Fast Distributed Outlier Detection in Mixed-Attribute Data Sets

Matthew Eric Otey, Amol Ghoting and Srinivasan Parthasarathy*

Department of Computer Science and Engineering

The Ohio State University

Columbus, Ohio 43210

{otey,ghoting,srini}@cse.ohio-state.edu

June 22, 2005

Abstract

Efficiently detecting outliers or anomalies is an important problem in many areas of science, medicine and information technology. Applications range from data cleaning to clinical diagnosis, from detecting anomalous defects in materials to fraud and intrusion detection. Over the past decade, researchers in data mining and statistics have addressed the problem of outlier detection using both parametric and non-parametric approaches in a centralized setting. However, there are several challenges that must still be addressed. First, most approaches to date have focused on detecting outliers in a continuous attribute space. However, almost all real-world data sets contain a mixture of categorical and continuous attributes. The categorical attributes are typically ignored or incorrectly modeled by existing approaches, resulting in a significant loss of information. Second, there have not been any general-purpose distributed outlier detection algorithms. Most distributed detection algorithms are designed with a specific domain (e.g. sensor networks) in mind. Third, the data sets being analyzed may be streaming or otherwise dynamic in nature. Such data sets are prone to concept drift, and models of the data must be dynamic as well. To address these challenges, we present a tunable algorithm for distributed outlier detection in mixed-attribute data sets.

*This work is supported by an NSF CAREER grant (IIS-0347662) and an NSF NGS grant (CNS-0406386)

1 Introduction

A common problem in data mining is that of automatically finding outliers or anomalies in a database. Outliers are those points in a data set that are highly unlikely to occur given a model of the data. Since outliers and anomalies are highly unlikely, they can be can be indicative of bad data or malicious behavior. Examples of bad data include skewed data values resulting from measurement error, or erroneous values resulting from data entry mistakes. An example of data indicating malicious behavior is anomalous transactions in a credit card database, which may be symptoms of someone using a stolen card or engaging in other fraudulent behavior (Bolton & Hand, 2002). In the field of network traffic analysis, anomalous IP packets may indicate either a possible intrusion or attack, or a failure in the network (Sequeira & Zaki, 2002). In the domain of patient medical data, outliers may arise when the patient is afflicted with some disease, or suffers side-effects from a drug (Penny & Jolliffe, 2001). Efficient detection of such outliers reduces the risk of making poor decisions based on erroneous data, and aids in identifying, preventing, and repairing the effects of malicious or faulty behavior.

Additionally, many data mining and machine learning algorithms, and techniques for statistical analysis may not work well in the presence of outliers. Outliers may introduce skew or complexity into models of the data, making it difficult, if not impossible, to fit an accurate model to the data in a computationally feasible manner. For example, statistical measures of the data may be skewed because of erroneous values, or the noise of the outliers may obscure the truly valuable information residing in the data set. Accurate and efficient removal of outliers may greatly enhance the performance of statistical and data mining algorithms and techniques (Gamberger et al., 1999). Detecting and eliminating such outliers as a pre-processing step for other techniques is known as *data cleaning*. As can be seen, different domains have different reasons for discovering outliers: They may be noise that we want to remove, since they obscure the true patterns we wish to discover, or they may be the very things in the data that we wish to discover. As has been said before, "one person's noise is another person's signal" (Knorr & Ng, 1998).

Effective outlier detection requires the construction of a model that accurately represents the data. Over the years, a large number of techniques have been developed for building such models for outlier and anomaly detection. However, real-world data sets and environments present a range of difficulties that limit the effectiveness of these techniques. Among these problems is the fact that these data sets may contain a heterogeneous mixture of attribute types (i.e. continuous and categorical attributes), the fact that the

data may be distributed over various sites, and the fact that the data sets involved may be dynamic.

The first problem is that the attributes in a data set may be a heterogeneous mixture of categorical (nominal) and continuous types. Categorical or nominal attributes may take on a relatively small number of values, and these values have only a partial ordering, if any at all. Continuous attributes have values that are usually numeric and have a total ordering. Many techniques for outlier detection assume that the attributes in the data set are either all continuous or all categorical. However, many real-life data sets contain a mixture of types. For example, a data point representing a network flow may contain continuous types (the number of bytes transferred between two hosts, the length of the connection in seconds, etc.) and categorical types (the service accessed, the network protocol used, etc.). Having different attribute types in a data set makes it difficult to find relations between two attributes (for example, the correlation between the attributes) and to define distance or similarity metrics for such data points (for example, what is the distance between the TCP and UDP protocols?). When processing data sets with a mixture of attribute types, many techniques homogenize the attributes by converting the continuous attributes into categorical attributes by discretization (quantization), or converting categorical attributes into continuous attributes by applying some (arbitrary) ordering, which can lead to a loss in information and an increase in noise. A better outlier detection system would need to develop meaningful and useful distance metrics in mixed-type attribute spaces and measure the dependencies between attributes of different types. In this paper we present an anomaly score function that takes into account the dependencies between continuous and categorical attributes.

The second problem is that of distributed data sets, which arise in a wide variety of domains. For example, in the domain of network intrusion detection, there may be multiple edge routers collecting disjoint information on the status of the network. In the domain of commercial transaction analysis (market basket analysis), the branches of a department store chain may collect data on each purchase made at their respective stores. In order to build a global model of the items purchased, information from each store's data set must be combined in some way. A naïve approach would be to transfer all the data to a single site and apply some outlier detection technique there. However, this might not be possible if the resulting data set would be too large, or it is too expensive in terms of network resources to transfer the data to a single site. Furthermore, to achieve good scalability, a distributed or parallel algorithm must minimize the amount of synchronization performed between nodes. In this paper we present an outlier detection technique that allows each node to work asynchronously to build a local model of their respective subsets of the data, and requires only a single round of communication to combine these local models into a global model.

A third problem is that of dynamic or evolving data sets. Dynamic data sets are those where data is constantly being added, removed, or otherwise updated. This situation occurs quite often: network data streams are, by definition, dynamic, and commercial transaction databases are constantly being augmented by new purchases. In many cases, the processes generating this data may be non-stationary, meaning that a model of the data built at one point in time may become invalid in the future. In network intrusion detection, the models of network traffic and intrusion patterns are in a state of constant change, as new network protocols, virii, and intrusion techniques are constantly being developed and modified. The models representing the data in commercial transaction databases may also be in flux, since the purchases may be dependent upon the time of year (a phenomenon known as seasonality), and the demographics of the population of purchasers (since people may be moving in to or out of a region). This problem is also known as *concept drift*. Any robust abnormality detection technique must be able to handle these types of evolving models. In this paper we extend our basic outlier detection algorithm to handle dynamic data sets and concept drift.

In this paper, we present work addressing the above challenges. Specifically, we make the following contributions:

- We define an anomaly score wherein one can effectively identify outliers in a mixed attribute space by considering the dependencies between attributes of different types.
- We present a two-pass distributed algorithm for outlier detection based on this anomaly score. We also present an approximate scheme by which this algorithm allows for more efficient outlier detection when memory and processor resources are an issue.
- We extend our technique to handle dynamic data sets, where only a single pass can be made, and concept drift is a concern.
- We extensively evaluate our algorithms on several real and synthetic data sets and confirm the utility of finding outliers in mixed attribute data. We also present results on the scalability of our approach.

The remainder of this paper is organized as follows. In Section 2 we discuss previous approaches to the problem of outlier detection. In Section 3 we present our anomaly score and develop an approach that uses it to detect outliers. We present experimental results in Section 4. Finally, we conclude with directions for future work in Section 5.

2 Related Work

There are several approaches to outlier detection. One approach is that of statistical model-based outlier detection, where the data is assumed to follow a parametric (typically univariate) distribution (Barnett & Lewis, 1994). Such approaches do not work well in even moderately high-dimensional (multivariate) spaces, and finding the right model is often a difficult task in its own right. Simplified probabilistic models suffer from a high false positive rate (Mahoney & Chan, 2002; Otey et al., 2002). Also, methods based on computational geometry (Johnson et al., 1998) do not scale well as the number of dimensions increase. To overcome these limitations, researchers have turned to various non-parametric approaches including distance-based approaches (Bay & Schwabacher, 2003; Knorr et al., 2000), clustering-based approaches (Guha et al., 2000; Sequeira & Zaki, 2002), and density-based approaches (Breunig et al., 2000; Papadimitriou et al., 2003). Here we consider these methods in more detail.

