Mining Deviants in a Time Series Database

H. V. Jagadish
University of Michigan
Ann Arbor, Michigan

Nick Koudas
AT&T Laboratories
Florham Park, NJ 07932

S. Muthukrishnan
AT&T Laboratories
Florham Park, NJ 07932

Abstract

Identifying outliers is an important data analysis function. Statisticians have long studied techniques to identify outliers in a data set in the context of fitting the data to some model. In the case of time series data, the situation is more murky. For instance, the "typical" value could "drift" up or down over time, so the extrema may not necessarily be interesting. We wish to identify data points that are somehow anomalous or "surprising".

We formally define the notion of a deviant in a time series, based on a representation sparsity metric. We develop an efficient algorithm to identify deviants in a time series. We demonstrate how this technique can be used to locate interesting artifacts in time series data, and present experimental evidence of the value of our technique.

As a side benefit, our algorithm are able to produce histogram representations of data, that have substantially lower error than "optimal histograms" for the same total storage, including both histogram buckets and the deviants stored separately. This is of independent interest for selectivity estimation.

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

1 Introduction

Outlier detection has a long history [Cha84] in statistics, and more recently, in data mining. However, the bulk of this work has been concerned with finding outliers in a large set of data, most often with respect to some parametric model in mind. When one deals with time series data, it is not appropriate to ignore the time axis and think of the observed values as an unordered set. As such, standard outlier detection techniques do not carry over easily [KN98].

Consider the data series shown in Figure 1. Point 11 "sticks out" as having a much larger value than neighboring points, and hence may be a deviant worthy of further analysis. Point 4 has a larger value than point 11, but is part of a group (points 4-6) of large-valued points and so is not an interesting outlier (or deviant) in itself[1].

The question then becomes how do we identify points with values that differ greatly from that of surrounding points. Do we consider immediate neighbors only? Do we consider points further away with some

[1] It is reasonable to consider the whole subsequence of points 4-6 for further analysis, since they all have value much higher than typical, and indeed our formulation below will permit this.

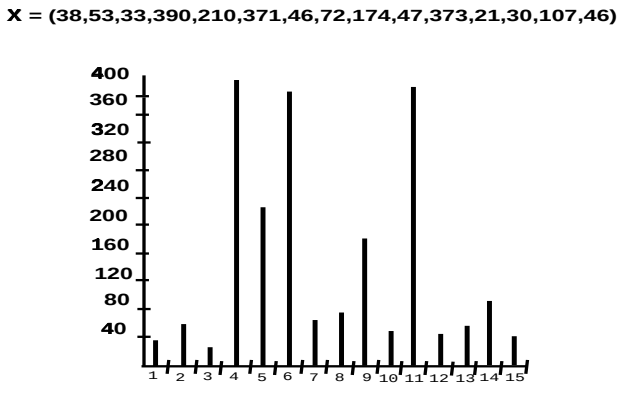


Figure 1: Example Sequence

sort of decreasing weight for distance? How do we deal with cases where a point differs greatly from its left neighbor, but less so from its right neighbor? One can come up with a variety of ad hoc local definitions, each of which gives a different answer, and none of which is conceptually satisfying.

Our proposal is to use the well recognized information theoretic principle of representation length. If the removal of a point P from the time sequence results in a sequence that can be represented more succinctly than the original one (by more than the increment required for explicitly keeping track of P separately), then the point P is a deviant. For succinct, (lossy), representation of series data, we adopt the histogram method, so popular in database literature and practice.

Thus, our problem is to find points in a given time series whose removal from the series results in a histogram representation with lower error than the original, even after the number of buckets has been reduced to account for the separate storage of these deviant points.

The solution to this problem is interesting in itself, and is the focus of the first half of this paper. We show in Section 5, that such identification of deviants can lead to substantial increases in the accuracy of histogramming techniques for the same storage, and only a small additional computational cost. Efficient algorithms for this purpose are developed in Section 4. The proposed algorithms are fast and optimal.

We discuss how to analyze deviants found using these techniques in Section 6. In particular, we develop a simple algorithm to find clusters of deviants. In Section 7, we present a series of experiments performed on real life time series databases, and highlight the utility of the deviant mining algorithms.

We begin with some background and our assumptions in Section 2, followed by the formal definition of a deviant as well as our deviant-finding problem in Section 3.

2 Background

Histogramming techniques are of profound importance in query optimization in relational databases. Accurate estimates of result size of queries are crucial for the choice of execution plan in a relational query optimizer. Histogramming is also invaluable for approximate query answering. There exists a sizeable volume of research on histogramming techniques [IP95, Ioa93, PIHS96, PI97, JKM+98]. We follow these works to define the problem in mathematical terms, and develop a framework within which a solution with general applicability can be described.

We are given a sequence of data points, $X = v_1 v_2 \cdots v_N$ where each v_i, $1 \leq i \leq N$, is an integer drawn from some bounded range. Let S be the set of indices of points in X, thus $S = \{1, 2, \ldots, N\}$. There is great flexibility to the semantics of X. X could represent the sequence of values obtained from the observation of some random variable, i.e., v_i is the value of the random variable at time i. Thus X could represent a stock price over time, the usage (in some time granularity) of an AT&T service, etc. In a different context such as that of query result size estimation, X could represent a frequency vector of an attribute in some relational instance. In this case, i corresponds to an attribute value[2] and v_i to the total number of

[2] We assume that attribute values can be mapped to an integer domain. More general assumptions are possible, but these issues are not important for the main arguments in our paper, so we will assume that the i take values from a dense integer set.

occurrences of i in the attribute instance[3]. Let $D \subseteq S$ be a set containing indices of points in X. We denote as $X - D$ the sequence that results if we remove from X all points with indices in D.

