection techniques work well for detecting global outliers in simply-structured data sets that contain one or more clusters with similar density. However, for many real world data sets which have complex structures in the sense that different portions of a database can exhibit very different characteristics, they might not be able to find all interesting outliers. A classic two-dimensional illustration to show this deficiency is shown in Fig. 1.

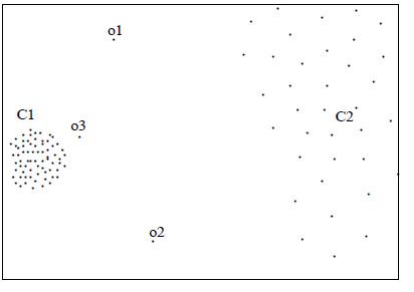

Fig. 1. Two clusters with different densities

This data set contains a dense cluster, C_1, a sparse cluster, C_2, and three outstanding objects. Obviously, the two well separated objects o1 and o2 are outliers no matter whether we look from a global or local point of view. However, according to any distance-based outlier definitions, object o3 can not be detected as an outlier, without all the objects in cluster C_2 being detected as outliers, though, from a more local point of view, it should be detected as an outlier. To deal with this situation, Breunig et al. pioneered the density-based outlier detection research by assigning to each object a degree of being an outlier, called the Local Outlier Factor (LOF), for judging the outlyingness of every object in the data set based on ratios between the local density around an object and the local density around its neighboring objects [10].

The LOF method works by first calculating the LOF for each object in the data set. Next, all the objects are ranked according to their LOF values. Finally, objects with top-n largest LOF values are marked as outliers. Calculating an exact LOF for each data object can be computationally expensive. To solve this problem, Jin et al. proposed to use the concept of micro-clusters to efficiently mine top-n LOF-based outliers in large databases [24]. As a further extension, the algorithm presented in [25] uses the reverse nearest neighbors additionally and considers a symmetric relationship between both values as a measure of outlyingness. Several other extensions and refinements have been proposed, including a Connectivity-based Outlier Factor (COF) [26], Local Outlier Integral (LOCI) [9], and a Spatial Local Outlier Measure (SLOM) [27]. The main difference between LOF and LOCI is that the former uses k nearest neighbors while the latter uses ε-neighborhoods.

2.3 Clustering-Based Outlier Detection

A problem associated with distance-based as well as density based outlier detection algorithms is their strong sensitiveness to the setting of some parameters. This can be illustrated by a 2-dimensional data set shown in Fig. 2. For distance-based outlier detection techniques, if $k = 6$ nearest neighbors are considered, all the points in cluster C3 will not be detected as outliers, while if $k = 7$, all the data points in cluster C3 are regarded as outliers. Similar problems exist for density-based outlier detection techniques. The situation could be worse for the detection of outliers in high-dimensional feature space since data points cannot be visualized there.

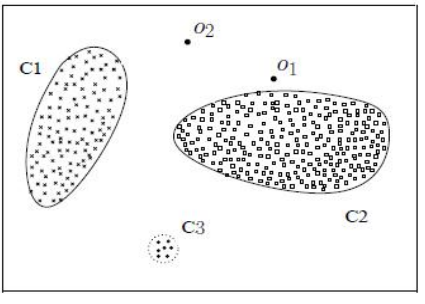

Fig. 2. Sample clusters in a 2-D data set

This is where clustering algorithms can be of some help. Being a very important data mining tool, the main concern of clustering algorithms is to find clusters by optimizing some criterion, such as minimizing the intra-cluster distance and maximizing the inter-cluster distance. As a by-product, data items in small groups can often be regarded as outliers (noise) that should be removed to make clustering more reliable. Classic clustering algorithms, such as *k*-means algorithm or PAM, rely on grouping the data points around some "centers" and do not work well when the boundaries of the clusters are irregular. As an alternate, graph theory based methods, typified by the MST-based clustering algorithms, can find clusters with irregular boundaries.

