A Unified Approach for Mining Outliers

Edwin M. Knorr and Raymond T. Ng

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4 Canada

Abstract

This paper deals with finding outliers (exceptions) in large datasets. The identification of outliers can often lead to the discovery of truly unexpected knowledge in areas such as electronic commerce, credit card fraud, and even the analysis of performance statistics of professional athletes. One contribution of this paper is to show how our proposed, intuitive notion of outliers can unify or generalize many of the existing notions of outliers provided by discordancy tests for standard statistical distributions. Thus, when mining large datasets containing many attributes, a unified approach can replace many statistical discordancy tests, regardless of any knowledge about the underlying distribution of the attributes. A second contribution of this paper is the development of an algorithm to find all outliers in a dataset. An important advantage of this algorithm is that its time complexity is linear with respect to the number of objects in the dataset. We include preliminary performance results.

1 Introduction

The data mining community has noted that knowledge discovery tasks can be classified into 4 general categories: (i) dependency detection, (ii) class identification, (iii) class description, and (iv) exception/outlier detection. The first 3 categories of tasks correspond to patterns that apply to many objects, or to a large percentage of objects, in the dataset. Recent work in data mining (e.g., association rules [2, 7, 22, 23], classification [1], data clustering [9, 24, 26], and concept generalization [14, 20]) belong to these 3 categories. The 4th category, on the other hand, focuses on a very small minority of data objects, which are often discarded as noise. For many applications, such as credit card fraud, the monitoring of criminal activities in electronic commerce [19], and the analysis of performance statistics of professional athletes, identifying outliers can often lead to the discovery of truly unexpected knowledge. For example, professional hockey, baseball, football, or basketball players typically have a plethora of statistics recorded about their individual performances. Many such statistics are of trivial interest only; however, sometimes these statistics differ noticeably in certain respects from the statistics of all (or the vast majority) of the other players in the league. For instance, we might find that a certain poorly paid hockey player (who receives very little media attention) is actually the top performer in the league in shorthanded or power play shooting situations.

Some existing algorithms in machine learning and data mining have considered outliers, but only to the extent of tolerating them in whatever the algorithms are supposed to do [3, 9, 24, 26]. Almost all studies that consider outlier identification as their primary objective are in statistics. Barnett and Lewis provide a comprehensive treatment, listing about 100 discordancy tests for normal, exponential, Poisson, and binomial distributions [5]. The choices of appropriate discordancy tests depend on: (i) the distribution, (ii) whether or not the distribution parameters (e.g., mean and variance) are known, (iii) the number of expected outliers, and even (iv) the types of expected outliers (e.g., upper or lower outliers in an ordered sample). For example, for a normal distribution with unknown mean and unknown variance, there are separate discordancy tests for: single upper outliers, upper outlier pairs, k upper outliers, single lower outliers, lower outlier pairs, k lower outliers, upper and lower outlier pairs, etc. There are other tests if the mean is not known, or if the variance is not known. Furthermore, there are separate tests for other distributions. Yet, despite all of these options and decisions, there is no guarantee of finding outliers, either because there may not be any test developed for a specific combination, or because no standard distribution can adequately model the observed distribution. In a data mining context, the distributions of the attribute values are almost always unknown. To fit the observed distributions into standard distributions, and to choose suitable tests, requires non-trivial computational effort for large datasets. This motivates our study of a *unified* notion of outliers, defined as follows:

An object O in a dataset T is a $UO(p, D)$-outlier if at least fraction p of the objects in T are $>$ distance D from O.[1]

We use the term $UO(p, D)$-outlier as shorthand notation for a *U*nified *O*utlier detected using parameters p and D. Our unified notion is intuitive because it captures the general spirit of an outlier. Hawkins eloquently describes an outlier as "an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [16]. As such, our notion is a natural candidate for situations where the observed distribution does not fit any standard distribution, or where no discordancy test has been developed. A key contribution of this paper is the analysis showing that the $UO(p, D)$ notion of outliers unifies or generalizes the notions of outliers provided by many of the discordancy tests for standard distributions. In this paper, we show that for many discordancy tests, if an object O is an outlier according to a specific discordancy test, then O is also a $UO(p, D)$-outlier for some suitably defined p and D. We do not claim that our unified approach can replace all statistical discordancy tests.

Data mining, by definition, implies large quantities of data; therefore, efficiency or scalability is an important goal. Another key contribution of this paper is the development of a partitioning-based approach for detecting $UO(p, D)$-outliers. Its efficiency, particularly for datasets with low dimensionality, will be demonstrated by the performance results presented at the end of this paper. One advantage of the algorithm is that its time complexity is linear with respect to the number of objects in the dataset, and is efficient for datasets with low dimensionality. Another advantage of our approach is that its underlying data structure resembles a data cube [13]. This makes our method easily integratable with the many OLAP systems and data mining systems using data cubes, such as DBMiner [15].

2 Properties of $UO(p, D)$-Outliers: Relationships with Existing Notions

In this section, we show how our notion of $UO(p, D)$-outliers relates to existing notions of outliers, most of which turn out to be specific instances of ours.

Definition 1 We say that $UO(p, D)$ *unifies* or *generalizes* another definition *Def* for outliers, if there exist specific values p_0, D_0 such that object O is an outlier according to *Def* iff O is a $UO(p_0, D_0)$-outlier. □

Let N be the number of objects in the input dataset T. Each object is identified with the

[1] In our current work, we assume that the distance function is Euclidean in nature.

same k attributes; k is called the dimensionality of the dataset. Suppose there is an underlying distance metric function F that gives the distance between any pair of objects in T. For an object O, the D-neighbourhood of O contains the set of objects $Q \in T$ that are within distance D of O (i.e., $\{Q \in T \mid F(O,Q) \leq D\}$). The fraction p is the minimum fraction of objects in T that must be *outside* the D-neighbourhood of an outlier.

