Efficient Pruning Schemes for Distance-Based Outlier Detection

Nguyen Hoang Vu and Vivekanand Gopalkrishnan

Nanyang Technological University, 50 Nanyang Avenue, Singapore
ng0001vu@ntu.edu.sg, asvivek@ntu.edu.sg

Abstract. Outlier detection finds many applications, especially in domains that have scope for abnormal behavior. In this paper, we present a new technique for detecting distance-based outliers, aimed at reducing execution time associated with the detection process. Our approach operates in two phases and employs three pruning rules. In the first phase, we partition the data into clusters, and make an early estimate on the lower bound of outlier scores. Based on this lower bound, the second phase then processes relevant clusters using the traditional block nested-loop algorithm. Here two efficient pruning rules are utilized to quickly discard more non-outliers and reduce the search space. Detailed analysis of our approach shows that the additional overhead of the first phase is offset by the reduction in cost of the second phase. We also demonstrate the superiority of our approach over existing distance-based outlier detection methods by extensive empirical studies on real datasets.

1 Introduction

The problem of detecting abnormal events, also called outliers, has been widely studied in different research communities as rare classes mining [1], exception mining [2], outlier detection [3,4], etc. Researchers have developed several *supervised* and *unsupervised* techniques to mine outliers in *static databases* and also recently in *data streams* [9]. Unsupervised outlier detection can be further classified as distance-based [5,6,4,7], density-based [3,8,9] and deviation-based [10]. In this paper, we focus on distance-based outliers which have been popularly defined as: (a) data points from which there are fewer than p points that are within distance r [4], (b) top n data points whose distance to their corresponding k^{th} nearest neighbor are largest [7], and (c) top n data points whose total distance to their corresponding k nearest neighbors are largest [6]. As these definitions indicate, a significant amount of distance computations need to be performed in order to verify whether a data point is an outlier or not. This leads to high execution times and has motivated many attempts to produce efficient algorithms to mine outliers. Among them, outstanding work by Bay and Schwabacher [11] and Ghoting et al. [12] aim to reduce execution time by utilizing a simple pruning nested-loop algorithm.

Reducing time complexity of outlier detection techniques in general generates many benefits for various applications where the speed of detecting deviations

W. Buntine et al. (Eds.): ECML PKDD 2009, Part II, LNAI 5782, pp. 160–175, 2009.

plays a critical role (e.g., fraud detection, intrusion detection). To illustrate our point, let us consider a system in which data arrives in batches and each batch of data is stored in buffer memory. It may be assumed that the buffer size is large enough to accommodate each batch but if many batches are stored at the same time, buffer will overflow. Such scenario is very popular in applications dealing with data streams [13,9]. The task of the system is to identify abnormal records in each batch. The buffer is automatically flushed when this monitoring process is done. However, if the speed of the detection technique is slower than the speed of arrival of batches, we may lose data because of the problem of buffer overflows. Therefore, developing a fast detection algorithm becomes a necessity since it leads to higher throughput for the system. Additionally, the higher throughput will also yield higher detection accuracy since data loss is avoided.

Motivated by this issue, we focus on reducing the execution time and present a two-phased *MultI-Rule Outlier* (MIRO) detection approach. Based on the definition [6], we develop an outlier scoring criterion. Then in the first phase, we partition the data into clusters, and make an early estimate on the lower bound of outlier scores. This phase prunes clusters that cannot have outliers, and the second phase then processes the remaining clusters using the traditional block nested-loop algorithm. Here two pruning rules are utilized: a) first triangular inequality on the data point's outlier score is used, and then b) the outlier score is compared with the minimum score required to be an outlier. The second check is similar to that of ORCA [11]. However, while ORCA starts with a cutoff of 0, in MIRO the initial cutoff is obtained from the first phase, and hence converges faster. Though the pruning rules seem simple, their combined effect is strong and efficiently reduces the search space. The main contributions of this work can be summarized as follows:

- We analyze the problem of outlier detection from the outlier score perspective and introduce the concepts of global and local outlier score functions. This gives a summary classification of all existing detection techniques.
- We demonstrate a huge improvement in execution time by using multiple pruning rules in two phases, compared with outstanding existing nested-loop distance-based methods, ORCA [11] and RBRP [12]. Since ORCA, RBRP and MIRO use the same notion of outlier (Section 2), outliers identified by the three techniques are exactly the same.
- We illustrate the effectiveness of our pruning rules on the overall detection process and give a detailed theoretical analysis on how those rules lead to the superior performance of MIRO. With extremely low CPU cost, MIRO is very suitable for detecting outliers in streaming environments as well as other real-time applications.

The rest of this paper is organized as follows. We compare related work, and describe the problem formally in the next section. Then we present our MIRO approach in Section 3, and theoretically analyze its complexity in Section 4. Then we empirically compare our approach with other current-best approaches using real-world datasets in Section 5. Finally, we conclude in Section 6 with directions for future work.

2 Literature Review

2.1 Background

Consider a dataset DS with N data points in dim dimensions. While most of these data points are normal, some are abnormal (outlier), and our task is to mine these outliers. Assume a metric distance function D exists, using which we can measure the dissimilarity in dim space between two arbitrary data points. A general approach that has been used by most of the existing outlier detection methods [5,4,3] is to assign an *outlier score* (based on the distance function) to each individual data point, and then design the detection process based on this score. The use of the outlier score is analogous to the mapping of the multi-dimensional dataset to $\mathbb{R}$ space (the set of real numbers). In other words, we can define the outlier score function (F_{out}) which maps each data point in DS to a unique value in $\mathbb{R}$.