Irrespective of the semantics of vector X, the objective is the same: given some space constraint B, create and store a compact representation of sequence X using at most B storage, denoted H_B, such that H_B is optimal under some notion of error $E_X(H_B)$ defined between X and H_B.

In the process of creating a compact representation for X, several issues arise. We first need to decide on a representation for H_B. Let s_i and e_i be indices of points in X such that $s_i \leq e_i$. The common choice is to collapse the values in the sequence of consecutive points (s_i, e_i) into a single value h_i (typically their average, $h_i = \sum_{j=s_i}^{e_i} v_j$). The value h_i along with (s_i, e_i) (recording the start and the end of the sequence thus collapsed) forms the *bucket* b_i, that is, $b_i = (s_i, e_i, h_i)$. Since h_i is only an approximation for the values in bucket b_i, whenever a bucket is formed an error is induced due to the approximation. We need to choose a way to quantify this error. As is common, we quantify this error using the *Sum Squared Error* (SSE) metric, defined as:

$$F(b_i) = \sum_{j=s_i}^{e_i} (v_j - h_i)^2 \qquad (1)$$

Assuming that the space constraint is expressed in buckets, the resulting approximation H_B will consist of B buckets and the total error in the approximation is $E_X(H_B) = \sum_{i=1}^{B} F(b_i)$.[4]

The optimal histogram construction problem is then defined as follows:

[3]Note in particular that the only property of X material to us is that it is an ordered set of values. In this paper, we have focused on time sequences, but all our results are directly applicable also to other ordered sets such as the frequency counts for an attribute value.

[4]Other error functions are possible such as $\max_i F(b_i)$ etc. All our results will hold for any point-wise additive error function $E_X(H_B)$, but we will focus on the specific error function above which is the most common one in the literature.

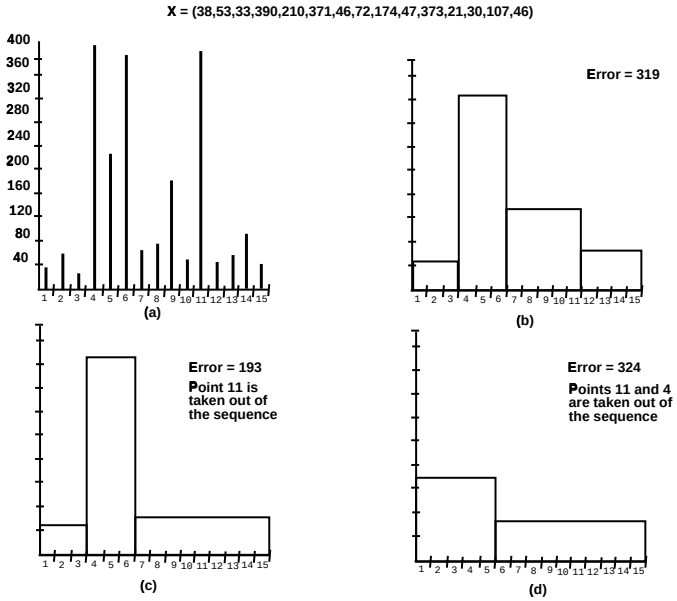


Figure 2: Histogram construction: (a) Original sequence (b) Optimal histogram with 4 buckets (c) Optimal histogram with 3 buckets and point $v_{11} = 373$ removed (d) Optimal histogram with 2 buckets and points $v_4 = 390$ and $v_{11} = 373$ removed.

Problem 1 (Optimal Histogram Construction) *Given a sequence X of length N, a number of buckets B, and an error function $E_X()$, find H_B to minimize $E_X(H_B)$.*

The resulting optimal histogram is the well known `vopt` histogram [IP95]. There exists an efficient algorithm to determine the `vopt` histogram [JKM+98]. This algorithm is based on dynamic programming and it chooses the non overlapping sequences of points that should be collapsed into single values (bucket boundaries), such that the total error metric is minimized.

3 Deviants: Motivation and Definitions

Consider Figure 2(a) which represents a portion of time series extracted from an AT&T data warehouse. Figure 2(b) presents the optimal histogram `vopt` on the sequence in Figure 2(a) using $B = 4$. The approximation in Figure 2(b) has total (sum of squares) error of 319. Assume that we are allowed to remove a point

from the sequence and store it separately. This requires two integers, one for the position in the sequence and one for the value of the point. The storage required for one histogram bucket is also two integers: the left boundary and the average value. (Only one boundary needs to be kept since buckets are non-overlapping and adjacent. Without loss of generality, we keep the left boundary.) Thus, we have spent the equivalent of one bucket from our total bucket budget, and we must approximate the remaining sequence with $B-1$ buckets, that is, 3 buckets in this case. One can observe that by removing the point $v_{11} = 373$, the total error of the approximation is reduced to 193 which is an improvement in accuracy of 40%. However, one should be careful about removing the points since not all points in the sequence of Figure 2(a) reduce the total error when they are removed. For example, by removing the point $v_1 = 38$, the total error in the approximation increases (recall that after removing v_1, only 3 buckets are available for the remaining sequence). In the case of a single point removal, one way to determine which point to remove in order to obtain the largest decrease in the total error would be to consider each point in turn and use `vopt` on the remaining sequence; there are a total of N `vopt` executions. Notice that the point causing the largest decrease in error (v_{11}) in this example is not the point that has the maximum value. We refer to points that have this property as *deviants*; these are formally defined below.