2.1 Outliers in Normal Distributions

For a normal distribution, outliers can be considered to be points that lie 3 or more standard deviations (i.e., $\geq 3\sigma$) from the mean μ [12].

Definition 2 Let T be a set of values that is truly normally distributed with mean μ and standard deviation σ. Define Def_{Normal} as follows: $t \in T$ is an outlier iff $\frac{t-\mu}{\sigma} \geq 3$ or $\frac{t-\mu}{\sigma} \leq -3$. □

Lemma 1 $UO(p,D)$ unifies Def_{Normal} with $p_0 = 0.9988$ and $D_0 = 0.13\sigma$, that is, t is an outlier according to Def_{Normal} iff t is a $UO(0.9988, 0.13\sigma)$-outlier.

Proof Outline: We use probabilities to reflect the number of points lying in a D-neighbourhood. Specifically, the probability is $1-p$ that the distance between 2 randomly selected points, O and Q, is less than or equal to D. Let T_1 and T_2 be random variables that are normally distributed with parameters μ and σ^2 (i.e., $T_1, T_2 \sim N(\mu, \sigma^2)$). Define $Z_1 = \frac{T_1-\mu}{\sigma}$ and $Z_2 = \frac{T_2-\mu}{\sigma}$ as standard normal variables (i.e., $Z_1, Z_2 \sim N(0,1)$). Using a table that lists areas for ranges $-\infty \leq Z_i \leq z$ under the standard normal curve to 4 decimal places, and given $p_0 = 0.9988$ and $D_0 = 0.13\sigma$, we present the following proof:

$$\begin{array}{lcr}
 & Pr(|T_1 - T_2| \geq 0.13\sigma) & \geq 0.9988 \\
\Longleftrightarrow & Pr(|T_1 - T_2| \leq 0.13\sigma) & \leq 0.0012 \\
\Longleftrightarrow & Pr(T_1 - 0.13\sigma \leq T_2 \leq T_1 + 0.13\sigma) & \leq 0.0012 \\
\Longleftrightarrow & Pr(\frac{T_1-\mu}{\sigma} - 0.13 \leq \frac{T_2-\mu}{\sigma} \leq \frac{T_1-\mu}{\sigma} + 0.13) & \leq 0.0012 \\
\Longleftrightarrow & Pr(Z_1 - 0.13 \leq Z_2 \leq Z_1 + 0.13) & \leq 0.0012 \\
\Longleftrightarrow & \Phi(Z_1 + 0.13) - \Phi(Z_1 - 0.13) & \leq 0.0012 \\
\Longleftrightarrow & Z_1 \leq -3.0 \quad \text{or} \quad Z_1 \geq 3.0 & \\
\Longleftrightarrow & T_1 \leq \mu - 3\sigma \quad \text{or} \quad T_1 \geq \mu + 3\sigma & \square
\end{array}$$

Note that if the value 3σ in Def_{Normal} is changed to some other value, such as 4σ, the above proof procedure can easily be modified with the corresponding p_0 and D_0 to show that $UO(p,D)$ still unifies the new definition of Def_{Normal}.

Barnett and Lewis give many specialized tests for identifying outliers in normal distributions with known or unknown means and/or standard deviations. Instead of using the standard normal curve, some of those tests use a t-distribution curve, which has a fatter tail than the normal curve. Nevertheless, the same proof procedure still works with appropriate modifications. For example, Barnett and Lewis give outlier test N6 (which we denote by Def_{N6}) for testing an upper and lower outlier pair in a normal distribution with unknown mean and variance. In a more detailed report [21], we give a proof showing that $UO(p,D)$ unifies Def_{N6} with $p_0 = 0.9990$ and $D_0 = 0.20$. Details and proofs are also available for the lemmas presented in the following sections.

2.2 Outliers in Other Distributions

Consider the following test for finding a single upper outlier in an exponential sample with unknown parameter [5]. Let the sampled values be ordered as $x_{(1)}, \ldots, x_{(n)}$, and let the test statistic T_1 be defined as $\frac{x_{(n)}}{\sum_{i=1}^{n} x_{(i)}}$. Let τ be the observed value for T_1, and $SP(\tau)$ be the significance probability that T_1 takes values more discordant than τ.

Barnett and Lewis illustrate this test with an example using 131 excess cycle times in steel manufacture, for which the distribution reasonably approximates an exponential distribution with parameter 0.14. Let us consider this example.

Definition 3 Define Def_{Exp} as follows. Observation t is an outlier (according to test statistic T_1 defined above) iff $SP(\tau) \leq 0.01$. For example, for the 131 excess cycle times, $x_{(131)} = 92$ is an outlier because $SP(\tau) = 0.00017 \leq 0.01$. □

Lemma 2 For an exponential distribution with parameter 0.14, $UO(p,D)$ unifies Def_{Exp} with $p_0 = 0.999$ and $D_0 = 0.0001$. □

For a binomial distribution, t can be defined as an outlier iff τ is greater than or equal to a specific critical value (listed in tables of such values [5]). Again, a proof is available showing that $UO(p, D)$ unifies this definition of outliers in a binomial distribution.