Among existing approaches to outlier detection problem, we can classify F_{out} into *global* and *local* score functions. An outlier score function is called *global* when the the value it assigns to a data point $p \in DS$, can be used to compare globally with other data points. More specifically, for two arbitrary data points p_1 and p_2 in DS, $F_{out}(p_1)$ and $F_{out}(p_2)$ can be compared with each other, and if $F_{out}(p_1) > F_{out}(p_2)$, p_1 has a larger possibility than p_2 to be an outlier. The definitions proposed by Angiulli et al. [6], Breunig et al. [3], and Ramaswamy et al. [7] straightforwardly adhere to this category. On the other hand, the definition of Ng and Knorr [4] can be converted to this category by taking the inverse of the number of neighbors within distance r of each data point. In contrast, a *local* outlier score function assigns to each data point p, a score that can only be used to compare within some local neighborhood. An example of such function was proposed in [8], where the local comparison space is the set of data points lying within the *circle* centered by p and the *radius* is user-defined. The choice of a global or local outlier score function clearly affects later stages of the algorithm design process. In this work, we employ a global outlier function based on [6], although the ideas employed in MIRO can also be adapted to use other functions. The intuition and quality of detection results of the chosen outlier definition are based on solid foundations as shown by prior work [6,11]. This definition is also employed in other popular techniques on outlier detection [12]. Therefore, in this paper we do not again demonstrate how well MIRO does in terms of actually discovering abnormalities in real data. Instead, we focus on showing its superiority in terms of CPU cost.

Let us denote the set of k nearest neighbors of a data point p in DS as kNN_p. We can now define F_{out} as follows.

Definition 1. [Outlier Score Function]. *The dissimilarity of a point p with respect to its k nearest neighbors is known by its cumulative neighborhood distance. This is defined as the total distance from p to its k nearest neighbors in DS. In other words, we have:* $F_{out}(p) = \sum_{m \in kNN_p} D(p, m)$.

Table 1. Definitions of symbols

Symbol	Definition
DS	The dataset
N	Number of points in the dataset
dim	Dimensionality of the data space
$D(p_1, p_2)$	Distance function between points p_1 and p_2
kNN_p	set of k nearest neighbors of a data point p
n	Number of outliers to be mined
F_{out}	Outlier Score Function

This definition has been proven by Angiulli et al. [6] to be more intuitive than the definition used by Ramaswamy et al. [7]. Given two positive integers k and n, our task is to mine top n outliers that have the largest outlier scores based on the chosen F_{out}. For ease of reference, symbols used in the definitions are presented in Table 1.

2.2 Related Work

Work in distance-based outlier detection was pioneered by Knorr and Ng in 1998 [4]. According to their proposal, outliers are points from which there are fewer than p other points within distance r. In order to detect such outliers, they introduced a nested-loop and a cell-based algorithm. The nested-loop algorithm has time complexity $O(N^2)$ and hence is usually not suitable for applications with a large dataset. On the other hand, the cell-based algorithm has time complexity linear with N, but exponential with the number of dimensions dim. In practice, this can only work efficiently when $dim \leq 4$, so it is not suitable in applications on high-dimensional datasets.

Ramaswamy et al. [7] had a different view of the problem. Instead of counting the r-neighborhood of a data point, their technique only takes the data point's distance to its k^{th} nearest neighbor into account. They proposed three algorithms: nested-loop with $O(N^2)$ time complexity, and index-based and partition-based algorithms. The most efficient among these - the partition-based algorithm, partitions the dataset, and computes the upper and lower bounds of outlier scores for each partition[1]. Keeping track of the minimum lower bound computed so far, the algorithm terminates bound computations of partitions whose upper bound score is lower than this minimum bound. This effectively reduces the search space, and then index-based or nested-loop algorithms can be used on the remaining partitions to detect outliers. In Section 4, we prove that the theoretical complexity of the partition-based strategy is also quadratic to the dataset size. In general, early distance-based approaches were usually involved in time-consuming computations of nearest neighbors. Later techniques aim to reduce this time complexity by various means. Among these, approaches for pruning the outlier search space and distance computation reduction tech-

[1] Alternatively called clusters, micro-clusters.

niques are dominant. Computation reduction approaches [7,12,11,6] usually fix the desired number of outliers to a certain value (e.g., top n outliers), and deploy data structures similar to those used in Ramaswamy's index-based algorithm.

Bay and Schwabacher [11] provide detailed analysis for this type of algorithm, and discover that, in average case, the time complexity becomes linear with the data set size. However, their proposed technique, ORCA, depends on some assumptions: the data is in random order and the values of the data points are independent. The analysis provided also depends on the outlier score cutoff c which is initialized to 0. However, domain knowledge or a training phase can help to achieve a better cutoff. More specifically, the authors suggest that by training a subset of the original data set, an initial cutoff threshold can be obtained. During testing phase, the training set is placed at the top of the data set so that the cutoff threshold calculated during training phase can be retrieved very soon, and hence the pruning occurs at the very first stage of the detection process. The linear time complexity presented in [11] can only be obtained if the cutoff threshold c converges to $O(\sqrt{N})$ quickly [12]. However this occurs only when the dataset contains many outliers. Recognizing this limitation, Ghoting et al. [12] proposes RBRP, an algorithm which finds *approximate* nearest neighbors for every normal data point but exact ones for outliers. By avoiding expensive computations to find the exact nearest neighbors for normal records, RBRP works in $O(N \cdot lgN)$ time. The approach first clusters the dataset, and then searches for a data point's approximate nearest neighbors in its own cluster and neighboring clusters.

While the above mentioned techniques attempt to reduce execution time of the detection process, Tao et al. [14] aims at reducing I/O cost without any heuristic to minimize the CPU cost. Furthermore, it uses the notion of outliers introduced in [4], which has been shown to be difficult to apply in practice [7]. Hence, we choose not to compare our technique against the one in [14].