Now consider removing two points. In this case, the smallest error one can achieve by removing two points and approximating the resulting sequence with $B-2 = 2$ buckets is 324. It is achieved by removing points $v_4 = 390$ and $v_{11} = 373$. The total error however is larger than the one with $B = 4$ buckets, so removing two points is not beneficial. In order to determine the pair of points whose removal results in the largest decrease in the total error of approximation, if one were to consider each pair of points and apply `vopt` on the remaining sequence when that pair is removed, the total number of `vopt` executions required is $\binom{N}{2}$. In general, the total number of point sets to consider for removal of "best" k points is $\binom{N}{k}$. By trying to remove additional points, one can verify that no further improvement is possible, with the given bucket budget.

We now formally define the notion of deviants and the problem of finding the optimal ("best") deviants.

Definition 1 *Given a sequence of points X and a number of buckets B, a point set $D \subseteq S$ is a deviant set if and only if $E_X(H^*_B) > E_{X-D}(H^*_{B-|D|})$, where H^*_B is the optimal histogram on X with B buckets (having error $E_X(H^*_B)$) and $H^*_{B-|D|}$ is the optimal histogram on $X-D$ with $B-|D|$ buckets (having error $E_{X-D}(H^*_{B-|D|})$).*

From the preceding discussion, it follows that for a given sequence of points X and a given number of buckets B, there is an optimal deviant set that, if identified, can be used to construct a histogram representation with minimum total error. (In the example above, the optimal number of deviant points is one.) This observation leads to the following optimization problem:

Problem 2 (Optimal Deviant Set) *Given set X and buckets B, identify a set $D \subseteq S$ such that $\forall D', D' \subseteq S, E_{X-D}(H_{B-|D|}) \leq E_{X-D'}(H_{B-|D'|})$.*

4 Algorithms

We are interested in providing an efficient solution to Problem 2. A straightforward solution, that was described earlier, is to examine every possible point set in turn and to compute the optimal histogram on the remaining sequence after that point set has been removed. The computational cost for such an approach would be prohibitive, being exponential in the number of deviants considered. We give a different algorithm which is optimal, and is significantly more efficient. We

first develop a solution to the following related problem:

Problem 3 (Optimal κ Deviants) *Given sequence X, a budget of storage B and a non-negative integer κ, identify set $D \subseteq S$ with $|D| = \kappa$ such that $\forall D', D' \subseteq S$, $|D'| = \kappa$, we have*

$$E_{X-D}(H^*_{B-\kappa}) \leq E_{X-D'}(H^*_{B-\kappa}).$$

We are given a sequence X of numbers of total size N. We are also given parameters B, the total storage and κ, the number of deviants. Our goal is to produce a histogram consisting of $B-\kappa$ buckets and κ deviants, so that the total error of the histogram is minimized.

Let (s_i, e_i) denote the starting (leftmost) and ending (rightmost) points respectively in sequence X, of the points contained in bucket b_i. Since some points in the range (s_i, e_i) might be deviants, we have to redefine the error formula given in eqn. (1) for the error induced by the bucket, to account for the points (the deviants) that are removed from the sequence. For bucket $b_i = (s_i, e_i)$, say there is a set T_i of deviants in (s_i, e_i) and $|T_i| = k$. The refined SSE for bucket b_i with these k deviants is denoted $F^{T_i}(s_i \cdots e_i, k)$, and it is defined as:

$$F^{T_i}(s_i \cdots e_i, k) = \sum_{s_i \leq j \leq e_i \ \& j \notin T_i} (v_j - h^{T_i}_{s_i,e_i})^2 \qquad (2)$$

where $h^{T_i}_{s_i,e_i}$ is the average of non-deviants (values at indices not belonging to T_i) within the bucket, that is,

$$h^{T_i}_{s_i,e_i} = \frac{\sum_{j=s_i \ \& \ j \notin T_i}^{e_i} v_j}{e_i - s_i + 1 - k}.$$

The total error of the approximation represented by the histogram and a given set of deviants T is the sum over the buckets of the error in each bucket b_i when the set of deviants $T_i \subseteq T$ that fall within that bucket are removed. Our proposed solution for solving Problem 3 is based on dynamic programming. We first parameterize the error term that we wish to minimize. Let $E(i, j, k)$ represent the error of using j buckets on the sequence v_1 through v_i with k deviants been identified in $[1 \ldots i]$, in the optimal way (therefore, the error is the minimum possible). Using Bellman's equation [Bel54] we can write a recursive expression for $E(i, j, k)$:

$$E(i, j, k) = \min_{1 \leq l \leq i, \ 0 \leq m \leq k} E(l, j-1, m) + F^*(l+1 \cdots i, k-m). \qquad (3)$$

Here, $F^*(a \cdots b, c)$ denotes the minimum error in the bucket $(a \cdots b)$ with c deviants, that is,

$$F^*(a \cdots b, c) = \min_{T| \ T \in \{a, \ldots, b\}, \ \ |T| = c} F^T(a \cdots b, c) \qquad (4)$$

In other words, the equation for $E(i, j, k)$ calculates the error of the optimal strategy with k deviants and j buckets on the sequence $v_1 \cdots v_i$ to be the sum of the errors of the optimal strategy with m deviants and $j-1$ buckets on the sequence $v_1 \cdots v_l$, and that for a single bucket on the sequence $v_{l+1} \cdots v_i$ using $k-m$ (the remaining) deviants. This is appropriately minimized over the choices for the parameters m and l.