An approach for discovering outliers using distance metrics was first presented in (Knorr et al., 2000). They define a point to be a *distance outlier* if at least a user-defined fraction of the points in the data set are further away than some user-defined minimum distance from that point. They primarily focus on data sets containing only real-valued attributes in their experiments. Related to distance-based methods are methods that cluster data and find outliers as part of the process of clustering (Jain & Dubes, 1988). Points that do not cluster well are labeled as outliers. This is the approach used by the ADMIT intrusion detection system (Sequeira & Zaki, 2002). A clear limitation of clustering-based approaches to outlier detection is that they require multiple passes to process the data set.

Recently, density-based approaches to outlier detection have been proposed (Breunig et al., 2000). In this approach, a local outlier factor (LOF) is computed for each point. The LOF of a point is based on the ratios of the local density of the area around the point and the local densities of its neighbors. The size of a neighborhood of a point is determined by the area containing a user-supplied minimum number of points (MinPts). A similar technique called LOCI (Local Correlation Integral) is presented in (Papadimitriou et al., 2003). LOCI addresses the difficulty of choosing values for MinPts in the LOF technique by using statistical values derived from the data itself.

Most methods for detecting outliers take time that is at least quadratic in the number of points in the data set, which may be unacceptable if the data set is very large or dynamic. Recently, a method for discovering outliers in near linear time has been presented (Bay & Schwabacher, 2003). This is a distance-based

approach to outlier detection that randomizes the data set for efficient pruning of the search space. In the worst case, the algorithm still takes quadratic time, but the author reports that the algorithm runs very close to linear time in practice.

An approach (Palpanas et al., 2003) has been presented for distributed deviation detection in sensor networks. This approach is tailored to the sensor network domain and targets misbehaving sensors. EMERALD is an approach for collaborative intrusion detection for large networks within an enterprise (Porras & Neumann, 1997). This approach allows for distributed protection of the network through a hierarchy of surveillance systems that analyze network data at the service, domain, and enterprise-wide levels. There has also been work examining techniques that allow different organizations to collaborate for enhanced network intrusion detection (Locasto et al., 2004). If organizations can collaborate, then each can build a better model of global network activity, and more precise models of attacks (since they have more data from which to estimate the model parameters). This allows for better characterization and prediction of attacks. There have also been several approaches for distributed and collaborative detection of network intrusions in wireless environments (Zhang & Lee, 2000; Zhang et al., 2003; Huang & Lee, 2003). However, these approaches focus on network intrusion detection, and are not suitable for general-purpose outlier detection.

A comparison of various anomaly detection schemes is presented in (Lazarevic et al., 2003). Its focus is on how well different schemes perform with respect to detecting network intrusions. The authors use the 1998 DARPA network connection data set to perform their evaluation, which is the basis of the KDDCup 1999 data set (Hettich & Bay, 1999) used in our experiments. They found detection rates ranging from a low of 52.63% for an approach based on the Mahalanobis distance to a high of 84.2% for an approach using support vector machines.

3 Algorithms

Our work is motivated by the following questions: First, *how does one find outliers in mixed attribute data sets*? Existing outlier detection algorithms do not handle mixed attribute data sets effectively. Our anomaly score function allows us to capture a) dependencies between categorical attributes, b) dependencies between continuous attributes, and c) dependencies between categorical and continuous attributes. As we will see, these dependencies capture information that is crucial for effective outlier detection.

Second, *how does one find outliers in a distributed setting?* Existing distance-based outlier detection

techniques flag a data point as an outlier based on its distance from the other points in the data set. In a distributed setting, the sites must either replicate the entire data set locally, or engage in voluminous queries to calculate the distances between data points residing on different sites. However, this is not feasible due to the excessive storage and communication requirements. Ideally, each site would find outliers locally while minimizing communication with other sites. This is the basis of our approach and is facilitated by an anomaly score function that is based on a global model of the data that is easily constructed by combining local models built independently at each site. The only communication that need take place between the sites is a single round of communication to build the global model from the local models.

Third, *how does one find outliers in dynamic data sets*? Most existing approaches for outlier detection need to make multiple passes over the data set since they need to capture the distances between every pair of points in the data set. To facilitate online discovery, one needs to be able to summarize essential parts of data set incrementally. We show that we can also process the data set in one pass, using an anomaly score function that facilitates single-pass processing in a distributed setting.

Our distributed approach to outlier detection is a direct result of our method for computing anomaly scores for each point in a data set. It builds on our previous work with the LOADED algorithm (Ghoting et al., 2004) by adding support for distributed processing and altering the anomaly score function to better handle continuous attributes. We first present our anomaly score that takes into account the dependencies between the categorical and continuous attributes. We then present the distributed approach, as well as techniques for reducing memory and computing requirements. Finally, we show how to extend our approach to handle dynamic data sets.

3.1 Computing Anomaly Scores

Our anomaly score function is computed from a data model containing two basic parts that take into account the categorical and continuous attribute spaces respectively. Our approach is based on the idea of *links*, which take into account the dependencies between the attributes in the data set. Outliers are then the points that violate these dependencies. We first consider links in categorical attribute space, and then consider them in a mixed attribute space.

3.1.1 Links in a Categorical Attribute Space

Informally, two data points P_i and P_j are considered *linked* if they are considerably similar to each other. Moreover, each link has a strength, which captures the degree of similarity, and is determined using a similarity metric defined on the two points. Formally, we consider a data set M in which each data point (record) P_i has N_c categorical attributes. We represent P_i as an ordered set

$$P_i =< (attribute_1, value_1), \ldots, (attribute_{N_c}, value_{N_c}) >$$

consisting of N attribute-value pairs. Specifically, we define two data points P_i and P_j to be linked in a categorical attribute space if they have at least one attribute-value pair in common. The associated link strength is equal to the number of attribute-value pairs shared between the two points.

Using these definitions, we define an outlier in a categorical attribute space as a data point that has either (a) very few links to other points or (b) very weak links to other points. A score function that generates high scores for outliers would assign scores to a point that are inversely proportional to the sum of the strengths of all its links. To estimate this score efficiently, we rely on ideas from frequent itemset mining (Agrawal & Srikant, 1994). Let I be the set of all possible attribute-value pairs in the data set M. Let

$$D = \{d \mid d \in PowerSet\,(I) \wedge \forall_{i,j:i\neq j} d_i.attribute \neq d_j.attribute\}$$

be the set of all itemsets (or groups of attribute-value pairs), where an attribute only occurs once per itemset. We define $sup(d)$ as the number of points P_i in the data set where $d \subseteq P_i$, otherwise known as the *support* of itemset d. We also define $|d|$ as the number of attribute-value pairs in d.

We can then define an anomaly score function as:

$$Score_1\,(P) = \sum_{d \subseteq P} \left(\frac{1}{|d|} \mid sup\,(d) \leq s \right) \tag{1}$$

where the user-defined threshold s is the minimum support or, using our link terminology, the minimum number of links. The score is proportional to the number of *infrequent* itemsets, and inversely proportional to the size of those infrequent itemsets (i.e. the link strength). The score function has the following characteristics:

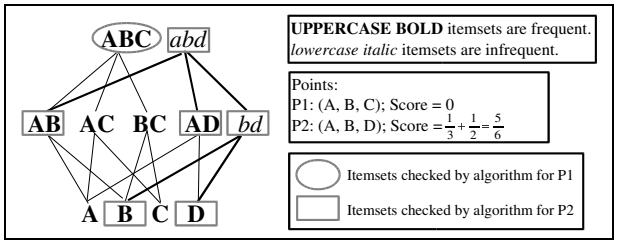

Figure 1: Lattice management

- A point with no links to other points will have the highest possible score.
- A point that has only a few links, each with a low link strength, will have a high score.
- A point that has only a few links, some with a high link strength, will have a moderately high score.
- A point that has several links, but each with a low link strength, will have a moderately high score.
- Every other point will have a low to moderate score.

We note that the function $sup(d)$ has the downward closure property, otherwise known as the *apriori* property.

Lemma 1 *If* $sup\,(d) > s$ *then* $sup\,(d') \geq s\ \forall d' \subseteq d$.