A minimum spanning tree is a connected weighted graph with no closed paths that contains every point in the data set and has the minimal total weight. If a weight denoting a distance between two end points is assigned to each edge, any edge in an MST will be the shortest distance between two subtrees that are connected by that edge. This fact is referred to as the cut property of MSTs. Therefore, removing the longest edges corresponds to choosing the breaks to form clusters. Based on this finding and initially proposed by Zahn [28], MST-based clustering algorithms have so far been extensively studied [17]. Regarding data points in smallest clusters formed by cutting the longest edges in an MST may likely be outliers, several MST-based outlier detection techniques have been proposed [29,30,31]. But these MST-based clustering algorithms are very computationally expensive (from $O(N\log N)$ to $O(N^2)$).

Recently, an MST-based clustering algorithm with a near linear performance has been developed [16], and can be modified to suit for our purpose. Basically, their

CPU efficient MST-based clustering algorithm has three phases. First, a simple spanning tree is constructed. Secondly, in order to quickly identify the longest edges, a modification of the classic *K*-means algorithm, called divisive hierarchical clustering algorithm (DHCA), is used to generate a spanning tree very close to a true minimum spanning tree. Finally, the cut and the cycle property of the minimum spanning trees are used to perform the clustering. The main advantage of the algorithm is its computation efficiency.

For a given data set, DHCA starts with *K* randomly selected centers, and assigns each point to its closest center, creating *K* partitions. Then, for each of these *K* partitions, it recursively selects *K* random centers and continues the clustering process within each partition to form at most K^n partitions for the nth stage. In our implementation, the procedure continues until the number of elements in a partition is below $K + 2$, at which time, the distance of each data item to other data items in that partition can be updated with a smaller value via a brute-force nearest neighbor search. Such a strategy ensures that points that are close to each other in space are likely to be collocated in the same partition. A detailed demonstration and proof of the effectiveness of DHCA on *k*-nearest neighbors search has been given in [16].

3 An Improved MST-Based Outlier Detection Algorithm

According to the cut property, MST-based clustering algorithms can be very useful in detecting small clusters that can be selected and regarded as outliers. For example, after removing the first three longest edges of an MST constructed for the data set shown in Fig. 1, four clusters result and global outliers such as o1 and o2 can be immediately recognized as small clusters. Even for local outlier o3, after clustering, it can be detected as a global outlier from the viewpoint of cluster C1. However, most MST-based clustering algorithms require a complete MST be constructed in the first step. For modern databases typically consisting of millions of high dimensional data items, this can be very computationally expensive. Motivated by the approach presented in [17], in this section, we describe a new scheme to facilitate efficient MST-based outlier detection for modern large databases, which is based on the observation that, for some MST-based clustering approaches, if we can find the longest edges in an MST very quickly, there is no need to compute the exact distance values associated with the shortest ones. Therefore, for cases where the number of the longest edges that separate the clusters (including the noisy ones) can be much fewer than the number of the shorter ones (e.g., several dense clusters and a small number of outlier groups), MST-based outlier detection can be more efficient if the longest edges can be identified quickly before most of the shorter ones are found, which allows us to reduce the number of distance computations to a value smaller than $O(N^2)$.

3.1 A Simple Idea

Based on the spanning tree (which is very close to a true MST) constructed following the first two stages of the approach presented in [16] (i.e., sequential initialization and DHCA updates), we then search for the edge that has the largest value, called the

potential longest edge candidate, and cut it to form two partitions. If the number of data items in one partition is no more than p, we check whether there exists another edge with a smaller weight crossing the two partitions connected now by this potential longest edge candidate. This can be done by no more than $p(N\text{-}p)$ number of distance computations. If the result shows that there exists no such edge, we declare these data items to be outliers. Otherwise, we record the update (i.e., replace the potential longest edge candidate with the edge of a smaller weight crossing the two partitions) and start another search for the longest edge candidate in the MST. The point is that the current potential longest edge candidate is the longest edge in the tree so far that connects the only partition with no more than p data items to one of the partitions with more than p data items and provides an upper bound to the distance between the two partitions now connected by it. Since the number of outliers is expected to be relatively small, the number of distance computations consumed is expected to be relatively small as well The working idea behind our efficient MST-based outlier detection algorithm is that some of the longest edges do not correspond to any cluster separations or breaks but are associated with the outliers.