The initialization to form the basis of this bootstrapping approach comes from observing that $E(i, 1, m) = F^*(1 \cdots i, m)$, for $1 \leq i \leq N$ and $0 \leq m \leq \kappa$. The remainder of the algorithm uses equation (3) to construct solutions for successively larger number of buckets and number of deviants. After $E(i, j, k)$ has been constructed for $1 \leq i \leq N, 1 \leq j \leq B-\kappa, 0 \leq k \leq \kappa$, the algorithm terminates.

The only piece that still remains unspecified is the computation of $F^*(a \cdots b, c)$ for all possible values of a, b, c. If $c \geq b - a + 1$, clearly $F^*(a \cdots b, c) = 0$. We next show how to efficiently compute $F^*(a \cdots b, c)$ in all the other cases. Recall the definition of F^* from Equation 4, and say the minimum there is reached for the set T^*. (We do not know T^* apriori and it is used only for making the definitions formal.) Define $S^*[a, b, c]$ to be sum of the non-deviant points of the sequence $(v_a \ldots v_b)$ in the solution with error $F^*(a \cdots b, c)$, that is, $S^*[a, b, c] = \sum_{a \leq i \leq b, \ i \notin T^*} v_i$. Similarly, define $SS^*[a, b, c]$ to be the sum of the

squares of the non-deviant points in this sequence, that is, $SS^*[a,b,c] = \sum_{a \le i \le b,\, i \notin T^*} v_i^2$. We will again use dynamic programming to compute $S^*[a,b,c]$ and $SS^*[a,b,c]$, and use those to compute $F^*(a \cdots b, c)$ at the same time.

Computing $F^*(a \cdots b, c)$ has two parts, depending on whether b is a candidate deviant point or not. If b is a deviant point in the minimum error solution, we would have to update the parameters as follows:

$$F^*(a \cdots b, c) = F^*(a \cdots b-1, c-1)$$
$$S^*[a,b,c] = S^*[a,b-1,c-1]$$
$$SS^*[a,b,c] = SS^*[a,b-1,c-1]$$

Since b is a deviant point, it is not included in the computation of the error; b is extracted from the sequence. The other case is when b is not a deviant point. In this case, the solution $F^*(a \cdots b-1, c)$ must be adjusted to include v_b. Thus, S^* and SS^* must be updated as $S^*[a,b,c] = S^*[a,b-1,c] + v_b$ and $SS^*[a,b,c] = SS^*[a,b-1,c] + (v_b)^2$. The $F^*(a \cdots b, c)$ value must be updated using the formula obtained by simple algebraic manipulation:

$$F^*(a \cdots b, c) = SS^*[a,b,c] - \frac{S^*[a,b,c]^2}{(b-a+1-c)} \qquad (5)$$

We consider both these cases and calculate the two possible values of F^*. We choose the case that yields the smallest error F^* and the values of F^*, S^* and SS^* are updated as described above depending on the chosen case.

That completes the description of the algorithm *ComputeF** for computing F^* values. These values are stored in a table, which can be looked up whenever needed while computing $E(i,j,k)$'s. That also completes the description of the algorithm for computing $E()$ values. The value $E(N, B-\kappa, \kappa)$ gives the minimum error in the solution to Problem 3. As in all dynamic programming solutions, it is now a standard step to determine a set of κ deviants and the bucket boundaries of the histogram which result in $E(N, B-\kappa, \kappa)$ error of approximation.

Theorem 1 *There is an $O(N^2(B-\kappa)\kappa^2)$ algorithm for solving Problem 3.*

proof. We calculate $S^*[a \cdots b, c]$ for all $1 \le a \le N$, $a+1 \le b \le N$ and $0 \le k \le \kappa$; thus there are $O(N^2\kappa)$ values of $S^*[a \cdots b, c]$ of interest. There are two cases to consider for updating S^* each of which takes $O(1)$ time. Thus, the time taken to calculate all the $S^*[\,]$'s is $O(N^2\kappa)$; the same holds for the $SS^*[\,]$ and $F^*()$ values too. After this precomputation, the value of $F^*(a \cdots b, c)$ for any $1 \le a \le N$, $a \le b \le N$, $0 \le c \le \kappa$, can be retrieved in $O(1)$ time.

We calculate $E(i,j,k)$ for each $1 \le i \le N$, $1 \le j \le B-\kappa$ and $0 \le k \le \kappa$; hence, there are $O(N(B-\kappa)\kappa)$ values of $E(i,j,k)$ of interest in all. Computing each such value using Equation 3 involves considering $O(N\kappa)$ values of $E(l, j-1, m)$ which can be obtained in $O(1)$ time each using the dynamic programming technique, and $O(N\kappa)$ values of $F^*(l+1 \cdots i, k-m)$ which can be obtained in $O(1)$ time each using the precomputation. Thus each $E(i,j,k)$ takes time $O(N\kappa)$ to calculate and the total time taken is $O(N^2(B-\kappa)\kappa^2)$. ∎

Optimal Deviant Set

Using the solution for Problem 3 we can obtain an efficient solution for Problem 2. The simplest approach would be to solve Problem 3 with the number of deviants κ taking values $0, \ldots, B-1$ and the number of buckets taking values $B-\kappa$ for each such choice of κ. This will involve invoking our solution to Problem 3 at most B times. However, we can solve Problem 2 in time that is essentially the time it takes to solve an instance of Problem 3 *once* as follows.

