

Outlier Detection By Sampling With Accuracy Guarantees

Mingxi Wu, Christopher Jermaine
Department of Computer and Information Sciences and Engineering
University of Florida
Gainesville, FL, USA, 32611
{mwu,cjermaine}@cise.ufl.edu

ABSTRACT

An effective approach to detect anomalous points in a data set is distance-based outlier detection. This paper describes a simple sampling algorithm to efficiently detect distance-based outliers in domains where each and every distance computation is very expensive. Unlike any existing algorithms, the sampling algorithm requires a fixed number of distance computations and can return good results with accuracy guarantees. The most computationally expensive aspect of estimating the accuracy of the result is sorting all of the distances computed by the sampling algorithm. This enables interactive-speed performance over the most expensive distance computations. The paper's algorithms were tested over two domains that require expensive distance functions as well as ten additional real data sets. The experimental study demonstrates both the efficiency and effectiveness of the sampling algorithm in comparison with the state-of-the-art algorithm and the reliability of the accuracy guarantees.

1. INTRODUCTION

One effective approach for detecting anomalous data points in a set is *distance-based* (DB) outlier detection [7, 12, 1]. In DB-outlier detection, each data point is represented as a point in a multi-dimensional feature space and a distance function is chosen based on domain-specific requirements. Data points that are significantly far away from all of the others are flagged as outliers.

Though it may seem that DB-outlier detection is a solved problem, it turns out that even using advanced algorithms, the method is computationally prohibitive even for small data sets. This is particularly true in domains that require computationally expensive distance functions. For example, the edit distance function for strings [13], the ERP distance function for time series [4], the quadratic-form distance function for color histograms [6] and various scoring matrices for aligning bioinformatics sequences [5] all are computationally expensive. Given two input points that are both of dimension n , the aforementioned distance functions require $\Theta(cn^2)$

time, where the constant c is usually above three. In these domains, state-of-the-art algorithms may take days to detect DB-outliers, even in data sets of small sizes.

The goal of this paper is to define an algorithm that can provide users with interactive-speed performance over the most expensive distance computations, giving the user the ability to “try out” various distance functions and queries during exploratory mining. The question we consider is: How can we reduce the required number of distance computations in DB-outlier mining? For certain distance measures and data sets, indexing and pruning techniques can be used to reduce the number of distance computations. Unfortunately, indexing [3] is not useful in the domains mentioned above due to high data dimensionality, and pruning [2] tends only to reduce the required number of distance computations to a few hundred or a few thousand per data point; as we will show experimentally, this is still too costly if every distance computation requires seconds of CPU time.

In this paper, we consider a simple sampling algorithm for mining the k th nearest neighbor (k th-NN) DB-outliers [12]. Given integer parameters k and γ , k th-NN outliers are the top γ points whose distance to its k th-NN is greatest. Sampling has been considered before as an option for detecting outliers [8], but never along with a rigorous treatment of the effect on result quality. If the user is able to tolerate some rigorously-measured inaccuracy, our algorithm can give arbitrarily fast response times.

Our algorithm works as follows. For each data point i , we randomly sample α points from the data set. Using the user-specified distance function, we find the k th-NN distance of point i in those α samples. After repeating the above process for each point, the sampling algorithm returns the γ points whose sampled k th-NN distance is greatest. Algorithm 1 presents the corresponding pseudo-code:

Algorithm 1 Sampling Algorithm

Input: A data set G of size n ; k , specifying which NN distance as the criterion; α ($\alpha > k$), the sample size; γ , the number of outliers to return.

Output: γ points as the k th-NN outliers

- 1: **for** each point $i \in G$ **do**
 - 2: Draw a random sample of size α from G (not including point i)
 - 3: Calculate point i 's k th-NN distance in its sample
 - 4: **end for**
 - 5: Return the top γ points whose k th-NN distance in its sample is the greatest
-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD '2006 Philadelphia, USA

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

Aside from the algorithm’s obvious simplicity, its biggest benefit is that it allows a user to control the total number of distance computations, thus bounding the time required to run the algorithm to completion. For example, with one million points and ten samples for each point, the algorithm will require ten million distance computations. During exploratory mining, such an algorithm may save significant user time by giving fast evidence that the data set in question lacks any meaningful outliers. Or, such fast results may detect an inappropriate distance function choice at an early stage. If the results are accurate or interesting enough, the algorithm may obviate the need to run an exact algorithm altogether.

Of course, sampling-based algorithms are of little use if it is not possible to provide the user with an idea of the result quality. This leads to this paper’s primary technical contribution. We formally analyze the statistical properties of the algorithm, and describe an effective technique that gives the user a statistical indication of the algorithm’s result quality. Specifically, we treat the number of the true top γ *kth*-NN outliers in Algorithm 1’s return set as a random variable whose characteristics can be used to measure the quality of the outliers returned by the algorithm. Thus, if we tell the user that the expected number of true outliers returned is 20 for a given data set, then this implies that if the sampling algorithm were run many times on that data set, on average 20 out of the 30 returned points will be among the true top 30 *kth*-NN outliers. In addition to the formal analysis, we demonstrate the effectiveness and efficiency of our proposed sampling algorithm and statistical estimation algorithms by giving extensive experimental results.

The remainder of the paper is organized as follows. In Section 2, we describe our overall process of providing quality guarantees for the sampling algorithm. In Section 3, a statistical analysis of the sampling algorithm is performed. Section 4 describes how the analysis can be applied to a constructed distance database in a highly scalable and efficient fashion. Section 5 details the experimental study on two expensive domains as well as ten real data sets. The paper is concluded in Section 6.

2. PROVIDING QUALITY GUARANTEES

Algorithm 1 (hereafter referred to as the “sampling algorithm”) is so simple that there is little benefit in discussing it in greater detail. However, the issue that clearly *does* deserve more attention is the question of how to give the user an understanding of exactly how accurate this algorithm is likely to be in practice; it is this question that we consider in detail in the remainder of the paper.

The paper describes a two-step process for analyzing the quality of the sampling algorithm for a given data set, so that the estimated result quality can be returned to the user along with the discovered outliers. In the first step, the distances between each point and their samples are used to (logically) construct a distance database D' , where the set of pairwise distances in D' is a reasonable approximation of the actual distance database D . By *distance database*, we refer to a matrix that stores the pairwise distance from point i to point j for every i and j . During the construction of D' , we wish to avoid additional distance computations beyond those required by the sampling algorithm. Thus, for a data point i , the sampled distances $\{d'_1, d'_2, \dots, d'_\alpha\}$ between i and the other points in its sample are replicated as needed

to serve as a surrogate for the actual neighbor distances $\{d_1, d_2, \dots, d_{n-1}\}$ of i . This reconstruction is identical to a type of reconstruction advocated when making use of the without-replacement bootstrap from statistics [11].