The following test can be used to find outliers in a Poisson distribution with parameter $\mu = 3.0$.

Definition 4 Define $Def_{Poisson}$ as follows: t is an outlier in a Poisson distribution with parameter $\mu = 3.0$ iff $t \geq 8$. □

Lemma 3 $UO(p, D)$ unifies $Def_{Poisson}$ with $p_0 = 0.9892$ and $D_0 = 1$. □

Proof: Let T_1 and T_2 be random variables defined on a Poisson density function with parameter $\mu = 3.0$ (i.e., $T_1, T_2 \sim P(3.0)$).

The probability distribution function for a Poisson distribution with parameter μ is:

$F_X(x) = Pr(X \leq x) = \sum_{i=0}^{x} \frac{e^{-\mu}\mu^i}{i!}$

Using a table that lists areas (to 4 decimal places) for ranges $0 \leq X \leq x$ under the $F_X(x)$ curve for $\mu = 3.0$, and given $p = 0.9892$ and $D = 1$, we compute:

$$\begin{array}{rcl} & Pr(|T_1 - T_2| \geq 1) & \geq 0.9892 \\ \Longleftrightarrow & Pr(|T_1 - T_2| \leq 1) & \leq 0.0108 \\ \Longleftrightarrow & Pr(T_1 - 1 \leq T_2 \leq T_1 + 1) & \leq 0.0108 \\ \Longleftrightarrow & Pr(F_X(T_1 + 1) - F_X(T_1 - 1)) & \leq 0.0108 \\ \Longleftrightarrow & T_1 \geq 8 & \Box \end{array}$$

2.3 Outliers in Regression Models

Consider the simple linear regression model given by the equation $y = \alpha + \beta x$. Suppose dataset T consists of observations of the form (x_i, y_i) for $i = 1, 2, \ldots, N$, that are fitted to this model. One possible way of identifying outliers is by examining the residuals, that is, the differences between the observed values and the fitted values. We denote the residual or error for the i'th observation by e_i in the equation $y_i = \alpha + \beta x_i + e_i$. An outlier among residuals can be considered to be a residual that is far greater than the rest in absolute value, and which lies at least 3 standard deviations from the mean of the residuals [8]. If we make the simplifying assumption (as some authors do) that residuals are independent and follow a normal distribution, then we can easily identify outliers using the same approach demonstrated in Section 2.1.[2]

The multivariate case is more complex. In general, it is quite difficult to come up with discordancy tests for multivariate regression models. Some authors believe that the best way to identify and address multivariate outliers is by the use of robust regression techniques [25]. In particular, a regression equation is computed that accounts for the majority of the data, and then outliers are computed as those points having large residuals from the robust equation.

In a general regression model, where k parameters are estimated from N observations, the residuals are not independent; the N residuals are associated with only $N - k$ degrees of freedom [8]. Consider the following example. Let T be a set of $N = 150$ observations for $k = 10$ parameters fitted to the linear regression model: $Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_{10} X_{10}$, and let the residual for the i'th observation be denoted by e_i.

Definition 5 Define $Def_{\mathrm{Regression}}$ as follows: $t_i \in T$ is an outlier *iff* e_i does not fall in the 99% confidence interval of Student's t-curve with 140 degrees of freedom. □

Lemma 4 $UO(p, D)$ unifies $Def_{Regression}$ with $p_0 = 0.99$ and $D_0 = 0.258$. □

We provide a proof of the lemma in a separate document [21].

We conclude this section by noting that the identification of outliers in regression models is a complex but important research topic in statistics. Nevertheless, despite its shortcomings, the relatively simple approach of identifying outlying values based upon outlying residuals is easily supported by our unified approach.

2.4 User Interaction

So far, we have seen how $UO(p, D)$-outliers can be used to find the same outliers obtained

[2]In general, residuals are not independent. For example, if the largest x_i value is sufficiently far from the other x_j values, the residual corresponding to x_i will be small. The regression equation is merely a best-fit, linear approximation.

via various statistical discordancy tests. Thus, given a definition of outliers (e.g., Def_{Normal}), we can find specific values of p_0 and D_0 such that the $UO(p_0, D_0)$-outliers are exactly the outliers according to the given definition. In these cases, the p and D values in $UO(p, D)$ are well-defined. A natural question is how to define the p and D values for situations where there is no appropriate statistical discordancy test. One reasonable approach is to provide effective visualization and efficient incremental tools for the user (who is not necessarily a domain expert) to experimentally vary the values of p and D. An efficient algorithm for finding unified outliers, such as the one to be presented in Section 3, is critical in achieving the performance demanded by usable interactive tools. Currently, our implementation of unified outliers does not try to obtain the best p and D values, for in general, this is very difficult to do. Instead, our system involves the user in the discovery process by asking the user to provide and change the values of p and D. We are currently working on sampling techniques to provide the user with suitable default values for the initial iteration. Without a suitable initial estimate, the user may run the program one or more times using values of D that are inappropriate for the dataset at hand, resulting in time consuming runs, especially for large datasets. By sampling $\hat{n}$ of the N tuples in the dataset, we can obtain a confidence level for our initial estimate of D, given p.

2.5 Comparison with Sequential Exceptions

Almost all studies in outlier identification are in statistics, which is why we focus on statistical discordancy tests. The only exception we have come across is the method proposed by Arning, *et al.* [4], in which the authors search a dataset for implicit redundancies, and extract data items called *sequential exceptions* that maximize the reduction in Kolmogorov complexity. Unlike $UO(p, D)$-outlier detection and all statistical discordancy tests and most distance-based data mining works (e.g., [10]), their method does not require a metric distance function.