Recall that the algorithm in the previous section finds $E(i,j,k)$ for all $1 \le i \le N$, $1 \le j \le B-\kappa$ and $0 \le k \le \kappa$ in time $O(N^2(B-\kappa)\kappa^2)$. We can extend the algorithm to compute all those values for $1 \le j \le B$ and $0 \le k \le B-1$ so the resulting algorithm takes time $O(N^2B^3)$. We can then read off the minimum error solution amongst $E(i,j,k)$ where $j = B-k$ for

all $0 \leq k \leq B-1$. This takes $O(B)$ additional time. Therefore we can conclude the following.

Theorem 2 *There is an $O(N^2B^3)$ time algorithm for solving Problem 2.*

5 Deviant Histogramming

In the previous section, we presented algorithms for finding the optimal κ deviants and for detecting the optimal deviant set in a given sequence X. In fact, our technique gives us a compact representation for X comprising the deviant points, as well as the histogram on the remainder. More precisely, we have κ deviants each stored as a pair of its position i in X and its value v_i, and $B-\kappa$ buckets each stored as a pair of its left endpoint in X and the average of the non-deviant values in that bucket.

This compact representation may be thought of as a histogram in itself, and it will serve the same purposes as a histogram. In this section, we explore its use in selectivity estimation. In particular, we focus on equality queries (other queries such as range queries may also be answered using our compact representation just as it is done with standard histograms). An estimate for an equality query i is obtained as follows. If i is a deviant, we merely return its value v_i and no error is incurred. If i is not a deviant, we determine the bucket to which i belongs and return the value stored with that bucket (recall that this is the average of the non-deviant points that lie within that bucket) as an estimate of v_i.

Here we report experimental results on a number of experiments performed to assess the utility of the proposed algorithm as a histogramming technique. The common method for evaluating selectivity estimation techniques on equality queries is to consider the sum-of-squares metric, namely, $\sum_i (v_i - e_i)^2$ where e_i is the estimate for the point query v_i [IP95]. This is the expected total error on a workload of equality queries if all equality queries are equally likely. Poosala et al [PIHS96] showed that the `vopt` histogram was the optimal histogram for estimating equality queries, provided that one uses a histogram representation without removing any points (deviants). We compare the `vopt` histogram to our histogram wherein our histogram uses the same space as the `vopt` histogram, but it uses a portion of it for storing the deviants and the remainder for the buckets.

We used real data sets extracted from an AT&T data warehouse in our experiments. All data sets consist of 2000 points in a time sequence. Because of proprietary reasons, we are not able to disclose the specific data sets used. However, we do number the data sets in order of increasing skew – data set D4 is close to a skewed (randomized) Zipfian whereas D1 is skewed very little.

In all our experiments, we keep the total space (expressed in buckets) devoted to the histogram fixed, and we vary the number of deviants. With total space B devoted to the histogram and k deviants, $B-k$ buckets are placed in the sequence. (Recall that it takes the same number of bytes to store one deviant point as it does to store a bucket.) The case when $k = 0$ is the well known `vopt` histogram.

Figure 3 shows the results for the four data sets, as k is varied, for a representative selection of three values for B, the total storage. For all data sets we see that the error decreases as more storage is devoted, as expected. We also see that identifying and storing deviants separately does help, and in some cases quite substantially. For instance, for data set D3, the error with a storage of 20 is approximately the same as the error with double the storage and no deviants; most of the space is devoted to the deviants in this case. In other words, just by identifying deviants and storing them separately, we are able to decrease error as if we had doubled the storage!

However, most of the curves exhibit a clear minimum indicating that there is a point beyond which devoting additional storage to deviants is not desirable. This minimum represents the "optimal" number of deviants for the specific data set and the storage budget.

As expected, the optimal number of deviants increases (linearly) with the storage budget. Also, more skewed is the distribution of data, greater is the benefit obtained from identifying deviants and larger is the number of deviants at the optimal point. In fact, for a highly skewed data set, D4, the minimum is beyond the right end of the feasible region, indicating that it is best for us to devote almost all of our storage budget to the deviants. (In this case, the optimal solution is an end-biased histogram where we keep the most deviant values rather than the extremal values.)

6 Data Mining with Deviants

Consider an analyst examining a large time series. Instead of manually examining the entire sequence, it would be beneficial to have it automatically tagged for regions (or points) of potential interest. The crucial issues are to define the notion of "interesting" regions in a time series, and to design efficient algorithms for finding such regions.

Deviants have an intriguing combination of local and global property. A purely global approach to finding interesting points in a time series may be to identify points farthest away from the mean of the entire sequence [AAR95]. This approach would find all the extremal points. In contrast, a purely local approach to finding interesting points may be to determine pairs of neighboring points that differ by the largest amount. The MaxDiff [IP95, GMP97] heuristic for histogram construction accomplishes this efficiently. For example, this technique will find all regions with the highest derivative, that is, the slope. While the points determined for the purely local, or the purely global approaches may be of interest, there are instances when one needs a more sophisticated notion of interesting points or regions. An optimal set of deviants combines the local and global aspects naturally: they are local in so far as trying to minimize the deviation from the average within a bucket, and global in trying to minimize the total sum of the errors from the different buckets.