In the second step, an exact, statistical analysis of the quality of the sampling algorithm for the database D' is performed, and the result is returned to the user as an indication of the accuracy of the algorithm when it is applied to D . While it is true that bounds that are returned rely on the supposition that D' is in fact a reasonable surrogate for D , such assumptions are generally unavoidable in statistical analysis. An analogous assumption in classical sampling theory is that the sample variance can be used as a reasonable surrogate for the population variance [14]. In order to address this concern, Section 5 of the paper shows that over twelve real-life data sets, for various sample sizes, the bounds derived by our method do in fact reliably predict the accuracy of the sampling algorithm.

The process of deriving accuracy guarantees for the sampling algorithm is described in the next two Sections. Section 3 provides rigorously statistical analysis of the sampling algorithm. Section 4 describes how the analysis can be applied to D' in a highly scalable and efficient fashion.

3. STATISTICAL ANALYSIS

This Section describes a formal, probabilistic analysis of the quality of the sampling algorithm’s return set.

3.1 Quality measure N

Our analysis needs to be performed with respect to some measure of the algorithm’s quality. Let A be the set of the true top γ *kth*-NN outliers, and let A' be the return set of the sampling algorithm. Then the size of the intersection $A \cap A'$ is a reasonable measure of the sampling algorithm’s quality. Let N be a random variable denoting the size of this intersection set for a single run of the sampling algorithm over a given data set. In order to precisely characterize the sampling algorithm’s quality, we characterize the expectation and variance of N . These statistics describe the average number of correct outliers returned by the algorithm, as well as how much the number of correct outliers is expected to vary depending on how lucky (or unlucky) we happen to be.

In the rest of the paper, we will use $E[X]$ and $Var(X)$ to denote the expectation and variance of a random variable X respectively. We will also use the notation $Pr[B]$ to denote the probability that event B will occur.

3.2 Average Number of Correct Outliers

To compute $E[N]$ for a given data set, we begin with a mathematical expression for N . Let y_i evaluate to one if the i th point in the data set G is a true *kth*-NN outlier, and zero otherwise. We define a series of Bernoulli (zero/one) random variables of the form M_i , where M_i is one if the i th point is declared as an outlier by the sampling algorithm, and zero otherwise. Then:

$$N = \sum_{i=1}^n y_i M_i \quad (1)$$

Taking the expectation of both sides, we have:

$$E[N] = \sum_{i=1}^n y_i E[M_i] \quad (2)$$

Note that for a given input of the sampling algorithm, y_i is a constant. Thus, deriving an expression for $E[N]$ reduces to the problem of deriving an expression for $E[M_i]$. Since M_i is a Bernoulli random variable, this is equivalent to computing the probability that M_i evaluates to one.

3.2.1 How Often Is Point i Declared an Outlier?

M_i evaluates to one if point i is reported as an outlier by the sampling algorithm. Obviously, point i is flagged as an outlier only if there are at most $\gamma - 1$ points in G whose k th-NN distance in its sample is larger than point i 's. Let T_i be a random variable denoting the total number of such points. Then M_i is one if and only if $T_i \leq \gamma - 1$. Noting that T_i is asymptotically normally distributed¹, we have:

$$\begin{aligned} E[M_i] &= Pr[M_i = 1] \\ &= Pr[T_i \leq \gamma - 1] \\ &= Pr\left[\frac{T_i - E[T_i]}{\sqrt{Var(T_i)}} \leq \frac{\gamma - 1 - E[T_i]}{\sqrt{Var(T_i)}}\right] \\ &= \Phi\left(\frac{\gamma - 1 - E[T_i]}{\sqrt{Var(T_i)}}\right) \end{aligned} \quad (3)$$

In Equation 3, $\Phi(x)$ is the standard normal cumulative distribution function. In order to make use of $\Phi(x)$, we need to calculate the mean and variance of T_i . Before we do this, for each point $i \in G$, we introduce a random variable N_i , which denotes the sampled k th-NN distance of point i . In addition, we define a function $one(expr)$ that evaluates to one if and only if the boolean-type argument $expr$ is true, and zero otherwise. Given these notations, $T_i = \sum_{j,j \neq i} one(N_j > N_i)$. If $D_i = \langle d_1, d_2, \dots, d_{n-1} \rangle$ refers to the vector of point i 's neighbor distances ordered from the smallest to the largest, then the following lemma gives the formulas for computing the mean and variance of T_i . This lemma provides the base formulas that we will use to compute $E[N]$ in Section 4.

Lemma 1. *The mean and variance of T_i are given by:*

$$\begin{aligned} E[T_i] &= \sum_{d \in D_i} Pr[N_i = d] \sum_{j,j \neq i} Pr[N_j > N_i | N_i = d] \\ Var(T_i) &= E[T_i] - E^2[T_i] + sum1[T_i] - sum2[T_i] \end{aligned}$$

In Lemma 1, $sum1[T_i]$ and $sum2[T_i]$ are given by:

$$\begin{aligned} sum1[T_i] &= \sum_{d \in D_i} Pr[N_i = d] \left[\sum_{j,j \neq i} Pr[N_j > N_i | N_i = d] \right]^2 \\ sum2[T_i] &= \sum_{d \in D_i} Pr[N_i = d] \sum_{j,j \neq i} Pr^2[N_j > N_i | N_i = d] \end{aligned}$$

Proof. Taking the expectation of both sides of T_i 's expres-

sion:

$$\begin{aligned} E[T_i] &= E\left[\sum_{j,j \neq i} one(N_j > N_i)\right] \\ &= \sum_{j,j \neq i} Pr[N_j > N_i] \\ &= \sum_{j,j \neq i} \sum_{d \in D_i} Pr[N_i = d] \times Pr[N_j > N_i | N_i = d] \\ &= \sum_{d \in D_i} Pr[N_i = d] \times \sum_{j,j \neq i} Pr[N_j > N_i | N_i = d] \end{aligned}$$

$Var(T_i)$ can be proved similarly. \square

3.2.2 Probability Formulas in Lemma 1

In Lemma 1, we expressed the mean and variance of T_i in terms of the probability that $N_i = d$ and the conditional probability that $N_j > N_i$ given $N_i = d$. We now give the exact formulas to compute these two probabilities.

- *Computing $Pr[N_i = d]$.* Recall that steps (2)-(3) of the sampling algorithm draw a random sample (without replacement) of size α from point i 's $n - 1$ neighbors and compute i 's k th-NN distance in this sample. This process can be modeled by the hypergeometric distribution. In the hypergeometric distribution, a jar has N balls, M red, $N - M$ green. If a person were to reach in, blindfolded, and select K balls at random *without replacement*, the probability that exactly x balls of the K balls retrieved were red is given by $H(X = x | M, N, K)$. Analogously, suppose that point i 's q th-NN distance (denoted by d_q) is chosen to be N_i . Then we can regard the $q - 1$ NNs of point i as red balls, and those NNs who are further away than i 's q th-NN as green balls. Given this:

$$Pr[N_i = d_q] = \frac{\alpha}{n-1} H(X = k-1 | q-1, n-2, \alpha-1) \quad (4)$$

In Equation 4, $\frac{\alpha}{n-1}$ is the probability that the q th-NN of point i is included in the sample during a run of the sampling algorithm; k and α are the parameters of the sampling algorithm; n is the size of the input data set.