3.2 Detecting Local Outliers

As mentioned previously, data sets under consideration may have complex structures in the sense that different portions of a database can exhibit very different density characteristics. For these cases, the measure of outlyingness only on simple distances between data points is not sufficient. As a further improvement, we take a step further by calculating LOF for each outlier candidate until the desired number of outliers are discovered. The LOF computation for every data point can be expensive. Fortunately, many indexing structures proposed in recent years for higher dimensional feature space can be utilized [32]. We are particularly interested in one called iDistance [39], which consists of three steps. First, the data are clustered into a set of partitions. Second, a reference point is identified for each partition. Finally, all data points are represented into a single dimensional space by indexing them based on the distance from the nearest reference point. In our approach, we choose the origin as the reference point. When the database is read in, the distance of each data item to the reference point is calculated. Next, the data items are sorted according to their distances. To be used as a search structure, given a data point, the search starts from its position in the sorted distances and proceeds bi-directionally along the radius axis until the k^{th} nearest neighbor is found. Thus, the use of the search structure can reduce the $O(N)$ time full search to a faster partial search. It is easy to see that the search structure can be constructed by any one dimensional sorting algorithm.

3.3 Our MST-Clustering Based Outlier Detection Algorithm

Our MST-based outlier detection algorithm can be summarized in the following:

1. initialize a spanning tree sequentially, that is, each data item can be assigned the distance between itself and its immediate predecessor.

4. refine the spanning tree by running DHCA multiple times until the percentage difference between two consecutive tree weights is below a threshold, say 0.001.
5. identify the potential longest edge candidate and remove it to form new clusters in a recursive fashion until the number of data items in the smallest cluster is below a predefined size of the largest outlier cluster (denoted by p here).
6. for each data item in the smallest cluster, calculate its distance to every other data item that is not in its cluster, which is N-p number of distance computations.
7. if no edges with a smaller weight crossing the smallest cluster to the rest of data set exist, we declare these data items to be outlier candidates, otherwise we update each data point to its nearest neighbor in the rest of data set.
8. calculate a LOF for each outlier candidates and rank the LOF scores.
9. if a required number of outliers are found, stop, otherwise, start another round of search, i.e., go to 3

To summarize, the numerical parameters the algorithm needs from the user include the data set, the loosely estimated numbers of outliers, the input K to DHCA, and the number of nearest neighbors, k, for LOF calculation, while the outputs will be the ranked outliers. From the algorithm description, we expect the time spent on the first phase (including the sequential initialization and the DHCA updates) to scale as $O(fN\log N)$, where f denotes the number of DHCAs constructed. The second phase partitions the obtained approximate minimum spanning tree to locate the potential outliers. Since the number of desired outliers is much smaller than the data set size N, we expect the second phase to scale as $O(Ne)$, where e denotes the number of data points checked. We expect the third phase to scale as $O(ek^2\log N)$, where k denotes the number of nearest neighbors predefined to calculate LOF. Therefore, the average time complexity of our algorithm is $O(fN\log N + Ne + ek^2\log N)$, though the worst case could still be $O(N^2)$.

4 A Performance Study

In this section, we present the results of an experimental study performed to evaluate our MST-clustering based outlier detection algorithm. First, we select four 2-dimensional outlier detection problems and compare the performance of our proposed algorithm to that of the MST-based clustering method, the ORCA method and the LOF method. For this comparison, we would like to show that our MST-inspired clustering-based algorithm can outperform the classic outlier detection algorithms in both the execution time and the classification accuracy. We also study the behavior of the proposed algorithm with various parameters and under different workloads. Finally, we evaluate our algorithm on several real higher dimensional large data sets with no assumptions made on the data distribution and compare it with three state-of-the-art outlier detection algorithms to check the technical soundness of this study. All the data sets are briefly summarized in Table 1.