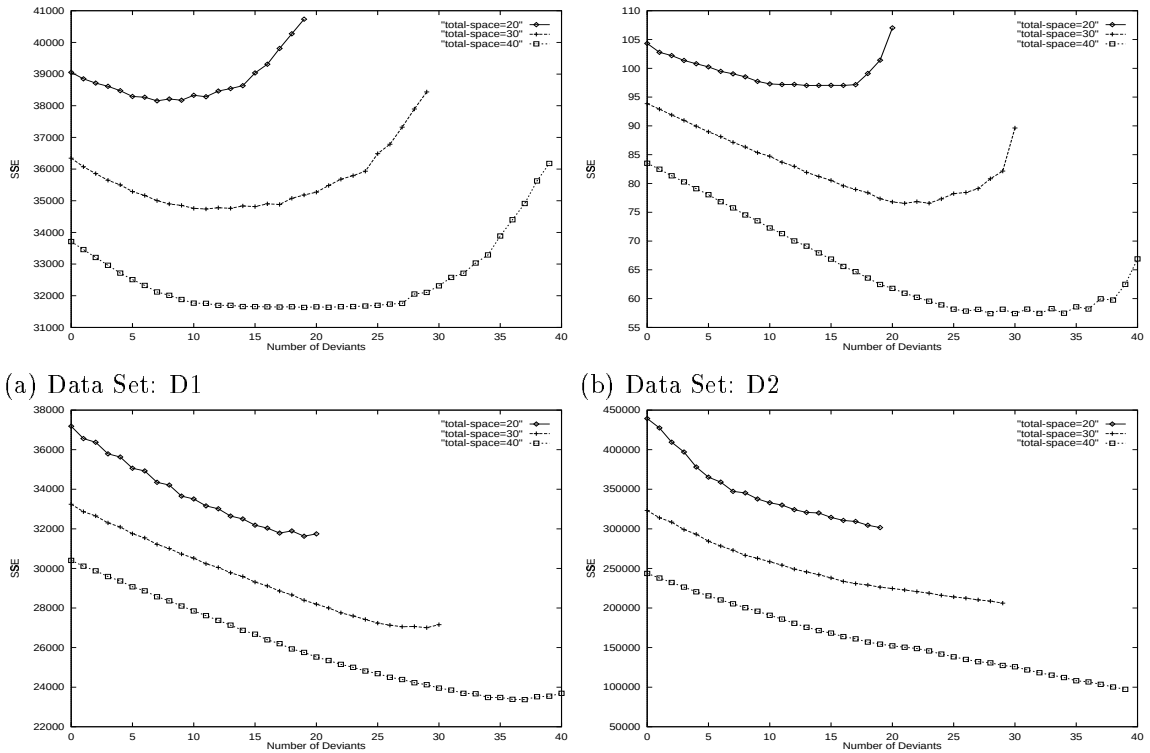
Our overall approach for data mining with deviants is as follows. We first determine a set of deviants in the original time series. We then analyze the set of deviants for useful structure. We discuss these two steps in the following subsections.

6.1 Deviant Sets

One method for obtaining a set of deviants for further analysis would be to choose parameters κ, the number of deviants, and B, the total storage allowed, and use the algorithms in Section 4 to determine an optimal set of deviants. A particular choice of interest for κ would be the optimal number of deviants for a given storage B. Another method would be to choose a small set of values for these parameters and determine the optimal deviant sets for each choice of the parameters. Then one can collate the "consensus" deviant points from the different optimal deviant sets. There are many ways to define the notion of consensus. Here, we adopt the notion exemplified in the following problem.

Problem 4 (Consensus Deviants) *Given sequence X of length N, a non-negative integer κ and parameters k_l, k_m, B_l, B_m, identify the κ most frequently occurring elements in the sets $D_{k,B}$ for $B_l \leq B \leq B_m$ and $k_l \leq k \leq k_m$, where each $D_{k,B}$ is the optimal set of k deviants for sequence X given B buckets of storage.*

It may seem on the face of it that solving Problem 4 requires multiple solutions to Problem 3 for different values of k and B. However, it turns out once again that the solution to Problem 3 already computes all the necessary additional solutions required here, so that only a single run of Problem 3 with a sufficiently large choice of k and B (k_m and B_m respectively) is required. It follows from our results in Section 4 that Problem 4 can be solved in time $O(N^2 B_m k_m^2)$. This method calls for choosing the parameter values k_l, k_m, B_l, B_m appropriately. Since one desires to isolate a few areas of potential interest, keeping κ small and varying k and B over small ranges is recommended. Many other



(a) Data Set: D1

(b) Data Set: D2

(c) Data Set: D3

(d) Data Set: D4

Figure 3: Histogram error for increasing number of deviants with fixed total space.

notions of consensus deviant sets can be incorporated into our approach here.

6.2 Deviant Clusters

The individual points in a set of deviants may be of interest by themselves, so they may be considered in isolation. However, from a mining perspective, it is useful to identify properties of groups of deviants in addition to those of individual deviants. More specifically, we consider the clustering of deviants. Since the deviants are derived from the underlying time series, it seems reasonable to consider contiguous segments (intervals) rather than arbitrary subsets of points in the time series, in order to understand the clustering of deviants. We adopt the rather natural notion that an interval is interesting only so far as the number of deviants in it significantly exceeds the "expected" number of deviants, based on uniformly random distribution of the deviants. The significance is controlled by a user-specified parameter δ. More precisely,

Definition 2 *Consider the time series in which κ deviants have been identified. An interval I is a* deviant cluster *if and only if $k \geq \delta |I| \frac{\kappa}{N}$, where k is the number of deviants in I and δ is a user-defined parameter of significance. Furthermore, I is a* maximal deviant cluster *if I is not a proper subset of any other deviant cluster.*

Note that the deviant clusters are not monotonic with respect to the interval size. To explain this further, we consider the example in Figure 4. Say we have identified six deviants and that we are interested in deviant clusters with size between six and twelve points and $\delta = 1$. Note that while $I1$, $I2$ are deviant clusters, $I4$, $I1 \in I4 \in I2$, is not one. Hence $I4$, which has a subset that is a deviant cluster and which is itself contained in a deviant cluster, is not a deviant cluster.