- *Computing $Pr[N_j > N_i | N_i = d]$.* Computing this conditional probability is nothing but finding out all the distances in N_j 's domain that are greater than d , and then summing the probabilities of observing those distances. Given point j 's sorted vector of neighbor distances $D_j = \langle d_1, d_2, \dots, d_{n-1} \rangle$, suppose that d_{min_1} is the smallest distance in this array that is larger than d . Then by making use of Equation 4:

$$Pr[N_j > N_i | N_i = d] = \sum_{q=min_1}^{n-1} Pr[N_j = d_q] \quad (5)$$

Note that it is the index q that solely determines the probability of $N_j = d_q$, given that the other parameters of the sampling algorithm are fixed.

3.3 Variance In Correct Outliers

Knowing on expectation how many correct outliers in the return set of the sampling algorithm is not enough. It is crucial to help the user to understand how the number of correct outliers is going to vary around its mean.

¹In general, T_i is normally distributed due to the Liapounov Central Limit Theorem. Since T_i is the sum of $n - 1$ independent Bernoulli random variables, this Theorem applies; see Lehmann [9], Corollary 2.7.1.

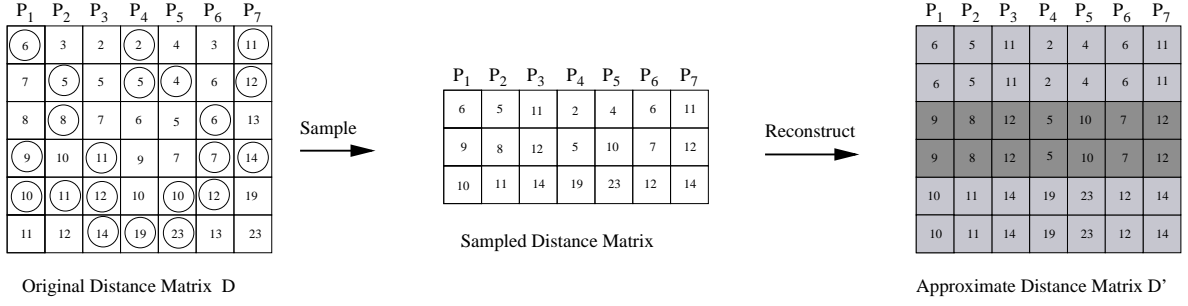


Figure 1: Replicating sample distances to construct D' .

The variance of N gives such a measure. This quantity is $Var(N) = E[N^2] - E^2[N]$. Since we have already discussed how to compute $E[N]$ in Section 3.2, the remaining task is to calculate the second moment of N :

$$\begin{aligned}
 E[N^2] &= E\left[\left(\sum_i y_i M_i\right)^2\right] \\
 &= E[N] + \sum_i \sum_{j, j \neq i} y_i y_j E[M_i M_j] \quad (6)
 \end{aligned}$$

Note that in Equation 6, y_i and y_j are constants for a given input of the sampling algorithm. Therefore, deriving an expression for $Var(N)$ reduces to deriving an expression for $E[M_i M_j]$. Since $M_i M_j$ is a Bernoulli random variable, this is equivalent to computing the probability that $M_i M_j$ evaluates to one.

3.3.1 How Often Is $M_i M_j$ One?

$M_i M_j$ evaluates to one if and only if both point i and point j are reported as outliers by the sampling algorithm. Point i and point j are both flagged as outliers only if there are at most $\gamma - 2$ points in G whose sampled k th-NN distance is larger than the smaller of i 's and j 's. Otherwise, either point i or j (or both) will not be flagged as outliers. In a manner analogous to the way that we use T_i to help calculate $E[M_i]$, we will use U_{ij} to help calculate $E[M_i M_j]$. Let U_{ij} be a random variable denoting the total number of the points whose sampled k th-NN distance is larger than the smaller of i 's and j 's sampled k th-NN distance. Given this, $M_i M_j$ is one if and only if $U_{ij} \leq \gamma - 2$. Noting that U_{ij} is asymptotically normally distributed (for reasons identical T_i 's normality), we have:

$$\begin{aligned}
 E[M_i M_j] &= Pr[M_i M_j = 1] \\
 &= Pr[U_{ij} \leq \gamma - 2] \\
 &= Pr\left[\frac{U_{ij} - E[U_{ij}]}{\sqrt{Var(U_{ij})}} \leq \frac{\gamma - 2 - E[U_{ij}]}{\sqrt{Var(U_{ij})}}\right] \\
 &= \Phi\left(\frac{\gamma - 2 - E[U_{ij}]}{\sqrt{Var(U_{ij})}}\right) \quad (7)
 \end{aligned}$$

In order to make use of $\Phi(x)$, we need to calculate the mean and variance of U_{ij} . To express U_{ij} in a mathematical form, we again introduce a series of random variables, where S_{ij} denotes the smaller of point i 's and point j 's k th-NN distances in their samples. Obviously, the domain of S_{ij} is $D_i \cup D_j$. Given these notations, we have $U_{ij} = \sum_{l, l \neq i, j} one(N_l > S_{ij})$. The following lemma gives the formulas for computing the mean and variance of U_{ij} . This lemma provides the base formulas that we will use to

compute $Var(N)$ in Section 4.

Lemma 2. *The mean and variance of U_{ij} are given by:*

$$\begin{aligned}
 E[U_{ij}] &= \sum_{d \in D_i \cup D_j} Pr[S_{ij} = d] \sum_{l, l \neq i, j} Pr[N_l > S_{ij} | S_{ij} = d] \\
 Var(U_{ij}) &= E[U_{ij}] - E^2[U_{ij}] + sum3[U_{ij}] - sum4[U_{ij}]
 \end{aligned}$$

In Lemma 2, $sum3[U_{ij}]$ and $sum4[U_{ij}]$ are given by:

$$sum3[U_{ij}] = \sum_{d \in D_i \cup D_j} Pr[S_{ij} = d] \left[\sum_{l, l \neq i, j} Pr[N_l > S_{ij} | S_{ij} = d] \right]^2$$

$$sum4[U_{ij}] = \sum_{d \in D_i \cup D_j} Pr[S_{ij} = d] \sum_{l, l \neq i, j} Pr^2[N_l > S_{ij} | S_{ij} = d]$$

The proof is similar to Lemma 1's proof.

3.3.2 Probability Formulas in Lemma 2

In Lemma 2, we express the mean and variance of U_{ij} in the form of the probability of $S_{ij} = d$ and the conditional probability of $N_l > S_{ij}$ given $S_{ij} = d$. We now give the exact formulas to compute these two probabilities:

- *Computing $Pr[S_{ij} = d]$.* Since S_{ij} 's domain is $D_i \cup D_j$, we may use the conditional probability definition to compute the probability of $S_{ij} = d$.