3 The MIRO Detection Approach

Our approach operates in two phases and employs three pruning rules. In the first phase, we partition DS into clusters, and compute upper and lower bounds of the outlier score for each cluster. Based on these bounds, some clusters are pruned, and the remaining candidates are sent for final processing in the traditional block nested-loop algorithm. Here two pruning rules are utilized: a) first triangular inequality on the data point's outlier score is used (R_1), and then b) the outlier score is compared with the minimum score required to be an outlier (R_2). The second check is similar to that of ORCA [11], however in MIRO the initial cutoff is obtained from the first phase (instead of using 0 as in [11]), and hence converges faster. The additional overhead of the first phase is offset by the reduction in cost of the second phase. While preprocessing by clustering has been proposed in RBRP, our preprocessing phase incorporates the pruning of unnecessary clusters while RBRP's does not. Additionally, the use of the simple triangular inequality in the second phase and the precomputation of the initial

Algorithm 1. CLUSTER

Input: M: the number of clusters, it: the number of iterations, DS: the dataset to be clustered
Output: B: the set of clusters

```
1  Set Y = KMeans(M, it, DS)
2  foreach cluster y ∈ Y do
3      if |y|/n_c > M then
4          Cluster(M, it, y)
5      else if |y|/n_c > 1 then
6          Set Y' = KMeans(⌊|y|/n_c⌋, it, y)
7          foreach cluster y' ∈ Y' do
8              Add y' to B
9      else
10         Add y to B
```

cutoff of outlier score before this phase commences, generates the distinct advantages of MIRO's nested-loop compared to that of ORCA. The detailed process is described below.

3.1 Cluster Based Pruning

In this phase, we first cluster the dataset DS (using Algorithm 1) and subsequently identify upper and lower bounds of the outlier score for each resultant cluster (using Algorithm 2). Algorithm 1 is in fact based on the clustering algorithm of RBRP [12], however we have made some modifications. We denote the *expected number of data points per cluster* as n_c. By changing n_c, we can control the degree of homogeneity of clusters, i.e., points that are close to each other in space are likely assigned to the same cluster. It is noted that in our approach, n_c has the same role as the parameter $BinSize$ of RBRP. Compared to the original algorithm [12], the cost of clustering is saved for those resultant clusters y having $1 < |y|/nc \leq M$, since a) they are re-clustered only once with the number of clusters being $\lfloor \frac{|y|}{n_c} \rfloor \leq M$ and b) the time complexity of K-Means algorithm is proportional to the number of clusters produced. Hence, our clustering algorithm takes less time than that of RBRP.

Let C be the set of clusters obtained as a result of applying Algorithm 1 on DS with predetermined values of M and it. For each cluster $C_i \in C$, let $|C_i|$ denote its cardinality (or the number of data points allocated to C_i), o_{C_i} its centroid, and r_{C_i} its radius. l_{C_i}, u_{C_i} are the estimated lower and upper bounds of the outlier scores of all data points in C_i respectively. These bounds are only estimations since the true bounds can only be known when the true scores of member data points are identified. A data point p by itself is also a cluster C_i with $o_{C_i} = p$, $r_{C_i} = 0$, $l_{C_i} = u_{C_i} = F_{out}(p)$.

Definition 2. [Distance between clusters]
The minimum distance between clusters C_i and C_j is

$$minDis(C_i, C_j) = max\{D(o_{C_i}, o_{C_j}) - r_{C_i} - r_{C_j}, 0\},$$

and maximum distance between clusters C_i and C_j is

$$maxDis(C_i, C_j) = D(o_{C_i}, o_{C_j}) + r_{C_i} + r_{C_j}.$$

Given a cluster $C_i \in C$, we now need to find clusters that potentially contain k nearest neighbors for every point in C_i. So we first find a set of clusters, Min_{C_i}, *closest* to C_i in terms of $minDis()$, containing at least k data points, i.e., $Min_{C_i} \subseteq C \setminus C_i$, s.t. $minDis(C_j, C_i) \leq minDis(C_k, C_i)\ \forall C_j \in Min_{C_i}, C_k \in C \setminus \{C_i \bigcup Min_{C_i}\}$, the total number of data points in $Min_{C_i} \geq k$.

Similarly, we identify a set of clusters, Max_{C_i}, *closest* to C_i in terms of $maxDis()$, which also contains at least k data points in total.

Consider a data point $p \in C_i$. To compute the lower bound of its outlier score, we have to find the *closest* clusters to p in terms of $minDis()$. In order to do this we consider all clusters closest to C_i as well as other data points in C_i (as clusters). So we choose $Min_p = Min_{C_i} \bigcup C_i \setminus p$. In order to estimate the cumulative distance from p to its k nearest neighbors, we order Min_p and choose the top z clusters $M_1 \ldots M_z$ s.t. $\sum_{i=1}^{z-1} M_i < k \leq \sum_{i=1}^{z} M_i$. Now the lower bound of the outlier score of p can be computed as $l_p = \sum_{i=1}^{z-1} |M_i| \cdot minDis(p, M_i) + (k - \sum_{i=1}^{z-1} |M_i|) \cdot minDis(p, M_z)$.

Similarly we can compute the upper bound of p's outlier score, $u_p = \sum_{i=1}^{z-1} |M_i| \cdot maxDis(p, M_i) + (k - \sum_{i=1}^{z-1} |M_i|) \cdot maxDis(p, M_z)$, where $\{M_1 \ldots M_z\}$ are the top z clusters in Max_p defined as $Max_{C_i} \bigcup C_i \setminus p$.