Our unified approach does not rely on redundancies within the data; however, even for a dataset that is applicable to both our method and that of Arning, *et al.*, it is apparent that the two methods will not necessarily identify the same outliers. We do not pursue sequential exceptions any further in this paper.

2.6 Relationships with Data Clustering Algorithms

Data clustering algorithms assign similar objects in a dataset to the same classes; however, they provide little support for identifying outliers [11, 17, 18]. CLARANS, BIRCH, and DBSCAN are clustering algorithms designed specifically for data mining applications. In CLARANS [24], an object is (removed as) "noise" if its removal raises the silhouette coefficient of the clusters. In BIRCH [26], an object is (removed as) noise if it is "too far from its closest seed," where a seed is some representative object such as the centroid of a cluster. For both algorithms, outliers are not defined among the data objects directly, but merely as by-products of clustering.

In statistics, numerous clustering algorithms have been developed [17, 18]. The algorithms can be broadly divided into two categories: partitioning and hierarchical algorithms. In both cases, each data object is assigned to at least one cluster. The algorithms provide no further support for identifying outliers. The same comment applies to virtually all clustering algorithms developed for machine learning applications.

Unlike CLARANS and BIRCH, DBSCAN [9] provides a more direct role for outliers to play. DBSCAN classifies objects as core, border, or outlying, depending on: (i) the number of objects in an ϵ-neighbourhood, and (ii) the reachability or connectivity of the object being considered. The ϵ-neighbourhood is similar in principle to our D-neighbourhood, but ϵ is intended to be set to a small value to get strong clusters, whereas D is intended to be large to obtain strong outliers. Regarding the reachability and connectivity notions in DBSCAN, there is no counterpart in our framework. A key observation is that DBSCAN, as a clustering algorithm, aims to produce maximal-size clusters and is reluctant to label objects as outliers. In contrast, our framework is designed to directly implement the notion or the parameters of outliers chosen by the user, and is more generous than DBSCAN in labeling outliers. In the event that ϵ is equal to D, our set of outliers coincides with the set of border and outlying objects obtained by DBSCAN; however, in practice, D should be larger than ϵ. Last but not least, DBSCAN, and for that matter CLARANS and BIRCH, is not designed to unify the kinds of distribution-dependent discordancy tests mentioned in Section 1.

2.7 Advantages of a Unified Approach

In summary, our notion of unified outliers can replace many statistical discordancy tests. The flowchart in Figure 1 shows a possible flow of control for detecting outliers using popular discordancy tests. As explained in Section 1, there are numerous discordancy tests to choose from. For example, for a normal distribution having unknown mean and unknown variance, different discordancy tests exist depending on how many outliers are "expected" of various types (e.g., upper, lower, both upper and lower). The choice(s) of appropriate discordancy tests depend(s) on the distribution, on whether or not the distribution parameters are known, and on whether or not the number and types of outliers are known. All of these "parameters" may be very difficult to determine, if not impossible, especially for large datasets with many attributes and unknown distributions. Conversely, using our unified approach, many of these questions, and virtually all of the decision boxes in the flowchart, can be avoided. The dashed line with the label "Yes/No" in Figure 1 indicates that the complex flowchart can be simplified by the single box labelled "Detect $UO(p, D)$-Outliers".

While we do not claim that our unified model can replace all discordancy tests for all distributions, our model can certainly supersede many existing discordancy tests for standard distributions.

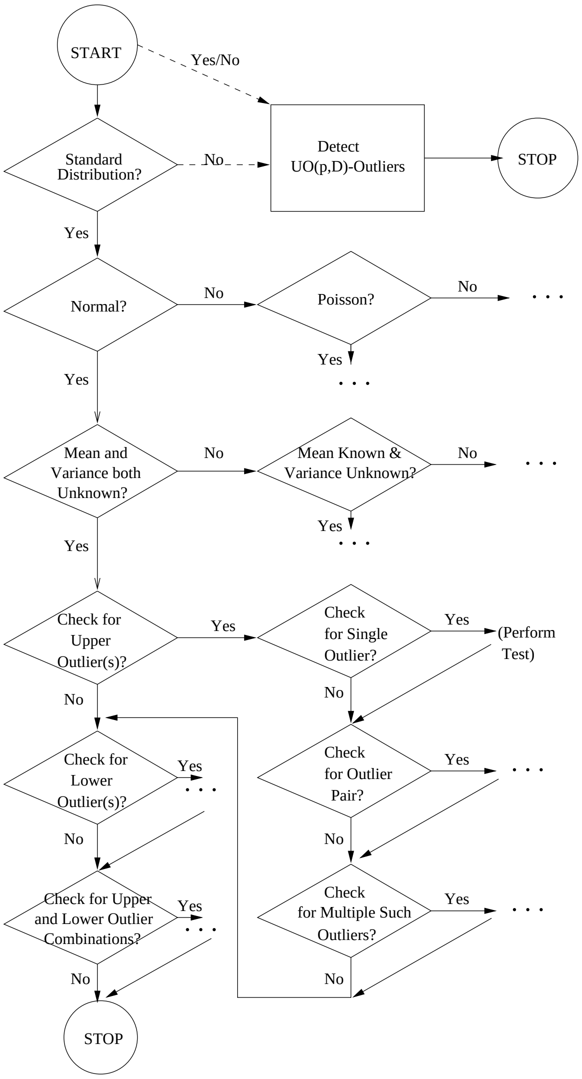

Figure 1: Possible Flow of Control using Statistical Discordancy Tests

3 An Algorithm for Finding All $UO(p, D)$-Outliers

In the previous sections, we analyzed how $UO(p, D)$-outliers relate to existing notions of outliers. Here, we show how all $UO(p, D)$-outliers in a dataset can be computed efficiently.