When $d \in D_i$:

$$Pr[S_{ij} = d] = Pr[N_i = d] \times Pr[N_j > N_i | N_i = d] \quad (8)$$

When $d \in D_j$:

$$Pr[S_{ij} = d] = Pr[N_j = d] \times Pr[N_i > N_j | N_j = d] \quad (9)$$

For each $d \in D_i \cup D_j$, we can use Equation 4 and 5 and Equation 8 and 9 to compute $Pr[S_{ij} = d]$.

- *Computing $Pr[N_l > S_{ij} | S_{ij} = d]$.* This conditional probability can be computed as we did in Equation 5. Given point l 's sorted vector of neighbor distances $D_l = \langle d_1, d_2, \dots, d_{n-1} \rangle$, suppose that d_{min_2} is the smallest distance in this array that is larger than d , then by making use of Equation 4:

$$Pr[N_l > S_{ij} | S_{ij} = d] = \sum_{q=min_2}^{n-1} Pr[N_l = d_q] \quad (10)$$

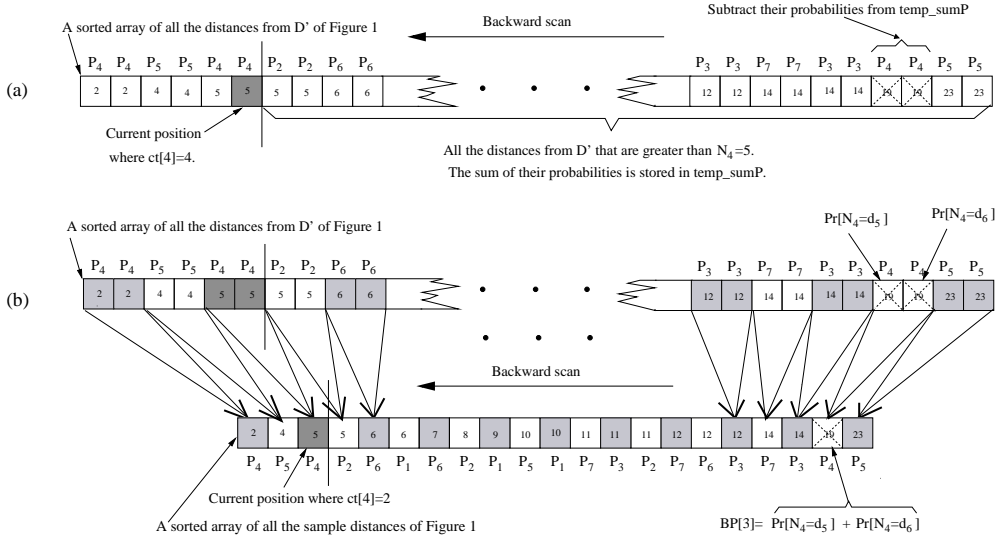


Figure 2: Computing $E[T_4]$ using the D' matrix depicted in Figure 1. (a) Computing $\sum_{j,j \neq 4} Pr[N_j > N_4 | N_4 = 5]$ (b) Scanning the sorted distances in D' is equivalent to scanning the sorted array of the sampled distances, as long as the probability of observing the sampled distance d is the sum of the probabilities associated with the $\frac{n-1}{\alpha}$ replications of d in D' . In the sorting, when there is a tie, higher priority is given to the point with smaller index.

4. SPEEDING UP THE COMPUTATION

The analysis of Section 3 exactly characterizes the statistical properties of the sampling algorithm. Equipped with this theoretical foundation, we now return to the two-step process described in Section 2 for evaluating the accuracy of the sampling algorithm: First, we logically construct a full-size distance database D' , which is treated as a reasonable approximation of the original neighbor distance database D . Next, we perform the exact statistical analysis of the sampling algorithm for D' , and then the result is returned to the user as an indication of the error of the sampling algorithm when it is applied to D . In this section, we describe these two steps in detail, and pay particular attention to the issue of speeding up the second step.

4.1 Constructing the Distance Database D'

Since we wish to avoid new distance computations in the construction of D' , one strategy is to uniformly replicate the distances computed by the sampling algorithm to create a full-size distance database that can subsequently be analyzed. We describe such a construction with an example, illustrated in Figure 1. On the left-hand side of Figure 1, a two-dimensional matrix is used to represent the distance database, where each column of the matrix contains the sorted distances from a given point to all of the other data points. Given this data, our sampling algorithm randomly selects three neighbor distances for each point of the input data set. As a result, we obtain the sampled distance matrix shown in the middle of Figure 1. In order to build a complete distance matrix D' by making use of the sampled distance matrix, we replicate each row of the sampled distance matrix twice, which gives us the approximate distance matrix D' shown on the right-hand side of Figure 1. Note that no additional distance computations are performed.

In general, the process of constructing D' uses $\frac{n-1}{\alpha}$ replications of each distance computed by the original sampling

algorithm. As a result, we regard each column D'_i in D' as an approximation of its corresponding column D_i in D . For ease of analysis, we assume that $\frac{n-1}{\alpha}$ is an integer in the remainder of this paper. It is an easy matter to generalize the analysis to the non-integer case.

4.2 Why Not Use D' Directly?

Given this simple construction, the next step is to apply the analysis of the previous section to D' . Though we now have the complete neighbor distance database D' , it turns out that directly applying Equation 2 and 6 to D' in order to estimate the mean and variance in the number of “real” outliers returned by the sampling algorithm will be unacceptably slow. The reason is that a direct implementation of Equation 2 will yield a triply-nested loop with complexity $O(n^3)$, where the outermost loop is for the summation of n ($y_i E[M_i]$)s, and the inner nested loop is due to the double summations in the $E[T_i]$ and $Var(T_i)$ formulas in Lemma 1. The situation becomes even worse when we use Equation 6 to compute the variance of N ; the direct implementation here will yield a quadruply-nested loop.

4.3 Efficiently Computing $E[N]$ Over D'

Since applying the naive $E[N]$ algorithm to D' will be unacceptably slow, we face the question: What can be done to alleviate this substantial performance penalty, in order to make the statistical analysis usable for a real-life data sets? In this section, we address the problem of speeding up the computation of $E[N]$ for D' . We show an efficient algorithm that allows the computation of $E[N]$ for D' in $\Theta(\alpha n \log(\alpha n))$ time. This algorithm is fast enough that the most expensive aspect of the computation is simply sorting the αn distances computed by the original sampling algorithm. In order to bring down the time complexity of computing $E[N]$, the first step is to reduce the size of the outermost loop of the naive $E[N]$ algorithm described in Section 4.2. Recall that y_i eval-