Definition 3. [Bounds of a cluster's outlier score]. *The upper and lower bounds of a cluster's outlier score in terms of its contained points are given as: $u_{C_i} = max\{u_p, p \in C_i\}$ and $l_{C_i} = min\{l_p, p \in C_i\}$, respectively.*

We now use a simple heuristic to prune clusters that do not contain outliers: pick clusters with the largest lower bounds of outlier scores, until we have a total of at least n data points. Let the last cluster picked be C_o. Clusters whose upper bounds of outlier scores are smaller than l_{C_o} cannot contain outliers, and are therefore pruned. This heuristic constitutes the first pruning phase and is presented in Algorithm 2. The value l_{C_o} is passed as an initial seed to the second pruning phase for faster pruning. While the above heuristic correctly prunes clusters containing data points which are all non-outliers, it may allow clusters containing some non-outliers. This happens for all clusters C_i, where $l_{C_i} \leq l_{C_o} \leq u_{C_i}$. This is undesirable, since not all data points in these clusters are potential outliers. In order to resolve this issue, we propose another heuristic called P_{points} which prunes all points $p \in C_i, u_p < l_{C_o}$. Time complexity of MIRO with and without P_{points} is discussed in Section 4.1.

Algorithm 2. PRUNECLUSTERS

1 $l_{C_i}, u_{C_i} \leftarrow$ estimateBounds $\forall_i C_i \in C$
2 Identify C_o, l_{C_o}
3 Prune $C_i | u_{C_i} < l_{C_o}$
4 Return l_{C_o}, C

3.2 Nested-Loop Algorithm

After the lower bound on the outlier score is obtained from the first phase, we process the remaining clusters using the traditional nested-loop algorithm similar to ORCA [11]. In the second phase of MIRO (Algorithm 3) we employ two pruning rules (R_1 in line 9 and R_2 in line 13 of Algorithm 3). Similar to [11], we check if the outlier score of the data point is smaller than the current cutoff c on the outlier score (rule R_2). However, while ORCA initializes c as 0, in our second phase, we converge faster by choosing c from the first clustering phase (with or without P_{points}).

Let us consider an arbitrary data point q. If $c > kD(p,q) + F_{out}(q)$, then by our definition of outlier score and using triangular inequality, we can show that $c > \sum_{m \in kNN_q} D(p,m) \geq F_{out}(p)$, i.e., $c > F_{out}(p)$. Therefore p is not an outlier and can be pruned. Despite its simplicity, this pruning rule is extremely efficient in the final processing phase as shown in Section 5. By using the combination of two pruning rules, the execution time is further reduced, creating a huge advantage over ORCA and RBRP [12]. It is also noted that by reserving Min_{C_i} and Max_{C_i} for each remaining cluster C_i, we are able to limit the search space for each data point $p \in C_i$. More specifically, to process p, in the worst case we only have to scan $C_i \cup \{\bigcup_{C_1 \in Min_{C_i}} C_1\} \cup \{\bigcup_{C_2 \in Max_{C_i}} C_2\}$. The search space is therefore much smaller than the original dataset DS.

4 Theoretical Analysis

In addition to the notations stated in Table 1, we define the following new terms for analysis: (a) p_1 is the probability that a cluster will be pruned during the first phase, and (b) p_2 is the probability that a data point will be pruned by rule R_1 before it is scanned with the $(k+1)^{th}$ data point among the remaining ones. It is also noted that in practice, $n_c \leq k$ and $n \ll N$. In the following discussion, we present detailed time and space complexity analysis for MIRO.

4.1 Time Complexity of MIRO

The execution time cost of the first phase without P_{points} includes (a) the cost of clustering ($S_{cluster}$), (b) the cost of computing upper and lower bounds outlier score for all clusters (S_{bounds}), and (c) the pruning cost ($S_{pruning}$). The expected clustering cost is $O(N \cdot logN)$ according to [12]. Now, for a cluster C_i, we need to identify Min_{C_i} and Max_{C_i}. Since the mean size of each cluster is n_c, on average

Algorithm 3. FinalProcessing

```
1  Set c, C ← PruneClusters()
2  Set TopOut ← ∅
3  foreach remaining cluster C_i ∈ C do
4      Set A ← C_i ∪ {⋃_{C_1 ∈ Min_{C_i}} C_1} ∪ {⋃_{C_2 ∈ Max_{C_i}} C_2}
5      foreach data point p ∈ C_i do
6          foreach cluster C_j ∈ A do
7              foreach data point q ∈ C_j do
8                  if q ≠ p then
9                      if (c − F_out(q))/k > D(p, q) then
10                         Mark p as non-outlier
11                         Process next data point in C_i
12                     Update p's k nearest neighbors using q
13                     if F_out(p) < c then
14                         Mark p as non-outlier
15                         Process next data point in C_i
16         if p is outlier then
17             Update TopOut with p
18             if Min(TopOut) > c then
19                 Set c ← Min(TopOut)
```