A naive algorithm for detecting all $UO(p, D)$-outliers for given values of p and D would be to count, for each object Q, the number of objects in the D-neighbourhood of Q. This has $O(N^2)$ time complexity, where N is total number of objects in the dataset. A more optimized algorithm is to build and search a spatial indexing structure, such as a k-d tree [6]. Then, for each object Q, the number of objects in its D-neighbourhood can be found using a range query. The time taken by this approach is between $O(N \log N)$ and $O(N^2)$, depending on the actual distribution of data and on the dimensionality of the dataset. Below, we present an algorithm that, for fixed dimensionality k, has a time complexity linear to N. This algorithm is particularly efficient for small values of k. In Section 3.5, we show some preliminary experimental results.

3.1 Underlying Cell Structure and Properties

Our approach for finding all $UO(p, D)$-outliers relies on an underlying cell structure. As will become obvious later, the idea is to reduce object-by-object processing to cell-by-cell processing, thereby gaining efficiency. For ease of presentation, we show the cell structure and its properties for the 2-dimensional case (i.e., $k = 2$). Later, we describe what changes are required to generalize to higher dimensions.

To begin, the 2-dimensional space spanned by the data objects is partitioned into cells or squares of length $l = \frac{D}{2\sqrt{2}}$. Let $C_{x,y}$ denote the cell that is at the intersection of row x and column y. The Layer-1 (L_1) neighbours of $C_{x,y}$ are all the immediate neighbouring cells of $C_{x,y}$ as defined in the usual sense, that is,

$$L_1(C_{x,y}) = \{C_{u,v} \mid u = x \pm 1, \\ v = y \pm 1,\ C_{u,v} \neq C_{x,y}\}. \qquad (1)$$

In the 2-dimensional case, a typical cell (except for cells that are on the boundary of the cell structure) has 8 L_1 neighbours.

Property 1 Any pair of objects within the same cell is at most distance $\frac{D}{2}$ apart. □

Property 2 If $C_{u,v}$ is an L_1 neighbour of $C_{x,y}$, then any object $P \in C_{u,v}$ and any object $Q \in C_{x,y}$ are at most distance D apart. □

Property 1 is valid because the length of a diagonal of a cell is $\sqrt{2}l = \sqrt{2}\frac{D}{2\sqrt{2}} = \frac{D}{2}$. Similarly, Property 2 is true because the distance between any pair of objects in the two cells cannot exceed twice the length of a diagonal of a cell. As will become obvious later, these two properties are useful in ruling out many objects as outlier candidates.

The Layer-2 (L_2) neighbours of $C_{x,y}$ are all the cells within 3 cells of $C_{x,y}$, that is,

$$L_2(C_{x,y}) = \{C_{u,v} \mid u = x \pm 3,\ v = y \pm 3, \\ C_{u,v} \notin L_1(C_{x,y}),\ C_{u,v} \neq C_{x,y}\}. \quad (2)$$

While Layer 1 is 1 cell thick, Layer 2 is 2 cells thick. This is significant because of the following property.

Property 3 If $C_{u,v} \neq C_{x,y}$ is neither an L_1 nor an L_2 neighbour of $C_{x,y}$, then any object $P \in C_{u,v}$ and any object $Q \in C_{x,y}$ must be at least distance D apart. □

Because the combined thickness of L_1 plus L_2 is 3 cells, the distance between P and Q must exceed $3l = \frac{3D}{2\sqrt{2}} > D$. In the 2-dimensional case, a typical cell (except for boundary cells) has $7^2 - 3^2 = 40$ L_2 cells.

3.2 Algorithm FindAllOuts

Using the cell structure described above, Algorithm FindAllOuts, shown in Figure 2, finds all $UO(p, D)$-outliers.

The key idea of our algorithm is summarized in the following property. For convenience, let M denote the maximum number of objects that can be *inside* the D-neighbourhood of an outlier (i.e., $M = N(1-p)$), and let m be the total number of cells.

1. For $q = 1, 2, \ldots m$, initialize $Count_q$ to zero.
2. For each object P, map P to an appropriate cell C_q, and increment $Count_q$.
3. For $q = 1, 2, \ldots, m$, if $Count_q > M$, label C_q red.
4. For each red cell C_r, label each of the L_1 neighbours of C_r pink, providing that neighbour has not already been labelled red.
5. For each non-empty, uncoloured cell C_w do:
   (a) Compute $Count_{w2} = Count_w + \sum_{i \in L_1(C_w)} Count_i$.
   (b) If $Count_{w2} > M$, label C_w pink.
   (c) else
      (1) Compute $Count_{w3} = Count_{w2} + \sum_{i \in L_2(C_w)} Count_i$
      (2) If $Count_{w3} \leq M$, mark all objects in C_w as outliers.
      (3) else for each object $P \in C_w$ do:
         (a) $Count_P = Count_{w2}$
         (b) For each object Q in each of the L_2 neighbours of C_w do:
            If $dist(P, Q) \leq D$ then
            (1) Increment $Count_P$.
            (2) If $Count_P > M$, P cannot be an outlier, so goto 5(c)(3).
         (c) Mark P as an outlier.