We implemented all the algorithms in C++ and performed all the experiments on a computer with Intel Core 2 Duo Processor E6550 2.33GHz CPU and 2GB RAM. The operating system running on this computer is Ubuntu Linux. We use the timer utilities

defined in the C standard library to report the CPU time. In our evaluation, we use the total execution time in seconds and the accuracy of the detected outliers as the performance metric. Each result we show was obtained as the average value over 10 runs of the program for each data set. In all the experiments, the total execution time account for all three phases of our MST-inspired outlier detection algorithm. The results show the superiority of our MST-inspired algorithm over the other algorithms.

Table 1. Descriptions of all data sets

Data Name	Data Size	Dimension	# of outliers	RT(s) of MST	RT(s) of LOF
Data11	10,180	2	28	23	N/A
Data12	20,360	2	56	88	N/A
Data13	30,540	2	84	198	N/A
Data21	11,550	2	30	28	N/A
Data22	23,100	2	60	114	N/A
Data23	34,650	2	90	253	N/A
Data31	16,165	2	34	N/A	89
Data32	32,330	2	68	N/A	360
Data33	48,495	2	102	N/A	798
Data41	20,066	2	36	N/A	137
Data42	40,132	2	72	N/A	546
Data43	60,198	2	108	N/A	1215
Corel	68,040	32	N/A	4,771	13,947
IPUMS	88,443	61	N/A	14,992	43,032
ourData	65,798	10041	N/A	31,610	51,938
Covertype	581,012	55	N/A	602,911	N/A

4.1 Performance of Our Algorithm on Synthetic Data

In this subsection, we investigate the relative performance of our proposed algorithm on four outlier detection tasks shown in Fig. 3. Data11 contains two clusters and 28 global outliers. Data21 contains two curving irregularly shaped bands, and 30 global outliers. To clearly demonstrate the advantage of our algorithm over classic distance-based outlier detection algorithms on large data sets, we make the following assumptions for these two tasks: the densities of the clusters are similar. Data31 contains two

clusters of different densities, with 219 data points in the lower-density cluster, and some local outliers. Data41 contains two clusters of different densities as well, with 1581 data points in the lower-density cluster, and some local outliers. To test the running time (RT) scalability of our algorithm with the size of the data sets, we double and then triple these data sets to form their 4-cluster versions (Data12, Data22, Data32 and Data42) and 6-cluster versions (Data13, Data23, Data43 and Data33), respectively.

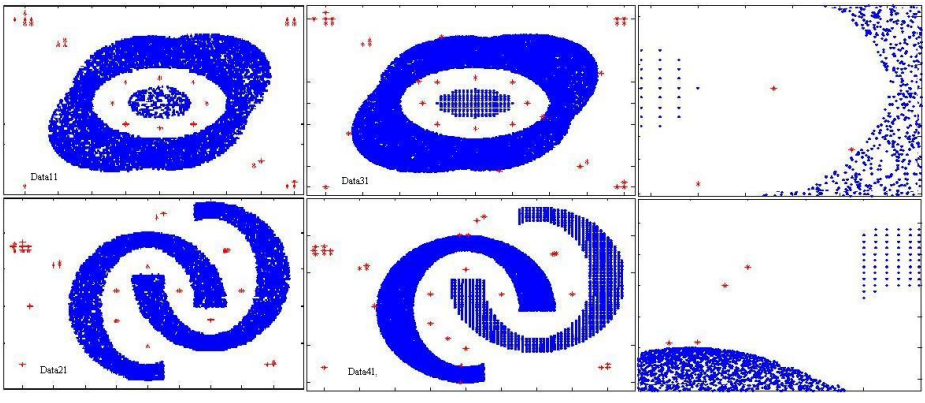

Fig. 3. (a) Data11 and Data21, (b) Data31 and Data41, (c) two snapshots of local outliers

We first study the effect of K, the input to the DHCA, and the impact of the dimensionality on the performance of our algorithm. Setting the largest number of data points in an outlying group to be 10, we varied K from 3 to 30 for Data13 and Data23.