we have $|Min_{C_i}| = |Max_{C_i}| = \lceil k/n_c \rceil$. A naïve approach sorts all clusters and extracts $\lceil k/n_c \rceil$ clusters for Min_{C_i}/Max_{C_i}, at a cost of $O(\frac{N}{n_c} \cdot log(\frac{N}{n_c}))$. However, we note that only $\lceil k/n_c \rceil$ clusters need to be reserved for Min_{C_i} as well as Max_{C_i}. Therefore a better approach is that for each cluster C_j, we compute the minimum/maximum distance from C_j to C_i and insert the result into the corresponding set. This approach leads to an total cost of $O(\frac{1}{2} \cdot \lceil \frac{k}{n_c} \rceil \cdot \frac{N}{n_c} \cdot (\frac{N}{n_c} - 1))$ over all clusters, which can be simplified to $O(\frac{N^2}{{n_c}^2})$. To estimate the cost of computing upper and lower bounds of the outlier score for each cluster C_i, we compute the cost of measuring the same bounds for each individual data point $p \in C_i$. To obtain p's bounds, we also need to extract $n_c + \lceil \frac{k}{n_c} - 1 \rceil$ clusters (including zero-radius ones) from a set of $n_c + \lceil \frac{k}{n_c} \rceil$ clusters. Since the number of items extracted is nearly no different from the total set of items, we apply the naïve sorting approach discussed above. As a consequence, the total cost incurred is $O((n_c + \lceil \frac{k}{n_c} \rceil) \cdot log(n_c + \lceil \frac{k}{n_c} \rceil))$, i.e., $O(n_c \cdot log(n_c))$. Hence, the cost of computing C_i's bounds $= O({n_c}^2 \cdot log(n_c))$. Therefore, $S_{bounds} = O(\frac{N}{n_c} \cdot {n_c}^2 \cdot log(n_c)) + O(\frac{N^2}{{n_c}^2}) = O(N \cdot n_c \cdot log(n_c)) + O(\frac{N^2}{{n_c}^2})$. To prune the clusters, we need to compute l_{C_o} and scan the whole set of clusters to check their corresponding upper bounds. To compute l_{C_o}, we need to extract $\lceil n/n_c \rceil$ clusters with largest lower bounds from a set of N/n_c clusters. In other words, $S_{pruning} = O(\lceil \frac{n}{n_c} \rceil \cdot \frac{N}{n_c}) + O(\frac{N}{n_c})$. Overall, the approximate overhead incurred by the first phase is:

$S_{phase1} = S_{cluster} + S_{bounds} + S_{pruning} = O(N \cdot logN) + O(N \cdot n_c \cdot log(n_c)) + O(\frac{N^2}{{n_c}^2}) + O(\lceil \frac{n}{n_c} \rceil \cdot \frac{N}{n_c}) + O(\frac{N}{n_c}) = O(N \cdot logN) + O(N \cdot n_c \cdot log(n_c)) + O(\frac{N^2}{{n_c}^2}) + O((\lceil \frac{n}{n_c} \rceil + 1) \cdot \frac{N}{n_c})$.

After the first phase, the number of remaining clusters is $(1 - p_1) \cdot \frac{N}{n_c}$, which implies that the total number of remaining data points is $n_c \cdot (1 - p_1) \cdot \frac{N}{n_c} = (1 - p_1) \cdot N$. Among them, the total number of data points pruned out by the rule R_1 with no more than k distance computations is $p_2 \cdot (1 - p_1) \cdot N$. On the other hand, for each of the data points left, we need to scan the entire cluster C_i as well as Min_{C_i} and Max_{C_i} in the worst case, i.e., the corresponding cost is $O(n_c + 2 \cdot n_c \cdot \lceil k/n_c \rceil)$, which simplifies to $O(3 \cdot n_c + 2 \cdot k)$. Hence the execution time of the second phase in the worst case can be expressed as:

$S_{phase2} = O(k \cdot p_2 \cdot (1 - p_1) \cdot N + (3 \cdot n_c + 2 \cdot k) \cdot (1 - p_2) \cdot (1 - p_1) \cdot N) = O((3 \cdot n_c \cdot (1 - p_2) + k \cdot (2 - p_2)) \cdot (1 - p_1) \cdot N)$.

Hence, the approximate cost of the whole algorithm is:

$S_{phase1} + S_{phase2} = O(N \cdot logN) + O(N \cdot n_c \cdot log(n_c)) + O(\frac{N^2}{{n_c}^2}) + O((\lceil \frac{n}{n_c} \rceil + 1) \cdot \frac{N}{n_c}) + O((3 \cdot n_c \cdot (1 - p_2) + k \cdot (2 - p_2)) \cdot (1 - p_1) \cdot N)$.

We can also reclassify the whole detection process into a more detailed sequence of operations: (a) clustering, (b) identifying neighboring clusters for all clusters, (c) computing the bounds for clusters (we consider the process for each cluster as a operation, so we have N/n_c operations), (d) pruning clusters (N/n_c operations on average) and (e) final processing step ($(1 - p_1) \cdot N$ operations on average). Among them, the cost of the operations (a) and (b) are loglinear and quadratic w.r.t. N, respectively. On the other hand, each of the remaining operations incurs costs independent of N. Furthermore, when p_1 has large values, the execution time of the second phase becomes very small which compensates the overhead incurred by the first phase. In addition, when p_2 receives a large value, a larger portion of the remaining data points after the first phase require no more than k distance computations to be identified as normal records, and a smaller number of these remaining points require more than k distance computations. This fact leads to another reduction of execution time. Besides, the pre-computation of cutoff c helps contribute to further reduction of the execution time. Therefore, practically each of the operations performed in item (e) takes nearly constant time. By applying the accounting method of amortized analysis, we expect the expensive cost of operations (a) and (b) would be compensated by the remaining inexpensive ones, i.e., the *amortized running time* of each individual operation is inexpensive and non-quadratic w.r.t. N. In the experiments carried out in Section 5, we always have $\max(p_1, p_2) \geq 0.7$ which leads to the practical linear execution time w.r.t N. It is also noted that based on our analysis, this quadratic overhead w.r.t. N is common for techniques that utilize similar partition-based strategy such as [7], which though using less pruning rules than MIRO, still reports linear execution time performance w.r.t N.

Time complexity with P_{points}. In the above analysis, we assume that the P_{points} heuristic (c.f., Section 3.1) is not used for the first phase. In contrast, if this heuristic is considered, we prune all points whose upper bound of outlier score is less than the cutoff obtained by the clustering phase, so $S_{pruning}$ has to be recomputed. Particularly, after applying l_{C_o} for pruning out clusters, we perform an additional scan on the set of clusters left. The mean number of clusters to scan is therefore $(1 - p_1) \cdot \frac{N}{n_c}$, and the expected cost for scanning each cluster is n_c. Consequently, the additional cost is $O((1 - p_1) \cdot \frac{N}{n_c} \cdot n_c) = O((1 - p_1) \cdot N)$.