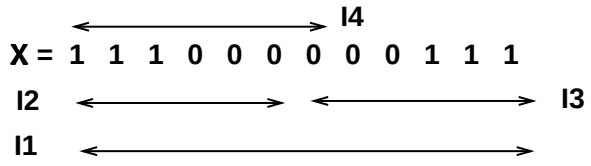


Figure 4: Non monotonicity of clusters.

Note further that while deviant clusters are monotonic with respect to δ, maximal deviant clusters do not have this property. In particular, as δ is decreased, longer intervals may get declared deviant and could subsume shorter intervals that were maximal deviant clusters for smaller values of δ.

6.3 Searching for Deviant Clusters

Given a set of κ deviants for a time series, we present an algorithm to determine the deviant clusters. Rather than return a specified subset of deviant clusters, our algorithm returns *all* maximal deviant clusters (these deviant clusters may overlap). Note that our algorithm can easily output all deviant clusters, not merely the maximal ones, but the output size may be too large for user attention.

Formally, we are given a binary vector $D = d_1 d_2 \ldots d_N$ such that $d_i = 1$ if d_i is a deviant, and it is zero otherwise. The user specifies a δ controlling the significance, as well as L, the minimum size and U, the maximum size of clusters of interest. The goal is to output all maximal deviant clusters of size between L and U.

The algorithm first computes the prefix sum of the number of deviants so that the number of deviants in any interval can be determined efficiently by looking at the prefix sums to the two endpoints of the interval.

The algorithm has two phases. In the first phase, it determines all the deviant clusters of length between L and U. In the first phase, the algorithm iterates over all intervals of interest ($(i, i+j-1)$ for $1 \leq i \leq N-j+1$ and $L \leq j \leq U$) and determines whether the number of deviants in the corresponding interval is above the threshold ($\frac{\delta \kappa j}{N}$), and if it is, it marks the interval as a deviant cluster by setting the array $\texttt{MaxCluster}(i, j)$ to 1. It does not matter in what order the `MaxCluster` array is evaluated: since deviant clusters are not monotonic, proceeding in increasing values of j (for a given i) does not necessarily help prune the search space.

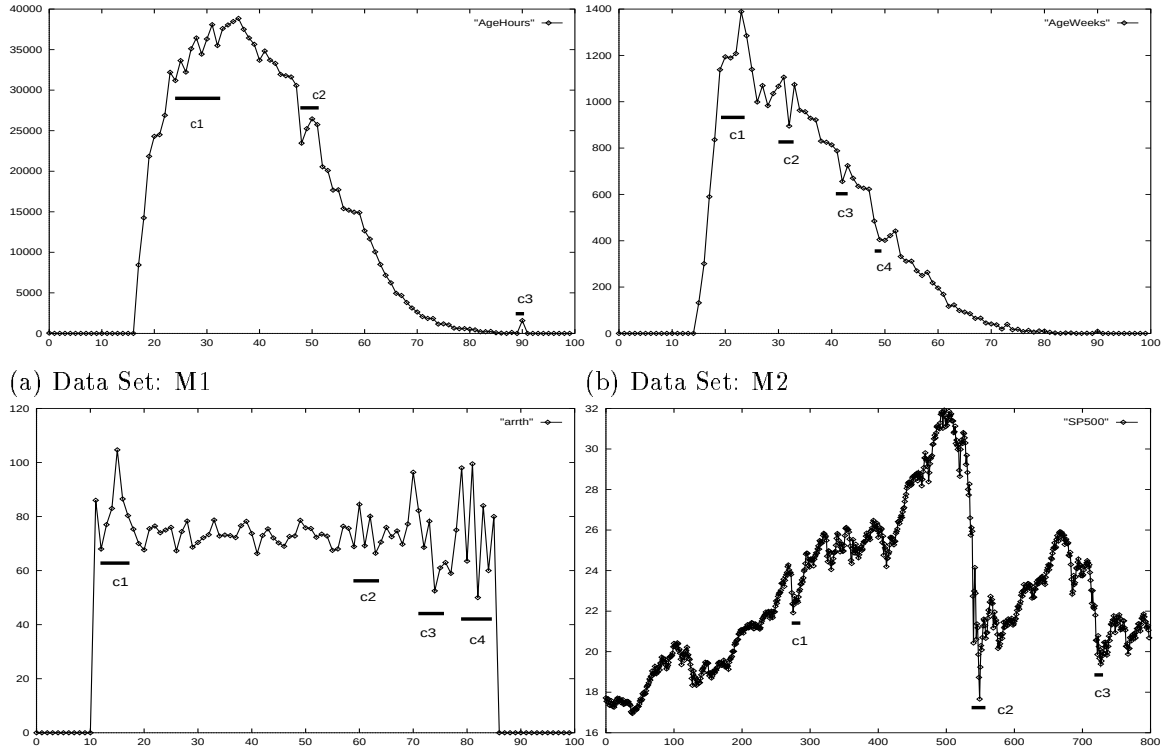
In the second phase of the algorithm, deviant clusters that are not maximal are removed. The algorithm proceeds by setting $\texttt{MaxCluster}(i, j) = 0$ if there exists a $k > j$, $i + j - k \leq l \leq i$, such that $\texttt{MaxCluster}(l, k) = 1$. (This is because if any such $\texttt{MaxCluster}(l, k) = 1$, then there is a deviant cluster $[l, l+k-1]$ which would contain the cluster $[i, i+j-1]$ that is under consideration.) This is implemented simply by iterating over all values $\texttt{MaxCluster}(i, j)$ for a given i.