Figure 2: Algorithm FindAllOuts

Property 4 (a) If there are $> M$ objects in $C_{x,y}$, then *none* of the objects in $C_{x,y}$ is an outlier.

(b) If there are $> M$ objects in $C_{x,y} \cup L_1(C_{x,y})$, then *none* of the objects in $C_{x,y}$ is an outlier.

(c) If there are $\leq M$ objects in $C_{x,y} \cup L_1(C_{x,y}) \cup L_2(C_{x,y})$, then *every* object in $C_{x,y}$ is an outlier. □

Properties 4(a) and 4(b) are direct consequences of Properties 1 and 2, and 4(c) is due to Property 3. Cells satisfying Property 4(a) are labelled red in Step 3 of Algorithm FindAllOuts. Cells that are L_1 neighbours of a red cell are labelled pink in Step 4, and cannot contain any outlier because of Property 4(b). Similarly, other cells satisfying Property 4(b) are labelled pink in Step 5(b). Then, in Step 5(c)(2), cells satisfying Property 4(c) are identified.

Note that these properties and the steps mentioned above help to identify outliers or non-outliers on a cell-by-cell basis, rather than on an object-by-object basis. This kind of "bulk processing" reduces execution time significantly. Unfortunately, there are cells that do not satisfy any of the three properties, and object-by-object computation must be carried out. Such cells are denoted as C_w in Step 5(c)(3). Each object P in C_w must be compared with every object Q in a cell that is an L_2 neighbour of C_w, to determine how many Q's are inside the D-neighbourhood of P. As soon as the count exceeds M, P is declared a non-outlier. If, after examining all Q's, the count remains $\leq M$, then P is an outlier.

3.3 Complexity Analysis

We now analyze the complexity of Algorithm FindAllOuts for the 2-dimensional case. Step 1 takes $O(m)$ time, where $m \ll N$. Steps 2 and 3 take $O(N)$ and $O(m)$ time, respectively. Since M is the maximum number of objects that can appear in the D-neighbourhood of an outlier, there are at most $\frac{N}{M}$ red cells, and Step 4 takes $O(\frac{N}{M})$ time. The time complexity of Step 5 is the most complicated. In the worst case, (i) no cell is labelled red or pink in the previous steps, and (ii) Step 5(c) is necessary for all cells. Since no cell is coloured, each cell contains at most M objects. Thus, in Step 5(c), each of the objects in a cell can require the checking of up to M objects in each of the 40 L_2 neighbours of the cell; therefore, $O(40M^2)$ time is required for each cell. Hence, Step 5 takes $O(mM^2) \approx O(m)$ time in the worst case, because M is intended to be a very small number.[3] Thus, the time complexity of Algorithm FindAllOuts is $O(m+N)$ for the 2-dimensional case. Note that this complexity figure is very conservative, because in practice, we expect many red and pink cells. As soon as this happens, there are fewer object-to-object comparisons; therefore, Step 5(c) becomes less dominant, and the algorithm requires less time. In Section 3.5, we show experimental results on the efficiency of Algorithm FindAllOuts.

3.4 Generalization to Higher Dimensions

To generalize from the 2-dimensional case to higher dimensions, Algorithm FindAllOuts requires no change. The only modification that is necessary is to have a general k-dimensional cell structure. Since the length of a diagonal of a k-dimensional hypercube/cell of length l is $\sqrt{k}l$, the length l must be changed to $\frac{D}{2\sqrt{k}}$ to maintain Properties 1 and 2.

The L_1 neighbours of cell $C_{x_1,\ldots,x_k}$ are:

$$\begin{aligned} L_1(C_{x_1,\ldots,x_k}) = \\ \{C_{u_1,\ldots,u_k} \mid u_i = x_i \pm 1 \ \ \forall\, 1 \le i \le k, \\ C_{u_1,\ldots,u_k} \ne C_{x_1,\ldots,x_k}\} \end{aligned} \tag{3}$$

which generalizes the definition given in (1). However, to maintain Property 3, so that Algorithm FindAllOuts can function correctly, the definition of L_2 neighbours needs to be modified. Because $l = \frac{D}{2\sqrt{k}}$, Layer 2 needs to be thicker than it was for $k=2$. In particular, if x denotes the thickness of Layer 2, then the combined thickness of Layers 1 and 2 is $x+1$. For Property 3 to hold, we need $(x+1)l > D$. Thus, we pick x as $\lceil 2\sqrt{k}-1 \rceil$. Accordingly, the L_2 neighbours of $C_{x_1,\ldots,x_k}$ are:

$$\begin{aligned} L_2(C_{x_1,\ldots,x_k}) = \\ \{C_{u_1,\ldots,u_k} \mid u_i = x_i \pm \lceil 2\sqrt{k} \rceil \ \ \forall\, 1 \le i \le k, \\ C_{u_1,\ldots,u_k} \notin L_1(C_{x_1,\ldots,x_k}), \\ C_{u_1,\ldots,u_k} \ne C_{x_1,\ldots,x_k}\} \end{aligned} \tag{4}$$

which generalizes (2). In this way, Properties 1 to 4 remain valid.