From the above analysis, it can be seen that the cost of S_{phase1} does not change theoretically whether P_{points} is used or not. But P_{points} is only effective if it does indeed help to prune out more data points after the first phase. We will examine that in Section 5.

4.2 Space Complexity of MIRO

As mentioned earlier, minimizing I/O cost is neither a focus of techniques in [11,12,6] nor of MIRO. Hence, in general MIRO uses space for: (a) storing the data points, and (b) storing the clusters created. Furthermore, the spatial cost for storing each cluster C_i can be simplified to the cost of storing its major components which include: (a) its member data points, and (b) Min_{C_i} as well as Max_{C_i}. This is simplified by space-efficient hash indexes, therefore each C_i takes $O(n_c + 2 \cdot \lceil \frac{k}{n_c} \rceil)$ space on average. Hence, the space complexity of MIRO is $O(N) + O(\frac{N}{n_c} \cdot (n_c + 2 \cdot \lceil \frac{k}{n_c} \rceil))$, which can be simplified to $O(N)$.

4.3 Analysis of Parameters Used

Cluster size. For a fixed dataset size, as the average cluster size n_c decreases, the total number of clusters will increase. Since the size of each cluster C_i becomes smaller, in order to compute the bounds of C_i, we need to include more clusters in Min_{C_i} as well as Max_{C_i}. In other words, more clusters are required for computing C_i's bounds. That increases S_{bounds} and leads to the increase in the overall execution time of our algorithm. In the extreme case, when $n_c = 1$, the first phase degrades to scanning the entire dataset, i.e., the total execution time becomes a normal nested-loop algorithm and the execution time saved during the second phase becomes insufficient to compensate this overhead. On the other hand, as n_c increases, there are less clusters than before. Since the size of each cluster becomes larger, we need to consider fewer clusters in the process of computing clusters' bounds on the outlier score. But that does not directly lead to a decrease in cost of computing bounds since we need to process more data points per cluster. Furthermore, as n_c increases and exceeds k, the lower bound score l_{C_o} becomes smaller since we only need to use data points in a cluster C_i to compute its bounds (the assumption here is that in general a cluster contains data that are relatively homogeneous). That means less clusters are pruned after the first phase hence the execution time will increase. Overall, we should choose a reasonable value of n_c such that the average number of data points per cluster is neither too small nor too large compared to k. More specifically, we need to

identify a threshold for n_c such that as n_c increases above as well as decreases below this threshold, the execution time of MIRO will increase. Consequently, picking this threshold to be n_c will be a wise choice. From the above analysis, we conclude that the impact of n_c over the overall performance of MIRO is complex and identification of reasonable values for n_c by analytical methods is practically infeasible. Through empirical study carried out in Section 5, we show that $k/5$ is a possible candidate value.

Number of nearest neighbors. As the number of nearest neighbors taken into account for the computation of outlier score, k, increases, the value that F_{out} assigns to each individual data point p in DS will increase correspondingly. This in turn leads to an increase in the lower bound l_{C_o}, and hence more clusters may be pruned by the first phase of MIRO. However, as demonstrated before, an increase of k results in having to consider more clusters when computing outlier score bounds for an arbitrary cluster. Therefore, the cost of computing cluster's bounds will increase. The increase of k creates a two-fold effect: (a) a decrease in execution time since more data points are pruned, and (b) an increase in execution time due to the increase in the cost of computing clusters' bounds. Our experimental result in Section 5 shows that MIRO's execution time increases as k increases, i.e., the latter factor outperforms the former one.

5 Empirical Results and Analyses

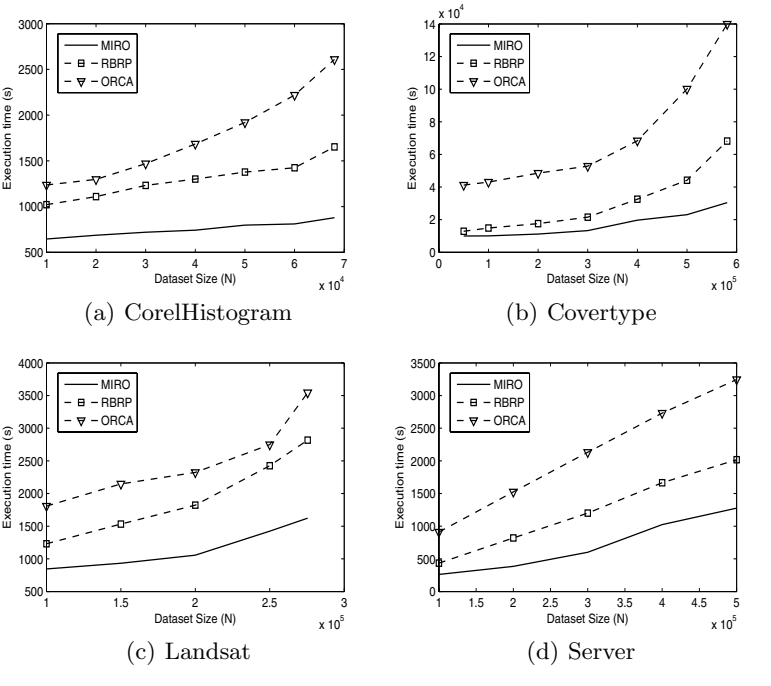
In order to assess the effectiveness of our proposed technique, we performed extensive experiments on four real and high-dimensional datasets CorelHistogram, Covertype, Server[2] and Landsat[3]. All of these are original datasets except for Server which is extracted from KDD Cup 1999 data, using the procedure provided in [14]. For each set of input parameters that affect the performance of the corresponding algorithm, we ran the experiment ten times. The results presented are from average outcomes obtained from multiple runs. It is noted that we set $M = 10$ and $it = 5$ throughout all experiments. Through the empirical studies, we demonstrate:

- The efficiency of MIRO in reducing the execution time of the traditional nested-loop algorithm. We measure the scalability of MIRO's execution time against the dataset size (N) as well as the number of nearest neighbors (k) used. In the latter case, we present MIRO's performance with and without P_{points}. The result is then compared with ORCA [11] and RBRP [12] to highlight the merit of our method.
- The pruning power of MIRO, in both phases of processing, with and without P_{points}. In addition, we also assess the effect of k on the pruning quality. The sensitivity of MIRO's execution time with respect to the cluster size (n_c) is also presented.