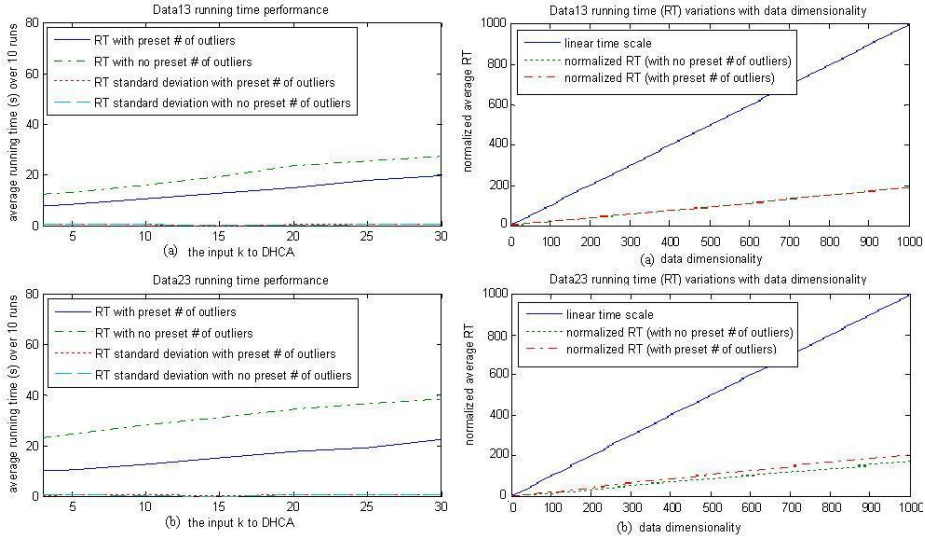

Fig. 4. Impacts of (left) K to DHCA (right) dimensionality on our algorithm

From the graphs on the left side of Fig. 4, we can see the impact of input to DHCA, K, on the efficiency of our algorithm. Overall, when K is small, the overhead of constructing the DHCA dominates. For large K, more distance computations to the partition centers are involved and the increases in the distance computations eventually dominate. Since the number of clusters and the longest edges are relatively small compared to the number of data points in the data sets, most nearest neighbor distances obtained using DHCA are much smaller than the longest ones and, therefore, our algorithm has a better scalability to the data size than the other algorithms. To see the impact of the dimensionality of data on our algorithm, we increase the data dimensionality by appending each data point to itself to result in a similarly distributed data set of a higher dimension. The results in terms of the running time (RT) on the first two data sets with extended dimensionality are shown on the left of Fig. 4. The runs were done to mine the outliers for $K = 5$ and the largest number of data points in an outlying group being set to be 10. It can be seen that all the running time increases with the size of the dimensionality in a linear fashion.