Algorithm 2 $EN(BP, BCP, BCPP, M, SortedSD)$

```
1: Let  $ET[1..\gamma]=VarT[1..\gamma]=sum1[1..\gamma]=sum2[1..\gamma]=\{0..0\}$ 
2: Let  $temp\_sumP=temp\_sumBCPP=tot=0$ ;  $ct[1..n]=\{\alpha..\alpha\}$ 
3: for  $d=\alpha n$  To 1 /*backward linear scan*/ do
4:    $col=SortedSD[d].cid$ 
5:    $Tid=M.find(col)$ 
6:   if  $Tid > 0$  /*following Lemma 1*/ then
7:      $ET[Tid]+=BP[ct[col]] \times (temp\_sumP - BCP[ct[col]+1])$ 
8:      $sum1[Tid]+=BP[ct[col]] \times (temp\_sumP - BCP[ct[col]+1])^2$ 
9:      $sum2[Tid]+=BP[ct[col]] \times (temp\_sumBCPP -$ 
10:     $BCPP[ct[col]+1])$ 
11:   end if
12:    $temp\_sumP+=BP[ct[col]]$ 
13:    $temp\_sumBCPP+=BCPP[ct[col]]-BCPP[ct[col]+1]$ 
14:    $ct[col]--$ 
15: end for
16: for  $Tid=1$  To  $\gamma$  do
17:    $VarT[Tid]=ET[Tid] - (ET[Tid])^2 + (sum1[Tid] -$ 
18:    $sum2[Tid])$  /*following Lemma 1*/
19:    $tot+=\Phi(\frac{\gamma-1-ET[Tid]}{\sqrt{VarT[Tid]}})$  /*Equation 2 and 7*/
20: end for
21: Return  $tot$ 
```

uates to one if point i in G is an outlier and zero otherwise. Observe that when y_i is zero, the term $y_i E[M_i]$ in Equation 2 contributes nothing to the final $E[N]$. Therefore, by only considering the $(y_i E[M_i])$ s corresponding to the “outlier columns” (those with the γ largest $D'_i[k]$ values) in D' , we can decrease the complexity of the outermost loop from $O(n)$ to $O(\gamma)$.

While reducing the complexity of the outermost loop of the naive $E[N]$ algorithm is straightforward, the task of reducing the size of the inner doubly-nested loop requires much more work, and we consider this problem presently.

4.3.1 Speeding Up the Double Summation

At first glance, it might appear that reducing the complexity of the inner doubly-nested loop is impossible, since with a given i we cannot avoid computing any terms in the double summations suggested by Lemma 1. To describe why it is possible to reduce the complexity, we start by showing an obvious way to calculate the double summations from Lemma 1. Then in the next subsection, we will demonstrate how to speed up the proposed method by exploiting the redundancy in D' .

In order to calculate the double summations from Lemma 1, the following two observations are critical:

1. Each distance $d \in D'$ is associated with a fixed probability $Pr[N_i = d]$ which is determined by d 's row position in D' .
2. The main task of computing $\sum_{j,j \neq i} Pr[N_j > N_i | N_i = d]$ and $\sum_{j,j \neq i} Pr^2[N_j > N_i | N_i = d]$ is nothing but finding out those distances that are larger than d and from a given column other than column i .

Thus, once we have a list of the distances described in observation 2 above, computing the summation $\sum_{j,j \neq i} Pr[N_j > N_i | N_i = d]$ is as simple as summing all the probabilities associated with those distances. Similarly, computing the summation $\sum_{j,j \neq i} Pr^2[N_j > N_i | N_i = d]$ is as simple as summing the squares of all the probabilities associated with those distances. Given a distance d from column i , the most obvious way to find all the distances in D' that are

Algorithm 3 $VarN(BP, BCP, BCPP, M, SortedSD, tot)$

```
1: Let  $EU[1..\frac{\gamma(\gamma-1)}{2}]=VarU[1..\frac{\gamma(\gamma-1)}{2}]=sum3[1..\frac{\gamma(\gamma-1)}{2}]$ 
2:    $=sum4[1..\frac{\gamma(\gamma-1)}{2}]=\{0..0\}$ 
3: Let  $temp\_sumP=temp\_sumBCPP=tot2=0$ 
4: Let  $ct[1..n]=\{\alpha..\alpha\}$ 
5: Let  $O[1..\gamma]$  stores the  $\gamma$  outlier column ids
6: for  $d=\alpha n$  To 1 /*backward linear scan*/ do
7:    $col=SortedSD[d].cid$ 
8:    $i=M.find(col)$ 
9:   if  $i > 0$  /*col indexes an outlier column*/ then
10:    for  $j=1$  To  $\gamma$  do
11:       $Uid=M.find2(col, O[j])$ 
12:      if  $Uid > 0$  /*following Lemma 2*/ then
13:         $Prob\_S_{ij}=BP[ct[col]] \times BCP[ct[O[j]]+1]$ 
14:         $EU[Uid]+=Prob\_S_{ij} \times (temp\_sumP - BCP[ct[O[j]]+1] -$ 
15:         $BCPP[ct[O[j]]+1])$ 
16:         $sum3[Uid]+=Prob\_S_{ij} \times (temp\_sumP -$ 
17:         $BCPP[ct[O[j]]+1] - BCPP[ct[O[j]]+1])^2$ 
18:         $sum4[Uid]+=Prob\_S_{ij} \times (temp\_sumBCPP -$ 
19:         $BCPP[ct[O[j]]+1] - BCPP[ct[O[j]]+1])$ 
20:      end if
21:    end for
22:     $temp\_sumP+=BP[ct[col]]$ 
23:     $temp\_sumBCPP+=BCPP[ct[col]]-BCPP[ct[col]+1]$ 
24:     $ct[col]--$ 
25:  end for
26: for  $Uid=1$  To  $\frac{\gamma(\gamma-1)}{2}$  do
27:    $VarU[Uid]=EU[Uid] - (EU[Uid])^2 + (sum3[Uid] -$ 
28:    $sum4[Uid])$  /*following Lemma 2*/
29:    $tot2+=2 \times \Phi(\frac{\gamma-2-EU[Uid]}{\sqrt{VarU[Uid]}})$  /*Equation 7*/
30: end for
31: Return  $(tot + tot2) - tot^2$  /*Equations 6 and 3.3*/
```

larger than d from a column other than column i is to simply sort all the distances in D' from the smallest to the largest, and then scan the sorted array from back to front until we reach d . By summing all of the probabilities associated with those distances that we pass which do not belong to column i , we can then obtain the desired sum. Since this technique allows us to compute $\sum_{j,j \neq i} Pr[N_j > N_i | N_i = d]$ and $\sum_{j,j \neq i} Pr^2[N_j > N_i | N_i = d]$ for any given d during a linear, backward scan of the sorted array, now we can compute $E[T_i]$ and $Var(T_i)$ by following Lemma 1 directly.