[2] `http://www.ics.uci.edu/~mlearn/MLRepository.html`
[3] `http://vision.ece.ucsb.edu`

Execution time v/s. N: First we evaluate the scalability of execution time of three distance-based outlier detection techniques MIRO, RBRP and ORCA w.r.t the dataset size N. In this experiment, we chose the number of outliers mined $n = 30$, number of nearest neighbors $k = 50$, set the size of each cluster $n_c = 20$, and varied N. We chose the implementation of MIRO without P_{points} since the efficiency of P_{points} is highlighted in a later part of this section. We observe from the result (Figure 1) that MIRO scales better than RBRP and ORCA on all datasets, although its theoretical asymptotic time complexity is quadratic in N. This agrees with the amortized analysis in Section 4.1. In order to analyze the cause of MIRO's efficiency, we also compare the execution time with and without the first phase.

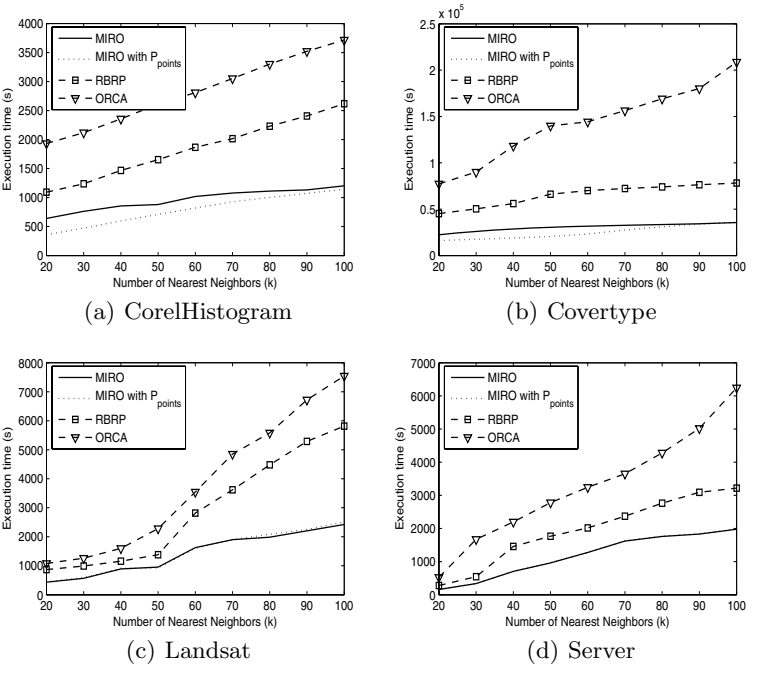
Execution time and MIRO's pruning power v/s. k: We now analyze the effect of the number of nearest neighbors (k) on execution time. This experiment is conducted on the entire datasets, and $n = 30$, $n_c = 20$ as in the previous case. The results (Figure 2) show that the execution time for every technique increases with k, but MIRO scales better (with and without P_{points}) compared to RBRP and ORCA. The reason is once again attributed to the effective pruning power of MIRO in both phases of processing. It is also clear that by using P_{points}, we are able to obtain better or equal performance in term of execution time. This observation is further analyzed later when we discuss the effect of k on MIRO's pruning power.