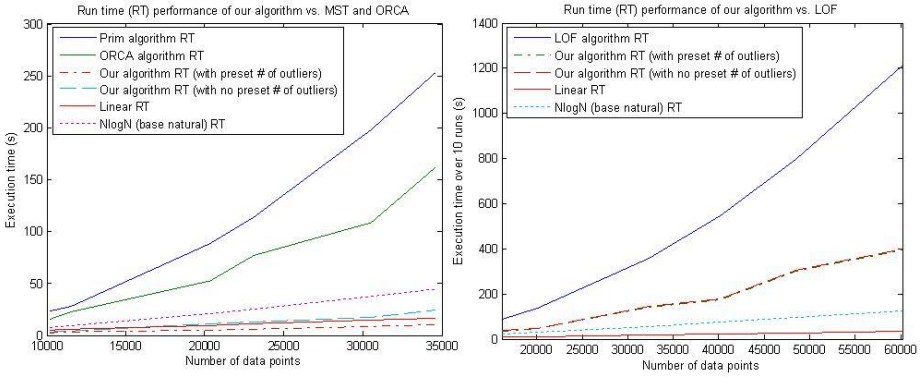

Fig. 5. Run time performance of our algorithm vs. (left) MST & ORCA (right) LOF

We next investigate the relative performance of our algorithm with respect to the classic MST- clustering based algorithm and the ORCA algorithm with/without the number of outliers given on Data11 through Data23. This set of experiments is run with K for the DHCA being set to be 5 and the largest number of data points in an outlying group being set to be 10. If the number of outliers is not given beforehand, our algorithm stops when the performance levels off, i.e., when there appears a big jump (between the $(n\text{-}1)^{\text{th}}$ and the n^{th} longest edges in this experiment) as measured by above 50% of the $(n\text{-}1)^{\text{th}}$ longest edge. This is a direct result of our assumptions made upon the data distribution. The running time results are presented on the left of Fig. 5. We then investigate the relative performance of our proposed algorithm with respect to the LOF algorithm on Data31 through Data43 when some local outliers also exist. The running time performance is shown on the right of Fig. 5.

In both graphs, the top lines represent the running time of the Prim's MST algorithm and that of the LOF algorithm, respectively, and, clearly, they increase with the

data set sizes in a quadratic form. The expected execution time to find the small number of outliers given an $N\log N$ time algorithm and a linear time algorithm are extrapolated from the running time consumed by the Prim's algorithm and the LOF method, respectively. From the left figure, it can be seen that our algorithm outperforms the Prim's algorithm by an order of magnitude in running time and exhibits near linear scalability with the data sizes with 100% correct detection rate. From the right figure, it can be seen that our algorithm outperforms the LOF algorithm by a factor between 3.0 and 4.0 with 100% correct detection rate.

4.2 Performance of Our Algorithm on Real Data

In this subsection, we investigate the relative performance of our proposed algorithm on four real data sets. This set of experiments is run to mine top 100 outliers. First, we show the impact of the input K to DHCA on the run time performance. Setting the largest number of data points in an outlying group to be 30, we varied K from 3 to 30. The results are shown on the left side of Fig. 6 and agree with our observation for the synthesis data sets.

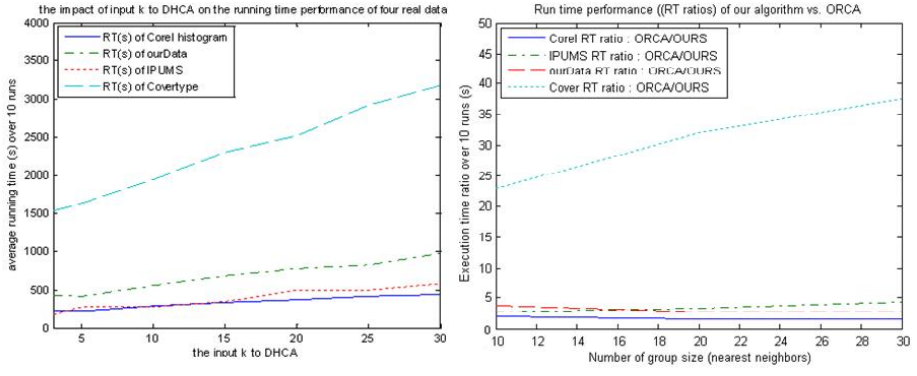

Fig. 6. (left) Impacts of K to DHCA (right) run time performance of our algorithm vs. ORCA

Setting the input value of K for the DHCA to be 5, we varied the largest number of data points in an outlying group from 10 to 30 to test the run time performance of our algorithm against the ORCA method. From the results shown on the right of Fig. 6, we can see our algorithm outperforms the ORCA method in all group sizes (or k, the number of nearest neighbors). Also we observe that the running time of our algorithm on ourData actually decreases with the increase of the outlying group size. This is intuitive because, when the outlying group size is set small, groups with a little larger size will be regarded as non-outlying groups, showing the advantage of clustering based outlier detection technique.

The run time performance of our algorithm vs. the MST-clustering based algorithm for the largest group size (i.e., 30) is summarized in Table 2, together with the $O(N\log N)$ (base natural) run time, which is extrapolated from the running time

consumed by the MST-clustering based method. From the table, we can see that our algorithm outperforms MST by an order of magnitude (with 100% correct detection rate) but is near the $O(N\log N)$ running time performance.