Example. Figure 2 (a) illustrates the idea. In this example, our current position is at $P_4 = 5$. We have been using $temp_sumP$ to store the sum of all the probabilities associated with the passed distances. Also, during the backward scan, we have been maintaining a counter $ct[col]$ for each column col of D' in such a way that it can always tell how many distances from column col remain to be scanned; we can also use this counter to tell how many distances from col have been passed. Since currently $ct[4] = 4$, a simple calculation $temp_sumP - Pr[N_4 = d_5] - Pr[N_4 = d_6]$ will give the value of $\sum_{j,j \neq 4} Pr[N_j > N_4 | N_4 = 5]$. In addition, from $ct[4] = 4$, we can infer the current distance's selection probability $Pr[N_4 = 5] = Pr[N_4 = d_4]$. Following Lemma 1, we update $E[T_4]$ by adding $Pr[N_4 = 5] \times \sum_{j,j \neq 4} Pr[N_j > N_4 | N_4 = 5]$ to it. If we do this for each d from column four throughout the scanning process, we will obtain the final result of $E[T_4]$ at the end of the scan. A similar idea applies to the calculation of $Var(T_4)$. Using this technique, we are able to compute the outlier columns' $E[T_i]$ s and $Var(T_i)$ s

using a single pass through the sorted distance array.

4.3.2 Compressing the Sorted Distance Array

The above method and example show how to compute the $E[T_i]$ and $Var(T_i)$ values using a linear scan of the sorted distance array. Unfortunately, this alone does not change the quadratic cost of the inner, doubly-nested loop of the naive $E[N]$ algorithm. Since the size of the sorted distance array is $n \times (n-1)$, scanning this array and checking each distance still has a cost that is $O(n^2)$.

To address this problem, notice that the $\frac{n-1}{\alpha}$ distances replicated from the same sample distance occupy $\frac{n-1}{\alpha}$ consecutive positions in the sorted array. As a result, there is no reason to store and process all of these repeated distances. We can easily compress the sorted array as follows:

1. For each block of the $\frac{n-1}{\alpha}$ identical elements in the sorted distance array of D' , we compress it to one element. The value of the compressed element is the sampled distance from which the replications originate.
2. In the compressed distance array (which now contains exactly the set of distances selected by our sampling algorithm) each element's associated probability is the sum of all the probabilities associated with the $\frac{n-1}{\alpha}$ distances represented by that element.

The compression step is illustrated in Figure 2(b). Note that in the original distance array, when the $\frac{n-1}{\alpha}$ distances replicated from the same sample distance participate in any multiplications in Lemma 1, they share the same multiplier (the innermost summations in $E[T_i]$ and $Var(T_i)$ formulas). As a result, applying the backward-scan procedure on the compressed distance array to calculate $E[T_i]$ and $Var(T_i)$ will not change the final result. Furthermore, since the linear scan of the sample distance array requires $\Theta(\alpha n)$ time, we can conclude that the total time complexity for computing the γ $E[T_i]$ s and $Var(T_i)$ s for the outlier columns has been brought down to $\Theta(\alpha n \log(\alpha n))$, which is simply the time required to sort all of the sampled distances.

4.3.3 Formal Algorithm Description

We now formally give the algorithm for approximating the average number of correct outliers from the set of sample distances produced in the original sampling algorithm. Before we begin, note that since now we treat each sampled distances as a representative of all its $\frac{n-1}{\alpha}$ replications in D' , it is beneficial to pre-compute the following probabilities so that we can use them directly in Algorithm 2 and 3:

- If we sort point i 's α sample distances, Block Probability (BP) is an array in which each element $BP[m]$ ($m \in \{1, 2, \dots, \alpha\}$) is the selection probability associated with point i 's m th smallest sample distance in our compressed distance array. Let $stride = \frac{n-1}{\alpha}$. Making use of Equation 4:

$$BP[m] = \sum_{q=(m-1) \times stride + 1}^{m \times stride} Pr[N_i = d_q] \quad (11)$$

- Block Conditional Probability (BCP) is an array in which each element $BCP[m]$ is the conditional probability of $N_j > N_i$ given N_i equals to one particular

sample distance. Let $BCP[\alpha + 1] = 0$. Making use of Equation 11:

$$BCP[m] = BP[m] + BCP[m + 1] \quad (12)$$

- Let $BCPP[m]$ store the square of $BCP[m]$. The $BCPP$ array is pre-computed because it will be used many times in the linear scan.

Given these notations, Algorithm 2 (subsequently referred to as “the EN algorithm”) formally describes how to efficiently compute $E[N]$. In the EN algorithm, a unique identifier Tid from $\{1, 2, \dots, \gamma\}$ is used to identify the set of variables associated with a particular outlier column, and $ET[Tid]$ and $VarT[Tid]$ are used to store the expectation and variance of T_{Tid} , respectively. Also, $ct[col]$ refers to the counter maintained for column col of D' and $SortedSD$ is the sorted array of all the sample distances. The EN algorithm also makes use of M , a data structure that stores the one-to-one mapping between Tid and the γ outlier column ids. M can be used to determine in constant time if a given column id col indexes an outlier column, If col does index an outlier column, M returns the Tid that corresponds to col . Otherwise, M returns a -1 .

4.4 Computing $Var(N)$ Over D'

Now that we have developed an efficient algorithm for computing $E[N]$, the next issue is how to speed up the computation for $Var(N)$. By observing that Lemma 2 and Lemma 1 have the same summation structure, we can speed up the naive $Var(N)$ algorithm in a similar manner as we did to the naive $E[N]$ algorithm. In this section, we show an efficient algorithm that allows the computation of $Var(N)$ for D' in $\Theta(\alpha n \log(\alpha n))$ time.

Recall that the naive $Var(N)$ algorithm requires a quadruply-nested loop. In order to bring down the time complexity of computing $Var(N)$, we can apply the same strategy as we did in section 4.3 to restrict our attention to the outlier columns. This time, we only consider the $(y_i y_j E[M_i M_j])$ terms corresponding to the pairwise combinations of the outlier columns in D' . As a result, the complexity of the outermost doubly-nested loop (the double summations in Equation 6) is reduced from $O(n^2)$ to $O(\frac{\gamma(\gamma-1)}{2})$. Here we take into account the fact that $y_i y_j E[M_i M_j] = y_j y_i E[M_j M_i]$.

By comparing Lemma 2 and Lemma 1, it soon becomes apparent that we can also do the backward, linear scan of the sorted sample distance array to compute $E[U_{ij}]$ and $Var(U_{ij})$. However, we need two modifications:

1. Upon seeing the current distance d belonging to an outlier column i during the scan, in addition to computing $Pr[N_i = d]$, for each other outlier column j , we also calculate $Pr[S_{ij} = d] = BP[ct[i]] \times BCP[ct[j] + 1]$, which makes use of Equations 8 and 9.
2. When computing $\sum_{l, l \neq i, j} Pr[N_l > S_{ij} | S_{ij} = d]$ and $\sum_{l, l \neq i, j} Pr^2[N_l > S_{ij} | S_{ij} = d]$, we take into account both column i and column j .