(a) CorelHistogram (b) Covertype

(c) Landsat (d) Server

Fig. 1. Execution time vs. the dataset size N

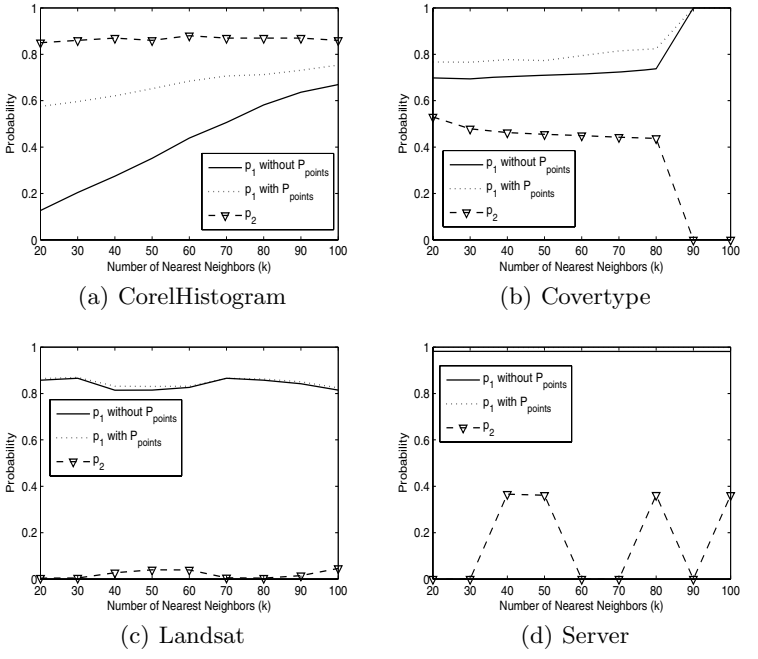


(a) CorelHistogram (b) Covertype

(c) Landsat (d) Server

Fig. 2. Execution time vs. the number of nearest neighbors k

Figure 3 presents two pruning probabilities in one plot for each dataset: the probability of pruning a cluster in the first phase (p_1), and the probability that a data point will be pruned out by rule R_1 before it is scanned with the $(k+1)^{th}$ data point among the remaining ones (p_2), as the number of nearest neighbors is varied. In all cases, very high values of p_1 and/or p_2 are achieved, with p_1 increasing when P_{points} is utilized. While we do not obtain high values for both p_1 and p_2 at the same time, we observe that in every case at least one of them receives a value greater than 0.7. This reflects a very high efficiency in pruning and explains why MIRO takes less execution time compared to RBRP and ORCA. In addition, the value of p_1 tends to increase as k increases (except in the case of Landsat dataset), which means more clusters will be pruned after the first phase when k receives higher value. This agrees with the discussion in Section 4.3. Furthermore, when p_1 without P_{points} already has relatively large value, applying P_{points} does not help much in increasing the pruning power of the first phase. This point is reflected by the tendency of p_1 with and without P_{points} to converge towards each other as p_1 increases. We also observe that when the pruning effect without using P_{points} is low, i.e., when p_1 is low, there will be a significant improvement in execution time if P_{points} is employed instead. This can be attributed to the fact that adjoining clusters' lower and upper outlier score bounds are too interleaved with each other which creates redundancy if we include the whole of each candidate cluster in the final processing step. In



(a) CorelHistogram (b) Covertype

(c) Landsat (d) Server

Fig. 3. MIRO's pruning power vs. the number of nearest neighbors k

contrast, if the value of p_1 is already high, which means l_{C_o} has been identified wisely, using P_{points} may not improve MIRO's performance by much, although the pruning effect obtained is still equal or better. The reason is that increase in pruning power in such cases is not enough to compensate the additional time spent to run P_{points}. However, it is noted that when p_1 receives a higher value, the cost of executing P_{points}, which is $O((1 - p_1) \cdot N)$, becomes lower. Therefore, it can be concluded that applying P_{points} does not degrade performance by much, but may lead to significantly better performance.

Execution time v/s. n_c: For studying the effect of the average cluster size (n_c) on the execution time of MIRO, we set $n = 30$ while varying k. For each value of k, we run MIRO with $n_c \geq 1$ and $\leq k$ and note the value of n_c which yields smallest CPU cost. The result obtained suggests that n_c should be $k/5$. A good selection of n_c helps to balance the tradeoff between the time spent on computing clusters' bounds, as well as the pruning effect of the first phase of MIRO. In practice, we can also determine n_c by performing a training process on a subset of the original dataset with $n_c = k/5$ as the initial seed.

6 Conclusions

This work contributes to outlier detection research by proposing a new combination of several pruning strategies to produce an efficient distance-based outlier

detection technique. The proposed technique, MIRO, consists of two pruning phases of processing which lead to amortized efficiency. During the first phase, a partition-based technique is employed to extract candidate clusters for the later processing step. Furthermore, an additional benefit of the first phase is that we are able to compute an initial value of the outlier cutoff threshold which is utilized in the nested-loop phase. In the second phase of MIRO, two pruning rules are employed to further reduce the overall temporal cost. In future work, we are considering to extend our analysis on more large and high-dimensional datasets to better study the full benefits of MIRO. We are also examining the possibility of applying the partition-based strategy to outlier detection problems where a *local* outlier score function is utilized. This will help us in building a general framework for creating faster detection techniques regardless of whether a *local* or *global* score function is employed.

References

1. Joshi, M.V., Agarwal, R.C., Kumar, V.: Mining needle in a haystack: Classifying rare classes via two-phase rule induction. In: SIGMOD Conference, pp. 91–102 (2001)
2. Suzuki, E., Zytkow, J.M.: Unified algorithm for undirected discovery of exception rules. In: Zighed, D.A., Komorowski, J., Żytkow, J.M. (eds.) PKDD 2000. LNCS (LNAI), vol. 1910, pp. 169–180. Springer, Heidelberg (2000)
3. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying density-based local outliers. In: SIGMOD Conference, pp. 93–104 (2000)
4. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: VLDB, pp. 392–403 (1998)
5. Aggarwal, C.C., Yu, P.S.: An effective and efficient algorithm for high-dimensional outlier detection. VLDB Journal 14(2), 211–221 (2005)
6. Angiulli, F., Pizzuti, C.: Outlier mining in large high-dimensional data sets. IEEE Transactions on Knowledge and Data Engineering 17(2), 203–215 (2005)
7. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: SIGMOD Conference, pp. 427–438 (2000)
8. Papadimitriou, S., Kitagawa, H., Gibbons, P.B., Faloutsos, C.: LOCI: Fast outlier detection using the local correlation integral. In: ICDE, pp. 315–324 (2003)
9. Nguyen, H.V., Vivekanand, G., Praneeth, N.: Online Outlier Detection Based on Relative Neighbourhood Dissimilarity. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 50–61. Springer, Heidelberg (2008)
10. Arning, A., Agrawal, R., Raghavan, P.: A linear method for deviation detection in large databases. In: KDD, pp. 164–169 (1996)
11. Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: KDD, pp. 29–38 (2003)
12. Ghoting, A., Parthasarathy, S., Otey, M.E.: Fast mining of distance-based outliers in high dimensional datasets. In: SDM (2006)
13. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Transactions on Knowledge and Data Engineering 15(3), 515–528 (2003)
14. Tao, Y., Xiao, X., Zhou, S.: Mining distance-based outliers from large databases in any metric space. In: KDD, pp. 394–403 (2006)