Then we investigate the relative performance of our proposed algorithm with respect to the LOF method on the first three real data sets. This set of experiments is run to mine top 100 outliers with the largest number of data points in an outlying group being set to 30. The results are shown in Table 3. From the table it can be seen that our algorithm outperforms the LOF method and has better performance when the dimensionality of the data set increases. In other words, the distance computation eventually dominates our algorithm for higher and higher dimensional data sets.

Table 2. Run time performance of OURS vs. MST

Data Name	MST/OURS	$N\ln N$/OURS
Corel histogram	9.6	0.59
IPUMS	54.9	2.15
OURS	44.1	1.29
Covertype	193.4	1.66

Table 3. Run time performance of OURS vs. LOF

Data Name	LOF/OURS	Dimensionality	Data Size
Corel histogram	1.6	32	68,040
IPUMS	3.2	61	88,443
ourData	5.1	90	65,798

Finally, the agreement of number of outliers detected using the MST-clustering based method, the ORCA method, the LOF method and our method are summarized in Table 4.

Table 4. Agreement of the number of outliers between different methods

Data Name	MST/OURS	ORCA/OURS	LOF/OURS
Corel histogram	100%	53%	73%
IPUMS	100%	59%	70%
OURS	100%	71%	65%

5 Conclusion

As a graph partition technique, MST-based clustering algorithms are of growing importance in detecting outlier clusters. A central problem in such applications in large high-dimensional data mining is usually its $O(N^2)$ time complexity. In this paper, we have presented a new MST-inspired outlier detection algorithm for large data sets by utilizing a divisive hierarchical clustering algorithm which makes the implementation

of the clustering-based outlier detection more efficiently. Additionally, our algorithm can be combined with density-based outlier concept to mine local outliers.

We conducted an extensive experimental study to evaluate our algorithm against the state-of-the-art outlier detection algorithms. Our experimental results show that our proposed MST-inspired clustering-based outlier detection algorithm is very effective and works reasonable well on all the data sets presented. However, the issue still remains that our algorithm, the ORCA method and the LOF method are based on different outlier definitions. From our observation, there exist some agreements as well as differences on the retrieved top outliers between these definitions. Since there often exist some structures in the data sets, the terminating condition of our algorithm should be further studied to unify all the outlier definitions.

References

1. Hawkins, D.M.: Identification of Outliers, Monographs on Applied Probability and Statistics. Chapman and Hall, London (1980)
2. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In: Data Mining for Security Applications (2002)
3. Lane, T., Brodley, C.E.: Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security 2(3), 295–331 (1999)
4. Bolton, R.J., David, J.H.: Unsupervised Profiling Methods for Fraud Detection. Statistical Science 17(3), 235–255 (2002)
5. Wong, W., Moore, A., Cooper, G., Wagner, M.: Rule-based Anomaly Pattern Detection for Detecting Disease Outbreaks. In: Proceedings of the 18th National Conference on Artificial Intelligence (2002)
6. Sheng, B., Li, Q., Mao, W., Jin, W.: Outlier detection in sensor networks. In: Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 219–228 (2007)
7. Hodge, V.J., Austin, J.: A Survey of Outlier Detection Methodologies. Artificial Intelligence Review 22, 85–126 (2004)
8. Chandola, V., Banerjee, A., Kumar, V.: Anomaly Detection: A Survey. ACM Computing Surveys 41(3), article 15 (2009)
9. Gibbons, P.B., Papadimitriou, S., Kitagawa, H., Christos Faloutsos, C.: LOCI: Fast Outlier Detection Using the Local Correlation Integral. In: Proceedings of the IEEE 19th International Conference on Data Engineering, Bangalore, India, pp. 315–328 (2003)
10. Breuning, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying Density-Based Local Outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
11. Knorr, E.M., Ng, R.T.: Algorithms for Mining Distance-Based Outliers in Large Datasets. In: Proceedings of the 24th VLDB Conference, New York, USA, pp. 392–403 (1998)
12. Knorr, E.M., Ng, R.T.: Finding intensional knowledge of distance-based outliers. In: Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, UK, pp. 211–222 (1999)
13. Angiulli, F., Pizzuti, C.: Outlier mining in large high dimensional datasets. IEEE Transactions on Knowledge and Data and Engineering, 203–215 (2005)
14. Niu, K., Huang, C., Zhang, S., Chen, J.: ODDC: outlier detection using distance distribution clustering. In: HPDMA 2007 in Conjunction with PAKDDd 2007, pp. 332–343 (2007)