3.1.2 Example

The score estimation process is illustrated as follows: Consider the two points $P1$ and $P2$ as shown in Figure 1 together with their itemset lattice. Note that Figure 1 only shows the subset of the overall itemset lattice corresponding to the points $P1$ and $P2$. For point $P1$, all of its subset itemsets are frequent, and thus using the function $Score_1$, its score would be zero (so it is not an outlier). On the other hand, for point $P2$, the subset itemsets abd and bd are not frequent. Since abd has size 3 and bd has size 2, the resulting score is $\frac{1}{3} + \frac{1}{2} = \frac{5}{6}$.

3.1.3 Links in a Mixed Attribute Space

Thus far, we have presented an approach to estimate a score for a point in a categorical attribute space. As motivated previously, we would like to capture the dependencies between the continuous and categorical attributes in a domain-independent manner. If the expected dependencies between categorical and continuous data are violated by a point, then it is most likely an outlier. We choose a unified approach to capture

this dependence. We incrementally maintain the covariance matrix (Rice, 1995) to capture the dependencies between the continuous attributes. We then maintain this matrix for each itemset in D. Since we are maintaining this matrix for each itemset, we are implicitly capturing the dependence between the values in the mixed categorical and continuous attribute space.

A point is defined to be linked to another point in the mixed attribute space if a) they are linked in the categorical data space, and b) if their continuous attributes adhere to the joint distribution as indicated by the covariance matrix. To determine whether a point's continuous attributes adhere to the distribution, we check to what degree the point violates the covariance matrix. Consider the vector P containing the continuous attributes of a data point which also contains the itemset d (in a slight abuse of notation, we express this as $d \subset P$). As stated above, for each itemset d in the lattice, there is a covariance matrix C, so let C^d be the covariance matrix for itemset d, and C^d_{ij} is then the covariance between continuous attributes i and j. To determine the degree to which P violates C^d_{ij}, we calculate the covariance score $c^{P,d}_{ij}$ for each pair of continuous attributes i and j of P:

$$c^{P,d}_{ij} = (P_i - \mu^d_i) \times (P_j - \mu^d_j) \tag{2}$$

where μ^d_i and μ^d_j are the means of the attributes i and j for all points in the data set containing the itemset d. We can then compute the violation score V of a point P as:

$$V_\tau(P) = \sum_i \sum_j v_\tau(P_{ij}) \tag{3}$$

where

$$v_\tau(P_{ij}) = \begin{cases} 0 & \text{if } | \frac{c^{P,d}_{ij} - C^d_{ij}}{\sigma_{C^d_{ij}}} | \leq \tau \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

and

$$\sigma^2_{C^d_{ij}} = \frac{1}{sup(d) - 1} \sum_{\{P | d \subset P\}} \left(c^{P,d}_{ij} - C^d_{ij} \right)^2 \tag{5}$$

Under the assumption that $\frac{c^{P,d}_{ij} - C^d_{ij}}{\sigma_{C^d_{ij}}}$ is normally distributed, the value of τ represents the width of a confidence interval for c^P_{ij} (or a measure of the degree of a violation we are willing to tolerate), and can be chosen from a table of the cumulative normal distribution.

We can now modify our anomaly score function for a point P to account for mixed attribute data as follows:

$$Score_2(P) = \sum_{d \subseteq P} \left(\frac{1}{|d|} \mid (C1 \vee C2) \text{ is true} \right) \tag{6}$$

where *C1*: $sup(d) \leq s$, and *C2*: $V(P) > \delta$. δ is then the maximum violation score allowed for each covariance matrix. Condition *C1* is the same condition used to find outliers in a categorical data space using $Score_1$, and condition *C2* adds continuous attribute checks to $Score_1$. This score function therefore captures:

- Dependencies between categorical attributes (C1).
- Dependencies between continuous attributes (C2).
- Dependencies between categorical and continuous attributes, since we maintain a covariance matrix for each itemset in the lattice.

3.2 Distributed Model Construction

As mentioned earlier, our basic approach is for each site to build a local model independently, and then combine these models to build a global model, which each site then uses to detect outliers in its subset of the complete data set. Furthermore, the local models can be constructed in a single pass of each site's subset of the data, and outliers can be detected in a second pass using the global model.

3.2.1 Local Model Construction

We must estimate all necessary model parameters in a single pass of the local data set. This is trivial for the support counts of each itemset in the lattice (i.e. $sup(d)$). For the covariance matrices, we note that after applying some basic algebra, we have:

$$C_{ij}^d = \frac{S_{ij}^d}{sup(d) - 1} + \frac{L_i^d \times L_j^d}{sup(d) \times (sup(d) - 1)} \tag{7}$$

where

$$S_{ij}^d = \sum_{\{P | d \subset P\}} P_i \times P_j \tag{8}$$

and

$$L_i^d = \sum_{\{P|d \subset P\}} P_i. \tag{9}$$

Similarly, for computing the violation score (see Equation 5), we note that

$$\sigma^2_{C_{ij}^d} = \frac{S_{c_{ij}^d} + 2C_{ij}^d L_{c_{ij}^P} + sup(d) \times (C_{ij}^d)^2}{sup(d) - 1} \tag{10}$$

where

$$S_{c_{ij}^d} = \sum_{\{P|d \subset P\}} (c_{ij}^{P,d})^2 \tag{11}$$

and

$$L_{c_{ij}^d} = \sum_{\{P|d \subset P\}} c_{ij}^{P,d}. \tag{12}$$

Note that the means μ_i and μ_j used in Equation 2 can be computed from L_i^d, L_j^d, and $sup(d)$. We also note that all of these values are exact, except for that of $\sigma^2_{C_{ij}^d}$, which must be calculated using only the partial estimates of μ_i and μ_j available at that point in the pass (see Equation 2). However, as more points are processed, the values of μ_i and μ_j will stabilize. At the end of the first pass, each site will have an itemset lattice, and for each itemset d in the lattice, there will be a support count $sup(d)$, matrices S^d and S_{c^d}, and vectors L^d and L_{c^d}. The psuedocode for the local model building algorithm is shown in Figure 2.

Procedure: BuildLocalModel
Input: $DataSet, MAXLEVEL$
Output: Local Model
  For each point $p \in DataSet$:
    G = Enumeration of all itemsets of size $MAXLEVEL$ for point p
    For each $g \in G$
      Update $sup(g)$, S^g, L^g, S_{c^g}, and L_{c^g} for g in the hash table
      Add all subsets of g of size $|g| - 1$ into G
    End For End For
end

Figure 2: Constructing the local model.

3.2.2 Global Model Construction

After the first pass of the data, the sites participate in a reduction operation to construct the global model. The reduction operation to produce the global model is a simple summation of the local models, since it

follows from Equations 7 to 12 that for each itemset d in the lattice:

$$Global(sup(d)) = \sum_{i=1}^{N} Local_i(sup(d)) \tag{13}$$

$$Global(S^d) = \sum_{i=1}^{N} Local_i(S^d) \tag{14}$$

$$Global(L^d) = \sum_{i=1}^{N} Local_i(L^d) \tag{15}$$

$$Global(S_{c^d}) = \sum_{i=1}^{N} Local_i(S_{c^d}) \tag{16}$$

$$Global(L_{c^d}) = \sum_{i=1}^{N} Local_i(L_{c^d}) \tag{17}$$

where N is the number of sites participating. From these values we can compute the global values of all model parameters. The result is that the global model is the same as a local model constructed by a site that has access to the complete data set.

3.3 Distributed Outlier Detection

In this section we detail the process of detecting outliers in the second pass of the algorithm. Once the reduction has been performed, each site will have a copy of the global model. In the second pass, each site, now armed with the global model, will calculate an anomaly score (see $Score_2$ in Equation 6) for each point in the local data set. Any point exceeding a threshold will be flagged as an outlier. The psuedocode for the outlier detection algorithm is shown in Figure 3. An anomaly score for a point is calculated based on the number of itemsets that it contains (as per function $Score_2$ in Equation 6) that are infrequent or whose corresponding covariances are violated. Our algorithm works as follows:

1. For each point, we enumerate all possible itemsets with a size of $MAXLEVEL = N$ into a set G.
2. For each of the itemsets $g \in G$, we increment its support count in the lattice.
3. We check if g's support is less than s. If so, we increase the score by a value inversely proportional to the size of the itemset, as dictated by the score function.