While Algorithm FindAllOuts requires no change for the general k-dimensional case, its complexity changes. First, while the time complexities of Steps 1 to 4 remain the same, m is exponential with respect to k, and is not necessarily much less than N. Second, the complexity of Step 5 is no longer $O(m)$, but $O(m(2\lceil 2\sqrt{k} \rceil + 1)^k) \approx O(c^k)$, where c is some constant depending on $\sqrt{k}$ and on $m^{1/k}$ (which roughly corresponds to the number of cells along each dimension). Thus, the complexity of the entire algorithm is no longer $O(m+N)$, but $O(c^k + N)$. While this complexity figure represents a worst case scenario, the question to ask is how efficient Algorithm FindAllOuts is in practice for the general k-dimensional case. Early experience indicates that the algorithm is more efficient in practice than the complexity suggests, for the following reasons:

- For the identification of "strong" outliers, the number of outliers to be found is intended to be small. This is achieved by having a relatively large value of D, and a fraction p very close to 1. For large values of D, there only needs to be a small number of cells along each dimension because, as shown by the cell structure, the length of a cell is directly proportional to D. Thus, the constant c is small, although it is greater than 1.

- For large values of p (i.e., $p \approx 1$), M is small, implying that there will be numerous red and pink cells. This means that the savings realized by skipping red and pink cells is enormous, and that the number of objects requiring pairwise comparisons is relatively small.

- Recall that m denotes the total number of cells in the k-space. As k increases, there is an increasing number of empty cells.

[3] Since M depends on N and p (i.e., $M = N(1-p)$), we can equate $O(mM^2)$ to $O(mN^2(1-p)^2)$. In practice, we expect p to be extremely close to 1, especially for large datasets, thus $O(mN^2(1-p)^2)$ can be approximated by $O(m)$.

We have developed an optimized version of Algorithm FindAllOuts that creates cells only when they are non-empty and that only processes non-empty cells. The time complexity no longer depends on m, but on n instead, where n denotes the number of non-empty cells. In practice n is much smaller than m.

In ongoing work, we are studying how to optimize $UO(p, D)$-outlier detection for high dimensional cases (e.g., $k > 10$).

3.5 Preliminary Experimental Results

We ran our algorithm against various datasets ranging in size from hundreds of records to 100000 records.[4] Even the small datasets were interesting to mine because they dealt with real-life data (e.g., laboratory and examination marks of large classes of students, and performance statistics of professional hockey players).

For a concrete example, we ran our algorithm on a 16-attribute, 856-record dataset consisting of 1995-96 performance statistics of players in the National Hockey League. These statistics include numbers of goals, assists, points, penalty minutes, shots on goal, power play goals, etc. For a 3-dimensional cell structure containing 1000 cells, and using parameter $0.979 \leq p \leq 0.99$, the search times ranged from 0.11 to 0.17 seconds. These times are approximately 50% of the k-d tree range query times. When we ran our algorithm against much larger, synthetic datasets (whose random distribution mirrors that found in the 1995-96 hockey file), we obtained results even more dramatic. For 100000 tuples and $0.99 \leq p \leq 0.99999$ (approximately), our algorithm yielded search times ranging from 0.52 to 95.09 seconds. In comparison, a somewhat optimized brute force approach required 4.54 to 447.68 seconds. Searches using various kinds of k-d trees exceeded the CPU timer limit of 2147 seconds.

[4]We do not claim that 100000 records is a large dataset by today's standards, but it is a decent size on which to perform large numbers of comparisons while varying p and D. Our preliminary work is encouraging. Future work will involve much larger synthetic and real-life datasets.

N	CS	BF	KD
10000	0.18	0.21	96.62
20000	0.32	1.02	412.17
30000	0.43	2.25	955.45
40000	0.54	4.26	1675.85
50000	0.61	6.11	>2147
60000	0.74	9.64	>2147
70000	0.88	13.78	>2147
80000	1.04	17.58	>2147
90000	1.12	22.07	>2147
100000	1.43	27.67	>2147

Table 1: Performance Results (in Seconds) for Algorithm FindAllOuts using Cell Structure (CS) Mode, Brute Force (BF) Mode, and k-d tree (KD) Mode. N is the number of tuples in the dataset; $p = 0.9995$.

Table 1 contrasts the performance and scalability of various modes of implementation of Algorithm FindAllOuts. We used synthetic datasets ranging between 10000 and 100000 data points. The modes of operation described in the table are:

- CS mode, which is a 3-dimensional cell structure implementation, as described in Section 3
- BF mode, which is a partially optimized brute force implementation that computes exactly one distance between each pair of points, but terminates neighbour comparisons as soon as the outlier threshold M is exceeded
- KD mode, which is a k-d tree range query implementation[5]

The time to load and build each structure is excluded from the table; however, that time is small. For example, loading and mapping/building time for $N = 100000$ tuples for: (i) CS mode was 4.40 seconds, (ii) BF mode was 4.07 seconds, and (iii) KD mode was 9.17 seconds.

[5]The statistics shown are for an adaptive, median-splitting implementation of k-d trees. We experimented with other types of k-d trees; however, the adaptive implementation typically outperformed the others.