With the above two modifications, Algorithm 3 (subsequently referred to as “the $VarN$ algorithm”) formally describes how to efficiently compute $VarN$ by a backward linear scan of the sorted sample distances. In the $VarN$ algorithm, a unique id Uid from $\{1, 2, \dots, \frac{\gamma(\gamma-1)}{2}\}$ is used

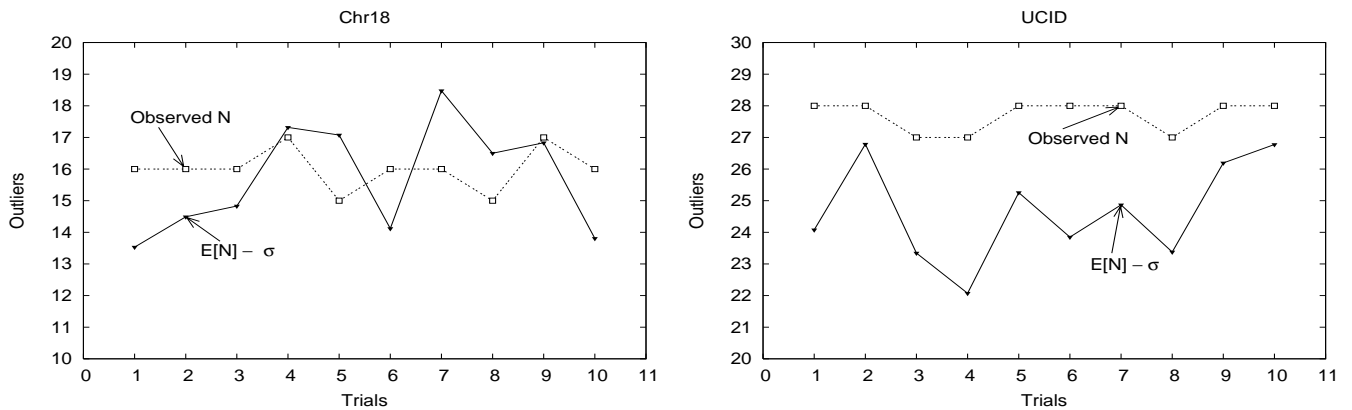


Figure 3: 10 trials of the sampling algorithm on the Chr18 and the UCID data set respectively. The square points mark the observed N in each trial. The triangle points are $E[N] - \sigma$ which are calculated based on the results of the EN and the $VarN$ algorithms.

to identify the set of variables associated with an unique pairwise combination of the outlier columns (here we treat $(i, j) = (j, i)$), and $EU[Uid]$ and $VarU[Uid]$ are used for storing the expectation and variance of U_{Uid} . Also, $ct[col]$, $temp_sumP$ and $temp_sumBCPP$ are maintained (line 19-21) exactly as we did in the EN algorithm. In the $VarN$ algorithm, it is assumed that the $find2(i, j)$ procedure of M (the map data structure from the EN algorithm) can always return the Uid corresponding to the combination of i and j , given i and j are two different outlier column ids. In other cases, $M.find2(i, j)$ returns a -1 .

5. EMPIRICAL EVALUATION

This Section details the results of two sets of experiments designed to test our methods:

1. First, we test the performance of our algorithms in two domains that require expensive distance computations by running fully-functional implementations of the paper’s algorithms over a bioinformatics sequence data set and an image histogram data set.
2. Second, we test the reliability of our estimations for the return set quality by repeating the algorithms ten times for various sample sizes over ten real data sets.

5.1 Testing Expensive Distance Functions

The first set of experiments has two goals. First, we wish to test whether our algorithms are able to return a high-quality answer set in an acceptably short time in a domain requiring an expensive distance function. Second, we wish to check whether a representative, state-of-the-art algorithm for this purpose (Bay’s nested loop algorithm [2]) can also provide acceptable performance.

To accomplish this goal, we considered two problems sharing the characteristics that:

1. Detecting DB-outliers from the data sets is a meaningful task in the selected domain.
2. The domains require expensive distance functions.

The first task we consider is detecting outlier sequences in human chromosome 18. Each data point in this data set is a nucleotide sequence. Detecting DB-outliers in such a data set is an interesting task, because it is well-established that comparisons of related protein and nucleotide sequences have facilitated many recent advances in understanding the information content and function of genetic sequences. The second data set we consider is a benchmark data set for content based image retrieval. Each data point in this data set is a color histogram derived from an individual image. In this case, the task is to discover the images that are least like the other images in the data set.

We now describe the characteristics of both data sets.

- Human chromosome 18 (Chr18). This data set is created by randomly selecting 4000 non-overlapping subsequences of length 2000 from human chromosome 18 downloaded from NCBI ftp site. Edit distance is used to measure the distance between two subsequences. The Edit distance between two subsequences is computed by the Needleman-Wunsch algorithm [10] which requires $\Theta(3 \times 2000^2)$ number of arithmetic computations per distance calculation in this case.
- UCID. UCID version 2 [15] contains 1338 uncompressed color images. We transform each image to a 576 dimensional color histogram and then normalize each dimension to the range $[0,1]$. HSV color space is used because it provides a breakdown of color into its most natural components: Hue, Saturation and Value (intensity). We uniformly divide hues into 36 buckets, saturation into 4 buckets, and value into 4 buckets. As a result, 576 color buckets are used to represent an image. Given histogram h_1 and h_2 , the distance between them is computed by the quadratic distance function $(h_1 - h_2)^T A (h_1 - h_2)$. The entry a_{ij} in the color similarity matrix A is determined by Equation 3 given in Section 3.1 of the VisualSeek paper [16].

The experiments were performed on a Linux workstation having an Intel Xeon 2.4GHz Processor with 1GB of RAM. All the algorithms were implemented in C++ and compiled

Algorithm	Chr18		UCID	
	Distance computations/point	Time	Distance computations/point	Time
Bay’s algorithm	331	24h:9m:31s	180	3h:48m:26s
Sampling algorithm	10	47m:24s	10	14m:7s
Ratios	33.1	30.6	18	16.2

Table 1: Efficiency comparisons between Bay’s algorithm and the sample algorithm on the Chr18 and the UCID data sets. The time reported for the sampling algorithm includes the running time of both the sampling algorithm and the EN and VarN algorithms.

Data set	$\alpha = 10$			$\alpha = 60$			$\alpha = 110$		
	observed N	$E[N]$	σ	observed N	$E[N]$	σ	observed N	$E[N]$	σ
Wisconsin cancer	16.60	13.29	3.55	24.60	22.20	0.00	25.60	23.12	0.00
Balance scale	8.60	5.17	5.29	8.60	6.46	5.75	7.60	5.70	5.77
Florida farm	27.10	27.09	0.27	28.20	26.90	0.86	28.10	26.48	0.71
California farm	22.70	24.59	0.73	24.40	25.15	0.00	25.60	25.12	0.00
FCAT Read	15.90	13.86	3.75	19.90	18.48	1.43	19.50	17.76	2.28
FCAT Math	12.90	12.69	4.40	18.70	17.14	2.11	18.50	17.94	1.46
California house	10.40	14.30	5.09	11.30	17.94	3.00	12.10	18.40	2.86
Baseball pitching	14.70	17.04	2.32	14.70	19.01	1.32	14.10	19.05	0.00
Baseball batting	6.10	4.19	5.57	8.50	7.30	5.99	11.40	9.67	5.42
Cover Type	0.10	3.24	6.34	0.00	2.98	5.17	0.00	2.67	5.30

Table 2: The average estimation results and the empirical results of N for 10 trials on the 10 real data sets. The sample sizes α is set to 10, 60 and 110 respectively.

with gcc version 4.02. Since we were interested in testing the scenarios that the distance computations dominate all the other costs, we load the entire data set into memory for all the algorithms that we tested. We report the wall clock time here. All experiments were run to return the top 30 5th-NN outliers.