Procedure: FindOutliers
Input: $DataSet$, s, τ, δ, Δ*Score*,
ScoreWindowSize, $MAXLEVEL$
Output: Outliers discovered in one pass
For each point $p \in DataSet$:
G = All itemsets of size $MAXLEVEL$ in point p
For each $g \in G$
Get $sup(g)$ from the hash table
If $sup(g) < s$
$score(p) \leftarrow score(p) + \frac{1}{|g|}$
Else
If $V_\tau(p) > \delta$
$score(p) \leftarrow score(p) + \frac{1}{|g|}$
Add all subsets of g of size $|g| - 1$ into G
End For
If $score(p) >$ (average score in ScoreWindow $\times$ ΔScore)
Flag as Outlier
Else
Normal
Update the Score window of size *ScoreWindowSize*
End For
end

Figure 3: Finding outliers in the second pass of the data set.

4. If the itemset g is frequent, we check that the continuous attributes of the point are in agreement with the distribution determined by the covariance matrix using the covariance violation threshold δ. If so, we will not increase the score. Otherwise, we increase the score, again by a value inversely proportional to the size of the itemset.

5. We find all subsets of g of size $|g| - 1$ and insert these into G.

6. If G is not empty, go to step 2, otherwise continue.

7. We maintain all reported scores over a recent window of size *ScoreWindowSize* and calculate the average score in the window. If the total score for the point is greater than Δ*Score* $\times$ average score, then we flag this point as a local outlier.

3.4 Complexity Analysis and Modifications for Increased Efficiency

We first examine the computational complexity of our outlier detection algorithm as presented above. For a feature space with N_c categorical attributes and N_q continuous attributes, let M be the maximum number

of distinct values taken by any categorical attribute. For a data set of size n,

$$\textit{Total execution time} \leq n \times N_q^2 \times \sum_{i=1}^{MAXLEVEL} M^i \binom{N_c}{i}$$

$$= O\left(nN_q^2 \left(MN_c\right)^{N_c}\right)$$

Thus, execution times scales linearly with n, quadratically with N_q and exponentially with MN_c. However, there are several modifications we can make to our algorithm so that it is more efficient in terms of both memory usage and execution time, especially in regards to the exponential complexity.

3.4.1 Using a Partial Lattice

Notice that in our data structure we keep a matrix S^d and a vector L^d for each itemset $d \in D$. This is clearly unrealistic in terms of memory requirements for data sets with high dimensionality. One modification is to reduce the number of itemsets we have to examine. Instead of storing all itemsets (with their respective matrices and vectors), we only store itemsets and corresponding matrices and vectors if those itemsets have a size less than or equal to a value $MAXLEVEL \leq N$. This also aids in reducing the amount of computation performed. For some applications, the amount of memory used by our algorithm or the time taken to determine if a point is an outlier may be too costly. In these cases, it would be better to use a smaller value for $MAXLEVEL$. Note that if we use a value of $MAXLEVEL < N$, the accuracy of the results will be affected. However, if we are given two 1-itemsets I_1 and I_2, then we know that the frequency of the 2-itemset $I_1 \wedge I_2$ is bounded by the smaller of the frequencies for I_1 and I_2. As a result, the frequencies of the lower order itemsets serve as good approximations for the frequencies of the higher order itemsets. In the general case, however, accuracy improves as $MAXLEVEL$ increases. We empirically validate this in Section 4.

3.4.2 Modifying the Anomaly Score Function

Another approach to reducing the number of nodes in the lattice we must consider is to modify our score function as follows:

$$Score_3\left(P\right) = \sum_{d \subseteq P} \left(\frac{1}{|d|} : (C1 \vee C2) \wedge C3 \text{ is true}\right)$$

The conditions $C1$ and $C2$ are the same as in $Score_2$ (see Equation 6). The new condition, $C3$ is defined as follows: $C1 \vee C2$ holds true for every superset of d in P. Condition $C3$ is a heuristic and allows for more efficient processing since if an itemset does not satisfy conditions $C1$ and $C2$, then none of its subsets will be considered. In other words, for point $P1 : (A, B, C)$ in Figure 1, if the itemset ABC is frequent and if covariances for itemset ABC agree with the continuous attribute values in point $P1$ (within a certain threshold), none of its subset itemsets AB, BC, AC, A, B, C need to checked. The rationale for $C3$ is as follows. First, as stated by the Apriori property in Lemma 1, subsets of frequent itemsets will also be frequent and thus, will not satisfy $C1$. Second, covariances intuitively tend to be less discriminatory as the size of the itemset is reduced, since they take into account more points, and only reflect the most general trends in the data set.

If we use $Score_3$, we greatly reduce the number of itemsets (and corresponding matrices and vectors) that we need to consider. This leads to a decrease in both memory consumption and execution time, since we only need to consider the itemsets between $MAXLEVEL$ and the *positive border*. The positive border is defined to be the set of the frequent itemsets which have no frequent supersets. In this way, in the worst case, our anomaly detection algorithm examines the itemsets between $MAXLEVEL$ and the positive border of the lattice. The following example (see Figure 1) illustrates the use of $Score_3$ in our algorithm.

3.4.3 Example

Note that in Figure 1, frequent itemsets are labeled using uppercase characters, while infrequent itemsets are labeled using lowercase characters. Consider the points $P1 = (A, B, C)$ and $P2 = (A, B, D)$. For simplicity, we will assume that the continuous attributes of $P1$ and $P2$ are always in agreement with the covariance matrices, and so are not shown here. The three-itemset corresponding to $P1$ is frequent, so we only need to calculate the score for itemset ABC. As for $P2$, ABD is not frequent. Therefore, we increment the score and examine its subsets AB, AD, and BD. AB and AD are frequent, so we only have to increment the score for both itemsets and disregard their subsets. However, itemset BD is not frequent. As before, we increment the score and examine its subsets, B and D. B and D are both frequent, so we just increment the score for both and move on to the next point. Note that in our algorithm, we do not have to examine all of the subsets of itemset ABD.

3.4.4 Complexity of the Algorithm with Modifications

The time complexity of our algorithm grows linearly in the size of the data set and quadratically in the number of continuous attributes. Furthermore, our algorithm maintains and examines the itemset lattice beginning at $MAXLEVEL \leq N$ down to the positive border. Therefore, the time it takes to check if a point is an outlier is proportional to the number of lattice levels between $MAXLEVEL$ and the positive border. Moreover, a decade of research (Agrawal & Srikant, 1994; Veloso et al., 2002) has shown that under practical considerations, an itemset lattice can be managed in an efficient and scalable fashion, which is also evident in our experimental results.

3.5 Single Pass Approach for Dynamic Data Sets

Many applications, such as network intrusion detection, demand that outlier detection be performed on dynamic or streaming data. In such an environment, it is impossible to make more than one pass over the data. In this section we present a variation on our outlier detection algorithm to discover outliers in distributed dynamic data sets.

Since the model parameters (see Equations 7 to 12) are computed incrementally, it is straightforward to use them immediately to detect outliers. Our basic single-pass approach exploits this fact and uses the values of the model parameters computed from all previous points to compute the anomaly score of the current point. The drawback of this approach is that it requires an anomaly score be computed for a point using incomplete knowledge of the full data set. This is particularly a problem for the first several points of the data set, when the model parameters have been estimated by only a few points, and so are relatively unstable. However, this approach has the advantages that it is the only appropriate approach available for dynamic or streaming data, and with some simple extensions, it can with the problem of concept drift. Our algorithm for single-pass distributed outlier detection is given in Figure 4. It is very similar to the second-pass outlier detection algorithm given in Figure 3, except that each new point to be examined is incorporated into the local model.

The major concern for the single-pass approach is how to build a global model. Because of the dynamic nature of the data, each site will always have only a partial local model. However, the sites can still construct a global model from their partial local models in much the same away as in Section 3.2.2. The difference is that the global model must be reconstructed regularly, since the local models are continually being updated.