Upon termination, the maximal deviant clusters are the intervals i to $i + j - 1$ where $\texttt{MaxCluster}(i, j) = 1$. The entire algorithm takes $O(N(U - L))$ time for each of the two phases.

7 Mining Experiments

We implemented a prototype system incorporating these algorithms that is capable of analyzing time series data. The user has to specify the parameters that defines the deviants, the parameter δ for the significance of the cluster density, as well as the minimum and maximum sizes of the clusters (L and U respectively) that are of interest. The system determines the deviants as described in Section 6 and also returns the maximal clusters of deviants meeting these specifications.

We present an exploration of a variety of publicly available data sets ($M1 - M4$ below). In all our experiments, we search for 10 most frequent deviants ($\kappa = 10$) and for choosing consensus deviants, we let k range between 1 and 10, and B between 1 and 40. Larger ranges for k and B are certainly possible. These values, however, provide a nice trade off between performance and the quality of the information obtained, in all our experiments. Figure 5 presents a subset of our results. We keep $\delta * \frac{\kappa}{N} = 0.4$ and search for clus-



(a) Data Set: M1 (b) Data Set: M2

(c) Data Set: M3 (d) Data Set: M4

Figure 5: Mining for deviant clusters in sample time series.

ters of size between 1 ($L = 1$) and 10 ($U = 10$) points. Note that one can potentially control the number of the clusters identified by changing the values of U and L.

The datasets that we present are described below:

- M1: A time series showing the total number of hours worked by people of a given age in a year, drawn from a census database (available at www.kdnuggets.com).
- M2: A time series showing the total number of weeks worked by people of a given age in a year, drawn from a census database (available at www.kdnuggets.com).
- M3: A times series showing the heart rate of individuals of different ages with arrythmia cases (available at www.ics.uci.edu/∼mlean/Machine-Learning.html).
- M4: A time series containing the closing S&P 500 index value each day starting from 1928.

For data set $M1$, we are able to identify the cluster containing points 23 to 30 (labeled $c1$ in Figure 5(a)), the cluster between 48 and 51 (labeled c_2), as well as the cluster at point 91 (labeled c_3). These clusters seem to match our intuitive visual notion of deviant sequences of points in this time series. If the value of δ is decreased, the identified clusters remain the same, but they become larger in size (encompassing more points). Similarly a larger value of δ produces the same, but smaller-sized clusters. Similar observations hold for Figure 5(b) where four clusters are identified (c_1 to c_4). Figure 5(c) is an example of a more "noisy" time series. Again, we can identify clusters corresponding to our intuitive visual notion of the "noisy" parts of the series. Finally, Figure 5(d), presents the results for M4. Curiously, cluster c_2 corresponds to a well known

financial disaster in the US economy in 1929.

We leave it to the reader to judge subjectively the quality of deviants we report. Notice that in each of the figures, the reported deviants (clusters) are rarely at the extrema of the functions; hence, deviant points and clusters are not necessarily the global extrema. Also in many cases, such as the datasets M1, M2, and M3 (in Figure 5a-c respectively), no deviants have been identified in the region of the fastest change in the function. Thus, deviants find regions that are not necessarily the local extrema of changes.

8 Conclusion

This work makes two specific contributions. First, we have presented a framework for the formal definition of a deviant point in time series, and demonstrated it to be of data mining value. We have proposed efficient optimal algorithms for the identification and mining of deviant points in a time series database. Second, we have shown how these algorithms can be used to decrease histogram error (or reduce storage for the same error) substantially.

This work raises several important issues for future exploration. Deviant points might have other properties as well, besides clusters, that might be of great mining value. For example, one could extend the framework and algorithms presented in this work, to mine for periodic patterns of deviants. Moreover, since deviants, most likely, will not be exactly periodic, notions of approximate periodicity, in the spirit of [HDY99], could be introduced. In addition, it would be interesting to couple deviants with other attributes as well, towards the design of effective discovery driven exploration tools for time series analysis.

References

[AAR95] A. Arning, R. Agrawal, and P. Raghavan. A Linear Method for Deviation Detection in Large Databases. *KDD*, August 1995.

[Bel54] R. E. Bellman. The Theory of Dynamic Programming. *Bull. Amer Math Soc. Vol 60*, pages 503–516, 1954.

[Cha84] C. Chatfield. *The Analysis of Time Series.* Chapman and Hall, 1984.

[GMP97] P. Gibbons, Y. Mattias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. *Proceedings of VLDB, Athens Greece*, pages 466–475, August 1997.

[HDY99] J. Han, G. Dong, and Y. Yin. Effcient Mining of Partial Periodic Patterns in Time Series Databases. *Proceedings of ICDE*, page to appear, March 1999.

[Ioa93] Y. Ioannidis. Universality of Serial Histograms. *Proceedings of VLDB, Dublin Ireland*, pages 256–277, August 1993.

[IP95] Y. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of ACM SIGMOD, San Hose, CA*, pages 233–244, June 1995.

[JKM+98] H. V Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. *Proceedings of VLDB*, pages 275–286, August 1998.

[KN98] E. Knorr and R. Ng. Algorithms for Mining Distance Based Outliers in Large Databases. *Proceedings of VLDB, New York*, pages 392–403, August 1998.

[PI97] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. *Proceedings of VLDB, Athens Greece*, pages 486–495, August 1997.

[PIHS96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD, Montreal Canada*, pages 294–305, June 1996.