We began our experiments by running Bay’s nested loop algorithm over both data sets. We recorded the time required, as well as the average distance computations per data point. The results are reported in Table 1. Next, we ran our implementations of the sampling algorithm and the EN and $VarN$ algorithms over the same data, and then intersected the result set of our sampling algorithm with the result set of Bay’s algorithm to get the observed N . For each data set, we ran 10 trials with the sample size α set to 10. The performance results are also shown in Table 1.

In addition, we plot the bound calculated by $E[N] - \sigma$, where $E[N]$ is the estimate of the expected value of N , and σ (the standard deviation of N) is the square root of the estimate of the variance of N . Using expectation and standard deviation to predict the “typical” observation is standard practice in statistics. The analytically-estimated standard error and the observed N for both of these runs are plotted in Figure 3.

Discussion. Even though the two data sets have just a few thousand points, Bay’s algorithm had relatively poor performance on both, taking nearly one day for the first one. We note that these results are actually fairly optimistic due to experimental time constraints. Both data sets tested were fairly small. Supposing that the subsequences in the Chr18 data set were of length 8000 instead of 2000, the cost of the Edit distance function would be increased by 16 times. Assuming that Bay’s algorithm had the same pruning power

Data Set	Cont./Feature	Size	BA
Wisconsin cancer	30/31	569	165
Balance scale	4/4	625	625
Florida farm	188/188	811	140
California farm	188/188	1,373	175
FCAT Read	27/27	1,404	275
FCAT Math	28/28	1,405	274
California house	9/9	20,640	654
Baseball pitching	22/22	36,245	454
Baseball batting	17/17	85,979	745
Cover Type	10/55	581,012	4527

Table 3: Description of the 10 real data sets and the average number of distance computations per data point (denoted by BA) by Bay’s algorithm. Cont./Feature means the number of continuous features over the number of total features.

for the longer sequences, we would expect more than two weeks’ running time to complete the computation over the Chr18 data set using the full dimensionality.

In contrast, our algorithms did an excellent job in these two domains. Our algorithms provided between one and two orders of magnitude speedup, and still maintained relatively high result quality. The sampling algorithm returned more than half of the total true 5th-NN outliers in both cases (and nearly all of them in the UCID case) using only ten samples per data set point. Furthermore, in seven of the ten trials over the Chr18 data set and in ten of the ten trials over the UCID data set, the actual number of outliers returned either exceeded or was almost equal to $E[N] - \sigma$. This indicates that (at least for these two data sets), the reported $E[N] - \sigma$

constitutes a safe lower bound on qualitative performance.

5.2 Reliability of the Estimation

This subsection describes an additional experimental evaluation of our algorithms, aimed at evaluating the reliability of our estimation algorithms. We wish to obtain some experimental evidence that the results given by our estimation algorithms are reliable with various sample sizes and different sizes of data sets.

The test was designed as follows. We selected 10 real data sets summarized in Table 3. They span a range of problems and have very different characteristics. Due to the space limit, we do not describe each in detail. The experimental setup was identical to what we described in the previous subsection. We processed the data by normalizing all continuous variables to the range $[0,1]$ and converting all categorical variables to an integer representation. Hamming distance was used for categorical features and Euclidean distance was used for continuous features. For each data set, before running our algorithms we experimented Bay's algorithm on the data set. The average number of distance computations per data point required by Bay's algorithm are reported in the last column of Table 3. Note that we happened to encounter the worst case scenario ($O(n^2)$ time complexity) using Bay's algorithm on the Balance scale data set from UCI Machine Learning Repository. That is, the average number of distance computations per data point in this data set was the size of the data set. For each of the 10 real data sets, we systematically tested the paper's algorithms with the sample size α set to 10, 60 and 110 respectively. For each experiment, we performed 10 trials. The average statistics of the 10 trials are reported in Table 2.

Discussion. We observed high reliability for our estimation algorithms, indicating that the constructed distance database D' is a reasonable surrogate for the original distance database D . Specifically, for the 300 total runs that were performed in order to construct Table 2, the actual number of true 5th-NN outliers returned was larger than $E[N]$ 199/300 times, larger than $(E[N] - \sigma)$ 236/300 times, and larger than $(E[N] - 2\sigma)$ 253/300 times. The table depicts in detail the close correlation between the predicted $E[N]$ and the observed average N . Again, this shows the general utility of our analysis for predicting the accuracy of the algorithm.

6. CONCLUSIONS

In our work, we have considered the problem of how to efficiently detect DB-outliers when the distance function is expensive. A simple sampling algorithm is proposed and the statistical characteristics of this sampling algorithm is formally analyzed. Based on the analysis, we have developed highly efficient and scalable estimation algorithms, whose efficiency and reliability in evaluating the quality of the sampling algorithm's return set have been verified by our extensive real case studies. As a result, this paper's algorithms make it practical to detect DB-outliers in expensive domains.

7. REFERENCES

- [1] F. Angiulli and C. Pizzuti. Outlier detection for high dimensional data. In *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases*, August 2002.
- [2] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 29–38, 2003.
- [3] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: identifying density-based local outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 93–104, May 2000.
- [4] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, August 2004.
- [5] M. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.
- [6] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.
- [7] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal: Very Large Database*, 8(3-4):237–253, February 2000.
- [8] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large datasets. *IEEE Transactions on Knowledge and Data Engineering*, 15(5), 2003.
- [9] E. Lehmann. *Elements of Large-Sample Theory*. Springer, 1998.
- [10] S. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
- [11] B. Presnell and J. Booth. Resampling methods for sample surveys. *Presnell, B. and Booth, J.G. (1994) Resampling methods for sample surveys. Technical Report 470, Department of Statistics, University of Florida, Gainesville, FL*, 1994.
- [12] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 427–438, May 2000.
- [13] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(5), 1998.
- [14] C.-E. Sarndal, B. Swensson, and J. Wretman. *Model Assisted Survey Sampling*. Springer-Verlag, 1992.
- [15] G. Schaefer and M. Stich. Ucid - an uncompressed colour image database. In *Proceedings of SPIE, Storage and Retrieval Methods and Applications for Multimedia 2004*, pages 472–480, 2004.
- [16] J. R. Smith and S.-F. Chang. Visualeek: A fully automated content-based image query system. In *Proceedings of the Forth ACM International Conference on Multimedia '96*, pages 87–98, November 1996.