Procedure: OnePassFindOutliers
Input: $DataSet$, s, τ, δ, Δ*Score*,
ScoreWindowSize, $MAXLEVEL$
Output: Outliers discovered in one pass
For each point $p \in DataSet$:
G = All itemsets of size $MAXLEVEL$ in point p
For each $g \in G$
Get $sup(g)$ from the hash table
Update $sup(g)$, S^g, L^g, S_{c^g}, and L_{c^g} for g in the hash table
If $sup(g) < s$
$score(p) \leftarrow score(p) + \frac{1}{|g|}$
Else
If $V_\tau(p) > \delta$
$score(p) \leftarrow score(p) + \frac{1}{|g|}$
Add all subsets of g of size $|g| - 1$ into G
End For
If $score(p) >$ (average score in ScoreWindow $\times$ ΔScore)
Flag as Outlier
Else
Normal
Update the Score window of size *ScoreWindowSize*
End For
end

Figure 4: Finding outliers in a single pass of the data set.

However regularly updating the models is an expensive operation considering the size of the lattice and the available network bandwidth. A more effi cient alternative would be to exchange local outliers between sites. If all sites agree that a point is an outlier, then we can assume that the point is a global outlier.

Our distributed single pass algorithm is presented in Figure 5. Each site runs the *OnePassFindOutliers* algorithm and the parameters are set uniformly across all sites. The sites only communicate when some user-specifi ed event occurs. Examples of such events include a user's query for the global outliers, when a site fi nishes processing a fi xed number of points, or when a site fi nds a fi xed number of outliers. When such an event occurs, each site broadcasts all outliers it has found since the last event to all other sites. Upon receiving a broadcast, a site will check all the points it has received in the broadcast to see if they are flagged as outliers locally, and will return that information to the requesting site. If, for a given point, all sites agree that it is a local outlier, then it is flagged as a global outlier.

There are trade-offs to consider when choosing which type of event to use. For example, since outliers are rare by defi nition, the simplest event to use is that a site requests validation each time it receives a new local outlier. However, this approach requires a synchronization between sites for any outlier found at a local site. As an alternative, we can choose an event where a site accumulates k potential local outliers before

broadcasting them for validation. In this case, there is less synchronization between the sites, but there is a greater communication cost. This increase in communication cost stems from the fact that the sites are receiving information about global outliers less often, which increase the number of local outliers that need to be broadcast at each event.

Distributed One-Pass Outlier Detection Algorithm

```
For each incoming point
   Process the point using OnePassFindOutliers()
   If it is found to be a local outlier,
      Add it to the set of outliers
       to be verified at next event
   If an event occurs,
      Broadcast the set of potential outliers
       to all other sites
      Receive normal/outlier labels from all the sites
      If all other sites also flag a point as an outlier,
         Flag the point to be a global outlier
   If another site needs outlier verification,
      Receive the points, and process them
      Return outlier/normal labels to that site
End For
```

Figure 5: Distributed one-pass algorithm for outlier detection

3.5.1 Modifications for Increased Accuracy

As motivated previously, several applications (e.g. network intrusion detection) demand outlier detection over dynamic data. In order to capture concept drift and other dynamics of the data, we need to update our model to capture the most recent trends in the data. We achieve this by introducing a bias towards the more recent data. We maintain the itemsets together with their frequency counts in a hash table and the bias is introduced in the following two ways: First, we maintain an itemset together with its frequency count in the hash table only if it appears at least once every W points, and second, the frequency for every itemset in the hash table will be decremented by a value Δf every W points. We apply these biases using smart hash-table management coupled with a down-counting approach described in Figure 6.

Our hash table management works as follows. For every W points, we create a new hash table with index $i \; div \; W$ and delete the oldest hash table with index ($i \; div \; W - 2$). Thus, at every instant in time, we maintain at most two hash tables. We estimate the frequency for an itemset based on its frequency in the two hash tables. Moreover, for every W points, relevant itemsets will have their frequencies biased with a value of $-\Delta f$. The oldest hash table is deleted every W points and the two most recent hash tables will contain

Hash Table Management:
Let i be the index of the point that just arrived
If $i \; mod \; W = 0$,
  Then create a new hash table with index $i \; div \; W$
  delete the hash table with index $i \; div \; W - 2$

Frequency Estimation:
When estimating the frequency for an itemset from a point with index i:
If an itemset for the i^{th} point is found in hash table with index $i \; div \; W$
  Increment and use the freq. from this hash table
Else
  If itemset is found in hash table with index $i \; div \; W - 1$
    insert it into the hash table with index $i \; div \; W$,
    set freq = freq in table $(i \; div \; W - 1) + 1 - \Delta f$
    and then use this freq.
  Else
    insert it into the hash table with index $i \; div \; W$
    with freq. = 1

Figure 6: Algorithm for hash table management and frequency estimation

the relevant itemsets with their biased frequencies.

4 Experimental Results

4.1 Setup

We evaluate our algorithm's performance in an eight-node cluster, where each node has dual 1 GHz Pentium III processors and 1 GB of memory, running Red Hat Linux 7.2. This setting also give us the ability to vary network bandwidth between the nodes and allows us to evaluate performance with varying network bandwidth and latency. Our implementations are in C++ and we use MPI for message passing. We evaluate our algorithm[1] on the following data.

4.1.1 KDDCup 1999 Intrusion Detection Data

The 1999 KDDCup data set (Hettich & Bay, 1999) contains a set of records that represent connections to a military computer network where there have been multiple intrusions and attacks by unauthorized users. The raw binary TCP data from the network has been processed into features such as *connection*

[1]For all experiments unless otherwise noted, we use the following parameter settings: *s = 10, τ = 1.96, δ = 30% , ΔScore = 10, ScoreWindowSize = 40.*

duration, *protocol type*, *number of failed logins*, and so forth. This data set was obtained from the UCI KDD archive (Blake & Merz, 1998). The training data set has 4,898,430 data instances with 32 continuous attributes and 9 categorical attributes. The testing data set is smaller and contains several new intrusions that were not present in the training data set. These data sets have an unrealistic number of attacks. Therefore, we pre-process them such that intrusions constitute 2% of the data set and proportions of different attacks is maintained. Since intrusion packets tend to occur in bursts in network data, each intrusion instance is not randomly inserted in the data, but occur in bursts that are randomly distributed in the data set. The processed training data set contains 983,561 instances with 10,710 attack instances, while the processed testing data set contains 61,917 instances with 1,314 attack instances.

4.1.2 Adult Data

The Adult data set (Blake & Merz, 1998), containing 48,842 data instances with 6 continuous and 8 categorical attributes, was extracted from the US Census Bureau's Income data set. Each record has features that characterize an individual's yearly income together with a class label indicating whether person made more or less than 50,000 dollars per year.

4.1.3 US Congressional Voting Data

The Congressional Votes data set consists of a representatives votes on 16 issues together with a label indicating if the representative was a Republican or Democrat. The data set has 435 data instances with 16 categorical attributes.

4.1.4 Synthetic Data

Since there are very few publicly available mixed-attribute data sets with labeled outliers, we wrote a synthetic data set generator to produce data to compare performance with existing algorithms, and with varying data set characteristics. The generator can produce data sets with a user-supplied number of continuous attributes (N_q) and categorical attributes (N_c). The data points are generated according to a multi-modal distribution, and the user supplies the number of modes M. Basically, each mode has associated with it an itemset of size N_c and a cluster of points in N_q-dimensional space. Therefore, the number of modes determines the number of distinct values for each of the categorical attributes, and so has an effect on the

size of the itemset lattice. The cluster of points associated with each itemset results from randomly generating points uniformly in a sphere, and then applying random transformations (scaling, shearing, translating, etc.) to ensure that the clusters have a different means and covariances. To create actual data sets for our experiments, we first generate a set of normal points from one distribution, and then separately generate a much smaller set of outliers from another distribution. The two sets are then randomly mixed to produce the final data set. In our experiments, we consider a synthetic data set with 100,000 normal points and 1,000 outliers. We expect the synthetic data set we have generated to be more challenging than the real data data sets we use because the distribution of the outliers is not guaranteed to be significantly different from that of the normal points, and this is empirically verified in our experiments.