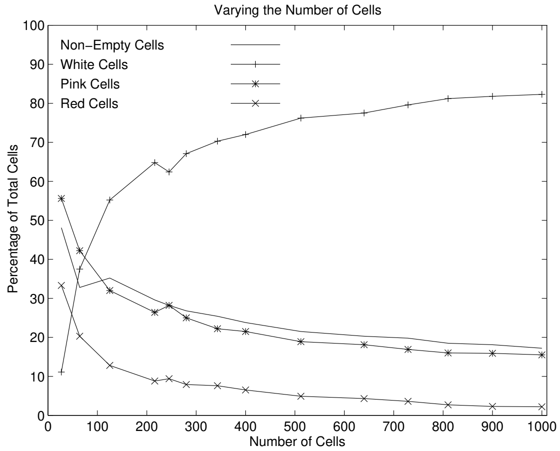

Figure 3: Percentages of Different Colours of Cells versus Total Number of Cells

p	Search Time	% White Cells	% Pink Cells	% Red Cells
0.99	95.09	83.1	15.1	1.8
0.997	16.25	69.9	25.1	5.0
0.999	3.12	56.3	33.7	10.0
0.9995	1.43	49.0	35.6	15.4
0.99991	0.52	35.7	33.4	30.9

Table 2: CS Mode: Search Times and Cell Colouring Statistics for $N = 100000$, $m = 1000$, and Variable p. The percentage of non-empty cells was 64.3% for this dataset.

Figure 3 shows how the percentage of cells in each of the categories—uncoloured (or white) cells, pink cells, red cells, and non-empty cells—varies with the total number of cells in the cell structure. The percentages of white, pink, and red cells add up to 100%. Even though the percentages of pink and red cells decrease as the total number of cells increases, the two percentages still account for a significant proportion of the overall cells (i.e., 20% or more). Furthermore, the percentage of non-empty cells also decreases, dropping below 20% rather quickly. This is significant in that these cells require the largest amount of computational effort. Table 2 shows how performance results are affected by varying p, and hence the percentages of coloured cells.

Finally, Figure 4 shows how search time scales with respect to the number of cells in the cell structure, for a constant number of outliers. When the number of cells is relatively small, search time actually increases because too many object-by-object comparisons are needed.

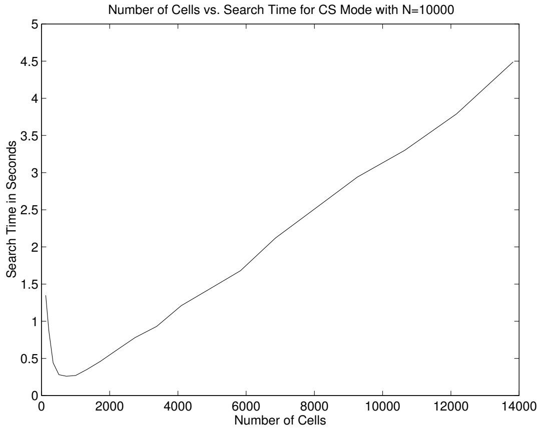

Figure 4: How Search Time Scales with the Number of Cells in the Cell Structure

These preliminary experimental results are encouraging because they suggest that Algorithm FindAllOuts using CS mode is a promising way of efficiently finding $UO(p, D)$-outliers.

3.6 Ongoing and Future Research

We believe that the development of a general framework for identifying outliers is a relevant and important research topic in data mining. We are actively studying strategies to efficiently detect outliers in large datasets (e.g., $N \geq 1000000$) with many dimensions (e.g., $k \geq 10$). These strategies will handle disk resident data.

One of the most important areas of our research concerns distance functions. We are researching ways to incorporate user-defined distance functions. For example, a user who is mining hockey data may wish to provide greater weight to shorthanded goals (versus regular strength or power play goals), or perhaps attach greater or less significance to a player's age, salary, or playing position. We are also investigating strategies to handle continuous, nominal, and ordinal data, rather than just discrete, cardinal data. Sampling techniques are being explored to provide suitable default values for p and D, thereby providing a reasonable starting point for user interaction.

The continual development of interactive tools is also important. We believe that the user should be involved in the knowledge discovery process since it is unreasonable to expect a machine to mechanically extract outliers for a generic assortment of attributes and distance functions.

Acknowledgements

We thank the anonymous referees for their constructive comments. This research has been partially sponsored by NSERC Grant OGP0138055 and IRIS-2 Grants HMI-5 & IC-5.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proc. 18th VLDB*, pages 560–573, 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD*, pages 207–216, 1993.

[3] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[4] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proc. KDD*, pages 164–169, 1996.

[5] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley & Sons, 1994.

[6] J. L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.

[7] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. ACM SIGMOD*, pages 255–264, 1997.

[8] N. Draper and H. Smith. *Applied Regression Analysis.* John Wiley & Sons, 1966.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD*, pages 226–231, 1996.

[10] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. ACM SIGMOD*, pages 163–174, 1995.

[11] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[12] D. Freedman, R. Pisani, and R. Purves. *Statistics.* W.W. Norton, New York, 1978.

[13] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. Technical Report MSR-TR-95-22, Microsoft Research, 1995.

[14] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th VLDB*, pages 547–559, 1992.

[15] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. KDD*, pages 250–255, 1996.

[16] D. Hawkins. *Identification of Outliers.* Chapman and Hall, London, 1980.

[17] A. Jain and R. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[18] L. Kaufman and P. Rousseeuw. *Finding Groups in Data.* John Wiley & Sons, 1990.

[19] E. M. Knorr. On digital money and card technologies. Technical Report 97-02, University of British Columbia, 1997.

[20] E. M. Knorr and R. T. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):884–897, 1996.

[21] E. M. Knorr and R. T. Ng. UO(p,D)-outliers: A unified notion of outliers. Unpublished Manuscript, Dept. of Computer Science, University of British Columbia, 1997.

[22] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proc. KDD*, pages 146–151, 1996.

[23] H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episodes in sequences. In *Proc. KDD*, pages 210–215, 1995.

[24] R. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. 20th VLDB*, pages 144–155, 1994.

[25] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley & Sons, 1987.

[26] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD*, pages 103–114, 1996.