15. Kreigel, H.P., Schubert, M., Zimek, A.: Angle-based outlier detection in high-dimensional data. In: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, pp. 444–452 (2008)
16. Wang, X., Wang, X.L., Wilkes, D.M.: A Divide-And-Conquer Approach For Minimum Spanning Tree-Based Clustering. IEEE Transactions on Knowledge and Data Engineering 21(7), 945–958 (2009)
17. Knorr, E.M., Ng, R.T., Tucakov, V.: Distance-based outliers: algorithms and applications. VLDB Journal: Very Large Databases 8(3-4), 237–253 (2000)
18. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: Proceedings of the ACM SIGMOD Conference, pp. 427–438 (2000)
19. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: Proceedings of the Sixth European Conference on the Principles of Data Mining and Knowledge Discovery, pp. 15–26 (2002)
20. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: KDD 2003, pp. 29–38 (2003)
21. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high-dimensional datasets. In: SDM 2006, pp. 608–612 (2006)
22. Wang, X., Wang, X.L., Wilkes, D.M.: A fast distance-based outlier detection technique. In: Poster and Workshop Proceedings of 8th Industrial Conference on Data Mining, Leipzig, Germany, pp. 25–44 (July 2008)
23. Wang, X., Wang, X.L., Wilkes, D.M.: Application of two partial search methods to Euclidean distance-based outlier detection. In: Proceedings of the 2008 International Conference on Data Mining, Las Vegas Nevada, USA, July 2008, pp. 420–426 (2008)
24. Jin, W., Tung, A.K.H., Han, J.: Mining top-n local outliers in large databases. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, California, USA, pp. 293–298 (2001)
25. Jin, W., Tung, A.K.H., Han, J., Wang, W.: Ranking Outliers Using Symmetric Neighborhood Relationship. In: Ng, W.-K., Kitsuregawa, M., Li, J., Chang, K. (eds.) PAKDD 2006. LNCS (LNAI), vol. 3918, pp. 577–593. Springer, Heidelberg (2006)
26. Tang, J., Chen, Z., Fu, A.W.-c., Cheung, D.W.: Enhancing Effectiveness of Outlier Detections for Low Density Patterns. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS (LNAI), vol. 2336, p. 535. Springer, Heidelberg (2002)
27. Sun, P., Chawla, S.: On local spatial outliers. In: Proceedings of the 4th International Conference on Data Mining (ICDM), Brighton, UK (2004)
28. Zahn, C.T.: Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. IEEE Transactions on Computers C-20, 68–86 (1971)
29. Rohlf, F.J.: Generalization of the gap test for the detection of multivariate outliers. Biometrics 31, 93–101 (1975)
30. Jiang, M.F., Tseng, S.S., Su, C.M.: Two-Phase Clustering Process for Outliers Detection. Pattern Recognition Letters 22, 691–700 (2001)
31. Lin, J., Ye, D., Chen, C., Gao, M.: Minimum Spanning Tree Based Spatial Outlier Mining and Its Applications. In: Wang, G., Li, T., Grzymala-Busse, J.W., Miao, D., Skowron, A., Yao, Y. (eds.) RSKT 2008. LNCS (LNAI), vol. 5009, pp. 508–515. Springer, Heidelberg (2008)
32. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Transactions on Data Base Systems (TODS) 30(2), 364–397 (2005)