4.2 Qualitative evaluation

4.2.1 KDDCup 1999

We compare our approach against ORCA (Bay & Schwabacher, 2003), a state-of-the-art distance-based outlier detection algorithm which uses the Euclidean and Hamming distances as similarity measures for the continuous and categorical attributes respectively. ORCA finds the top-k outliers in a data set, with k being a user-supplied parameter. We set k equal to the number of outliers in the data set. For the comparison between ORCA and our algorithm, we used the 10% subset of the KDDCup 1999 training data as well as the testing data set, as ORCA did not complete in a reasonable amount of time on the full training data set.

Detection rates for the two approaches are reported in Table 1 (Note that "n/a" indicates that the attack was not present in that particular data set). Since the intrusion packets tend to occur in bursts, we mark an intrusion as detected if at least one instance in a burst is flagged as an outlier. Consequently, the detection (true positive) rates are in terms of the number of intrusion bursts detected, while false positive rates are in terms of number of normal packets marked as outliers. *Overall, our detection rates are very good and much better than those of* ORCA. *Our approach provides an overall detection rate of 95% compared to a detection rate of 45% afforded by ORCA*. Furthermore, our approach gives us a false positive rate of 0.35%. This false positive rate is extremely good for anomaly detection schemes, especially considering our high detection rates. ORCA has a false positive rate of 0.43%, but this is not as significant considering its lower detection rates. Our algorithm's low false positive rate can be attributed to a better representation for anomalous behavior obtained by capturing the dependence between categorical attributes and continuous

Attack Type	Detection Rate (10% Training) Our Approach/ORCA	Detection Rate (Testing) Our Approach/ORCA	Detection Rate (Training) Our Approach
Apache 2	n/a	100%/0%	n/a
Buffer Overflow	94%/63%	90%/100%	91%
Back	100%/5%	n/a	98%
FTP Write	28%/88%	n/a	33%
Guess Password	100%/21%	100%/0%	100%
Imap	100%/13%	n/a	100%
IP Sweep	42%/3%	28%/0%	37%
Land	100%/66%	n/a	100%
Load Module	100%/100%	n/a	100%
Multihop	94%/57%	70%/75%	94%
Named	n/a	100%/40%	n/a
Neptune	92%/1%	n/a	98%
Nmap	94%/8%	n/a	91%
Perl	100%/100%	n/a	100%
Phf	0%/25%	20%/100%	0%
Pod	45%/12%	100%/18%	54%
Port Sweep	100%/13%	100%/3%	100%
Root Kit	33%/70%	n/a	33%
Satan	75%/9%	n/a	72%
Saint	n/a	100%/1%	n/a
Sendmail	n/a	50%/50%	n/a
Smurf	24%/1%	21%/0%	22%
Snmpgetattack	n/a	52%/0%	n/a
Spy	100%/100%	n/a	100%
Teardrop	50%/1%	n/a	30%
Udpstorm	n/a	0%/0%	n/a
Warez Client	48%/3%	n/a	43%
Warez Master	0%/15%	n/a	25%
Xlock	n/a	50%/66%	n/a
Xsnoop	n/a	100%/100%	n/a

Table 1: Detection rates - (a) Our approach and ORCA on 10% KDDCup1999 training data set (b) entire testing data set, (c) Our approach on entire KDDCup 1999 training data set

attributes. On the testing data set our approach is able to detect most of the unknown attacks (a problem for almost all of the KDDCup 1999 participants). Note that our approach is a general anomaly detection scheme and has not been trained to catch specific intrusions – one can use this approach in conjunction with a signature detector to handle this problem (Mahoney & Chan, 2002). In terms of execution time, our algorithm processed the 10% sample of the KDDCup 1999 training data set in approximately *8 minutes*. ORCA, on the other hand, took *212 minutes* to process the same data set.

4.2.2 Adult Data

Here we are unable to make a comparison with ORCA, since this data set does not provide us with labeled outliers. The following data records are marked as the top outliers:

- A 39 year old self-employed male with a doctorate, working in a Clerical position making less than 50,000 dollars per year
- A 42 year old self-employed male from Iran, with a bachelors degree, working in an Executive position making less than 50,000 dollars per year
- A 38 year old Canadian female with an Asian Pacific Islander origin working for the US Federal Government for 57 hrs per week and making more than 50,000 dollars per year.

Apart from these, the other outliers we find tend to be persons from foreign countries that are not well represented in this data set. On inspection, one can affirm that these records are indeed anomalous.

4.2.3 US Congressional Voting Data

Here we are unable to compare with ORCA because, as in the previous case, this data set does not contain labeled outliers. Many of the top outliers contain large numbers of missing values, though there are examples in which this is not the case. One such example is a Republican congressman, who has four missing values, but also voted significantly differently from his party on four other bills. According to the conditional probability tables supplied with the data set, the likelihood of a Republican voting in a such a manner is very low.

4.2.4 Synthetic Data

We consider a synthetic data set with 8 categorical attributes and 10 continuous attributes. As ORCA uses hamming distance, it is unable to effectively capture distance between categorical attributes. Consequently, it achieves a detection rate of 55% with a false positive rate of 0.45%. Our algorithm achieves a detection rate of 76.2% with a false positive rate of 0.2% because of a better representation of categorical data. ORCA processed the data set in 350 seconds, while our algorithm processed this same data set in 214 seconds. We would like the reader to note that when we have fewer categorical attributes (less than 3) compared with continuous attributes, ORCA outperforms our algorithm. With fewer itemsets, covariance matrices tend to not contain decisive information. ORCA, on the other hand, uses Euclidean distance to measure distance between continuous attributes. Euclidean distance outperforms our covariance-based score when we have very few categorical attributes. Consequently, our detection rates and false positive rates increase as the number of categorical attributes decrease.

4.3 Speedup in a Distributed Setting

First, we explore the speedup obtained when running our distributed outlier detection algorithm on two, four, and eight sites. The KDDCup 1999 training data set and the synthetic data set is split evenly between the sites for this experiment. The other data sets are too small for this experiment. Figure 7(a) shows the speedup obtained on the two data sets. As there is only one round of communication, the overall message passing overhead is minimal. Most of the time is spent in the two phases a) building the local model in the first pass and b) finding outliers in the second pass. Consequently, each node works independently, and we see up to 7.7-fold speedup on 8 sites. The speedup appears to be linear. The slight reduction in speedup with increasing number of sites is due to increasing communication overhead associated the local model exchange.

Next, we vary the link bandwidth between the sites in our controlled environment in order to simulate a network spanning larger distances. As shown in Figure 7(b), for a wide area setting consisting of eight nodes, efficiency varies from a low of 69% to a high of 96%, for link bandwidths equal to 1 MB/s and 100 MB/s respectively. Even when link bandwidth is equal to 10 MB/s, our approach achieves an efficiency of 95%. This is suggestive of good scalability for outlier detection within an organization, for sites that can be on different subnets.

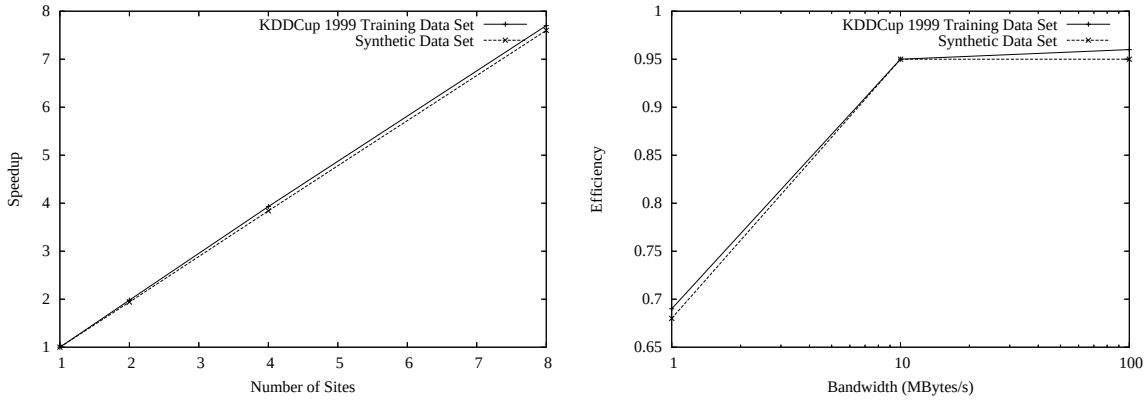

Figure 7: (a) Speedup (b) Expected efficiency in a wide area network with 8 sites

4.4 Benefits of the Approximation Scheme

We measure the execution time, detection rate and the false positive rate as a function of the number of itemset lattice levels maintained. The primary benefit of our approximation scheme is that we achieve far better execution times if we maintain fewer lattice levels, as can be seen in Figure 8. However, our detection rates decrease as the number of lattice levels decrease. Figure 9 shows detection rates and false positive rates with increasing lattice levels on the KDDCup 1999 training and testing data sets[2]. False positive rates are not affected significantly with changing lattice levels. On the other hand, detection rates seem to increase as the number of lattice levels increase to 3, after which they stabilize. This indicates the use of 3-attribute dependencies in the outlier detection process. We also note that the processing rate per 1000 network transactions is within reasonable response time rates even for a lattice level equal to four. Ideally, one would like to tune the application based on the learning curves in an application specific manner, as each application will exhibit a specific dependence behavior. Empirically, it appears from Figure 8 that execution time grows quadratically with the number of lattice levels.

4.5 Execution time variation with number of attributes

We next evaluate how execution time varies with increasing number of categorical and continuous attributes using synthetic. First, we measure execution time as we increase the number of categorical attributes, with 10 continuous attributes. Next, we measure execution time as we increase the number of continuous attributes, with 4 categorical attributes. As can be seen in Figure 10, in both cases there appears to be a

[2]Detection and false positive rates for the Adult data set are not available since the data is unlabeled

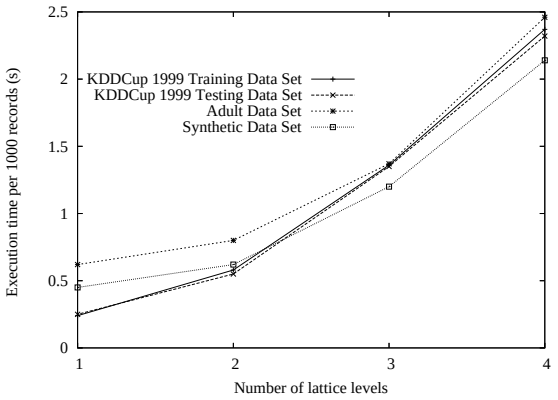

Figure 8: Execution time variation with increasing lattice levels

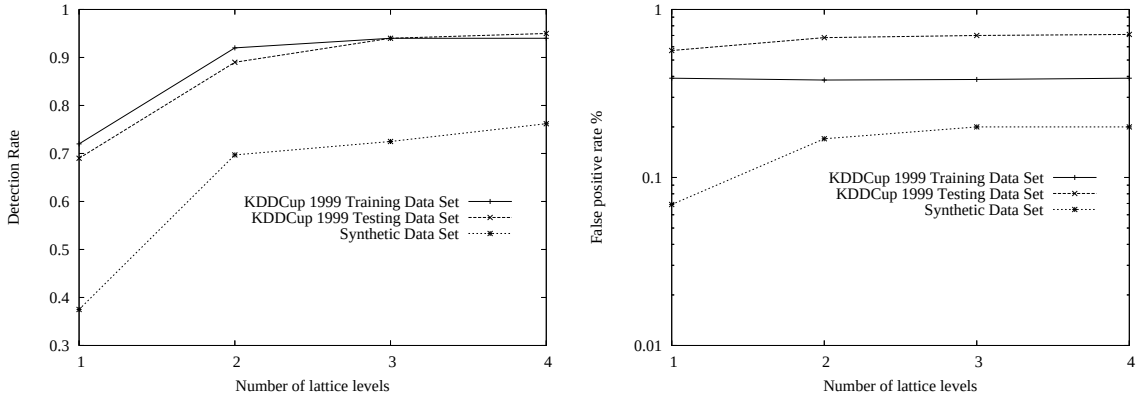

Figure 9: (a) Detection rates (b) False positive rates

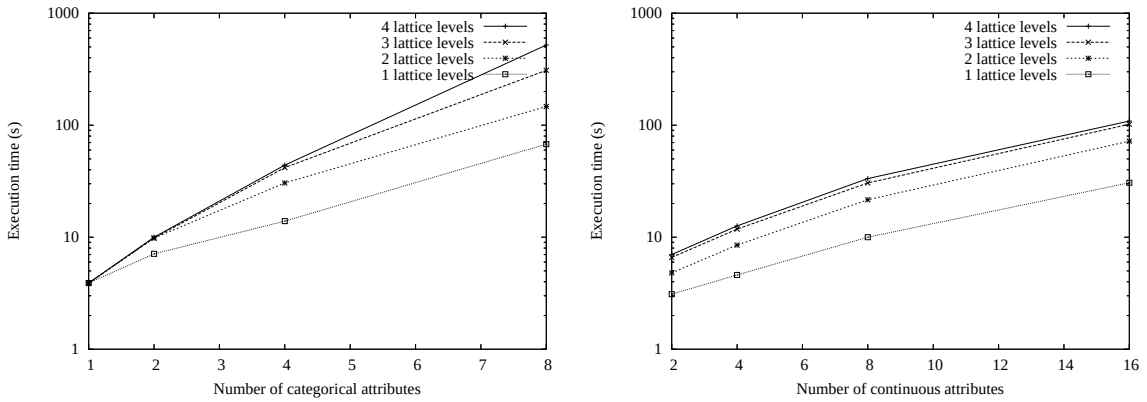

Figure 10: Execution time variation with increasing (a) Categorical attributes and (b) Continuous attributes

polynomial dependency between execution time and the number of categorical and continuous attributes.

4.6 Evaluation of one pass approach

4.6.1 Impact on detection quality

First, we evaluate the detection quality of our one-pass algorithm in a centralized setting on the KDDCup 1999 and synthetic data sets. Comparing Tables 1 and 2 tells us that detection quality does not deviate signifi cantly from that of our two-pass approach on the KDDCup 1999 data set. The false positive rate is remains unaffected. This trend also holds true for the synthetic data set, on which the detection rate fell to 0.75 (from 0.762) with the false positive rate remaining the same. Next, we compare our algorithm's detection quality as we scale up from a centralized to a distributed setting with two and four sites. Table 2 presents detections rates for different attacks in a centralized and distributed setting for the KDDCup 1999 data set. Our detection rates remain nearly unchanged as we move to two and four sites. The slight reduction in detection rate is attributed to data points that are flagged as local normals when they are global outliers. Similarly, the detection rate fell to 0.74 on the synthetic data set with four sites. On both the data sets, false positive rate remains unaffected. One can see that in practice, the number of global outliers that are being missed is not that signifi cant.

4.6.2 Scalability

We explore the speedup obtained when running our distributed one pass outlier detection algorithm on two, three and four sites. The KDDCup 1999 and synthetic data sets are evenly split between the nodes for

Attack Type	Detection rate Centralized Training/Testing	Detection rate Distributed (2 sites) Training/Testing	Detection rate Distributed (4 sites) Training/Testing
Apache 2	n/a/100%	n/a/100%	n/a/100%
Buffer Overflow	91%/90%	91%/90%	91%/90%
Back	97%/n/a	97%/n/a	96%/n/a
FTP Write	33%/n/a	33%/n/a	33%/n/a
Guess Password	100%/100%	100%/100%	100%/100%
Imap	100%/n/a	100%/n/a	100%/n/a
IP Sweep	37%/28%	35%/28%	32%/28%
Land	100%/n/a	100%/n/a	100%/n/a
Load Module	100%/n/a	100%/n/a	100%/n/a
Multihop	94%/100%	94%/100%	94%/100%
Named	n/a/100%	n/a/100%	n/a/100%
Neptune	98%/n/a	96%/n/a	94%/n/a
Nmap	91%/n/a	90%/n/a	90%/n/a
Perl	100%/n/a	100%/n/a	100%/n/a
Phf	0%/20%	0%/20%	0%/20%
Pod	53%/100%	52%/100%	52%/100%
Port Sweep	100%/100%	100%/98%	100%/97%
Root Kit	33%/n/a	33%/n/a	33%/n/a
Satan	72%/n/a	71%/n/a	70%/n/a
Saint	n/a/100%	n/a/100%	n/a/100%
Sendmail	n/a/50%	n/a/50%	n/a/50%
Smurf	22%/21%	21%/20%	20%/20%
Snmpgetattack	n/a/52%	n/a/52%	n/a/52%
Spy	100%/n/a	100%/n/a	100%/n/a
Teardrop	49%/n/a	49%/n/a	49%/n/a
Udpstorm	n/a/0%	n/a/0%	n/a/0%
Warez Client	43%/n/a	42%/n/a	41%/n/a
Warez Master	0%/n/a	0%/n/a	0%/n/a
Xlock	n/a/50%	n/a/50%	n/a/50%
Xsnoop	n/a/100%	n/a/100%	n/a/100%

Table 2: Detection rates for the single pass approach (Centralized and Distributed) for KDDCup 1999 data

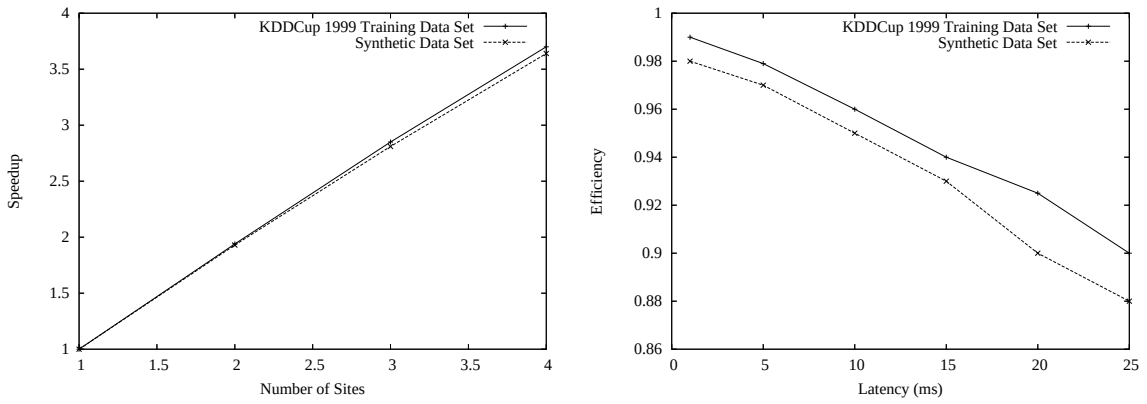

Figure 11: (a) Speedup for one pass approach (b) Expected efficiency in a wide area network

this experiment. Figure 11(a) shows speedup obtained on the two data sets. As there are relatively few outliers in the data set, and we have a low false positive rate, there is very little communication overhead. Therefore, there is very little synchronization between the sites and each site is able to work independently. As the number of nodes increases, the communication overhead also increases, as more nodes are involved in the local outlier exchange. As a result we see a slight reduction from the ideal speedup. For this reason, efficiency falls to 95% on the two data set when using four sites.

As nodes primarily communicate by exchanging outliers, which are small messages, link latency will be the primary performance bottleneck in a wide area setting. We vary the average link latency between the nodes in our controlled environment to simulate a wide area network spanning larger distances. As shown in Figure 11(b), efficiency falls to 90% for the KDDCup 1999 data set when using 4 sites, with an average link latency of 25ms. This is representative of networks spanning across several states (for instance, we have a 25ms average latency between a machine located at The Ohio State University and the University of Rochester) and excellent scalability. We would like to note that execution time per 1000 transactions for the one-pass approach does not change significantly when compared to the two pass approach. This is because the cumulative work done in the one-pass and two-pass approaches is nearly the same.

5 Conclusions

Outlier detection is an important problem for a wide range of data mining applications. To date, there have not been any general-purpose outlier detection algorithms that work on mixed attribute data in a distributed setting. In this paper, we presented the first general-purpose distributed outlier detection algorithm that

addresses these concerns. We also extended our algorithm to handle dynamic and streaming data sets. Experimental results validate the effectiveness of our approach on several real and synthetic data sets. In the future, we will examine alternative approaches for building models of mixed attribute data sets that have reduced memory requirements and allow for faster processing.

References

Agrawal, R., & Srikant, R. 1994. Fast algorithms for mining association rules. *Proc. of the International Conference on Very Large Data Bases VLDB* (pp. 487–499). Morgan Kaufmann.

Barnett, V., & Lewis, T. 1994. *Outliers in statistical data*. John Wiley.

Bay, S. D., & Schwabacher, M. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Blake, C., & Merz, C. 1998. UCI machine learning repository.

Bolton, R. J., & Hand, D. J. 2002. Statistical fraud detection: A review. *Statistical Science, 17*, 235–255.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. 2000. LOF: Identifying density-based local outliers. *Proc. of the ACM SIGMOD International Conference on Management of Data*.

Gamberger, D., Lavrač, N., & Grošelj, C. 1999. Experiments with noise filtering in a medical domain. *Proc. of the International Conference on Machine Learning*.

Ghoting, A., Otey, M. E., & Parthasarathy, S. 2004. Loaded: Link-based outlier and anomaly detection in evolving data sets. *Proceedings of the IEEE International Conference on Data Mining*.

Guha, S., Rastogi, R., & Shim, K. 2000. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems, 25*, 345–366.

Hettich, S., & Bay, S. 1999. KDDCUP 1999 dataset, UCI KDD archive.

Huang, Y.-A., & Lee, W. 2003. A cooperative intrusion detection system for ad hoc networks. *Proc. of the ACM workshop on Security of ad hoc and sensor networks (SASN)* (pp. 135–147). Fairfax, Virginia: ACM Press.

Jain, A. K., & Dubes, R. C. 1988. *Algorithms for clustering data*. Prentice Hall.

Johnson, T., Kwok, I., & Ng, R. 1998. Fast computation of 2-dimensional depth contours. *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Knorr, E., Ng, R., & Tucakov, V. 2000. Distance-based outliers: Algorithms and applications. *VLDB Journal*.

Knorr, E., & Ng, R. T. 1998. Algorithms for mining distance-based outliers in large datasets. *Proc. of the International Conference on Very Large Databases*.

Lazarevic, A., Ertoz, L., Ozgur, A., Kumar, V., & Srivastava, J. 2003. A comparative study of outlier detection schemes for network intrusion detection. *Proc. of the SIAM International Conference on Data Mining*.

Locasto, M. E., Parekh, J. J., Stolfo, S. J., Keromytis, A. D., Malkin, T., & Misra, V. 2004. *Collaborative distributed intrusion detection* (Technical Report CUCS-012-04). Department of Computer Science, Columbia University in the City of New York.

Mahoney, M. V., & Chan, P. K. 2002. Learning nonstationary models of normal network traffi c for detecting novel attacks. *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Otey, M., Noronha, R., Li, G., Parthasarathy, S., & Panda, D. 2002. Nic-based intrusion detection: A feasibility study. *IEEE ICDM Workshop on Data Mining for Cyber Threat Analysis*.

Palpanas, T., Papadopoulos, D., Kalogeraki, V., & Gunopulos, D. 2003. Distributed deviation detection in sensor networks. *SIGMOD Record, 32*, 77–82.

Papadimitriou, S., Kitawaga, H., Gibbons, P. B., & Faloutsos, C. 2003. LOCI: Fast outlier detection using the local correlation integral. *Proc. of the International Conference on Data Engineering*.

Penny, K. I., & Jolliffe, I. T. 2001. A comparison of multivariate outlier detection methods for clinical laboratory safety data. *The Statistician, Journal of the Royal Statistical Society, 50*, 295–308.

Porras, P. A., & Neumann, P. G. 1997. EMERALD: Event monitoring enabling responses to anomalous live disturbances. *Proc. of the 20th NIST-NCSC National Information Systems Security Conference* (pp. 353–365).

Rice, J. 1995. *Mathematical statistics and data analysis*. Duxbury Press.

Sequeira, K., & Zaki, M. 2002. ADMIT: Anomaly-based data mining for intrusions. *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Veloso, A. A., Meira Jr., W., de Carvalho, M. B., Possas, B., Parthasarathy, S., & Zaki, M. J. 2002. Mining frequent itemsets in evolving databases. *Proc. of the SIAM International Conference on Data Mining*.

Zhang, Y., & Lee, W. 2000. Intrusion detection in wireless ad-hoc networks. *Mobile Computing and Networking* (pp. 275–283).

Zhang, Y., Lee, W., & Huang, Y.-A. 2003. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, *9*, 545–556.