Identifying Representative Trends in Massive Time Series Data Sets Using Sketches

Piotr Indyk
Stanford University
indyk@cs.stanford.edu

Nick Koudas
AT&T Laboratories
koudas@research.att.com

S. Muthukrishnan
AT&T Laboratories
muthu@research.att.com

Abstract

Many data stores, including scientific and financial databases, business warehouses and network repositories, contain time series data. Time series data depict trends for an observed value e.g., value of a stock, number of bytes sent on a router interface, etc., as a function of time. Analysis of the trends over different time windows is of great interest.

In this paper, we formalize problems of identifying various "representative" trends in time series data. Informally, an interval of observations in a time series is defined to be a representative trend if its distance from other intervals satisfy certain properties, for suitably defined distance functions between time series intervals. Natural trends of interest such as periodic or average trends are examples of representative trends.

We present efficient algorithms for analyzing massive time series data sets for representative trends over arbitrary windows of interest. Our algorithms are highly processor and IO efficient; they are approximate but provide probabilistic guarantees for the approximations achieved. Our approach for identifying representative trends relies on a dimensionality reduction technique that replaces each interval by a "sketch" which is a low dimensional vector. We present efficient algorithms to construct such sketches using a pool of select sketches that we precompute using polynomial convolutions. Using such sketches, we can compute representative trends accurately.

Finally, we present results of a detailed experimental study of our technique on very large real data sets. Our results show that, compared to approaches that determine representative trends exactly, our approach shows significant performance gains with only a small loss in accuracy.

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

1 Introduction

Many data sources are observations that evolve over time leading to time series data. For example, financial databases depict stock prices over time which is a common example of a time series. Reporting meteorological parameters such as the temperature over time gives rise to a time series in the area of scientific databases. Telecommunications and network databases represent many different time series data derived from the usage of various networking resources over time such as the total number and duration of calls, number of bytes or electronic mails sent out from one ISP to another, amount of web traffic at a site, etc. Business warehouses represent time series such as the sale of a specific commodity over time. Therefore, time series data abound in various databases.

Time series data depict the trends in the observed value over time, and hence, capture valuable information that users may wish to analyze and understand. For example, users may wish to know for a given time window, the "typical" trend of values or an "outlier" trend, in the intuitive sense of these words; or, as a dual, users may wish to find the time window such that most trends are as similar as possible or clustered. Finding such trends, what we will call "representative trends", will have many uses. For example, representative trends may be used in lieu of the entire data for quick approximate analysis; this maybe thought of as Data Reduction [2], specifically for the time series domain. In addition they can be used for identifying and detecting anomalous behavior or intrusion and for prediction.

We formalize the problems of finding various "representative" trends in time series data. In general, an interval of observations in the time series is defined to

be a representative trend if its distance to other intervals satisfy certain properties, for suitable distance functions defined between intervals. However, the intervals, properties and distance functions can vary and this leads to many different notions of representative trends. Another aspect of our study here is the focus on massive time series data. For example, AT&T collects around 500GB of data per year about one of the services it provides. Aggregate statistics about a single attribute (say usage of that service for every second) over a year results in a time series of size approximately 250MB, containing more than 31 million values. Running a quadratic algorithm for data analysis to analyze, say 5 years worth of data, would be prohibitively time consuming. The situation is aggravated if one wishes to perform analysis over a longer window of time or over a refined time scale.

In this paper, we formulate the problems of finding representative trends and focus on designing processor and I/O efficient algorithms for identifying them in large time series databases. In Section 2 we present examples of representative trends. In Section 3 we formally define various representative trends. Section 4 presents exact algorithms for identifying representative trends. These are expensive but exact. Section 5 presents our overall approach as well as our techniques for preprocessing time series data, which will be suitable for finding different representative trends. Section 6 presents how these techniques can be used to search for relaxed periods and average trends, presenting efficient algorithms for this task. Our algorithms will be very efficient, but will only be approximate; we provide guarantees on the accuracy. Section 7 contains the results of a detailed experimental evaluation of the proposed technique and algorithms, using real data sets, analyzing various tradeoffs. Section 8 discusses other notions of representative trends and application of our techniques to solve them. Section 9 presents related work and finally Section 10 presents concluding remarks.

2 Examples of Representative Trends

In this section, we describe two examples of representative trends in some detail, namely, relaxed periods and average trends, in order to motivate our more formal study in the later sections.

Relaxed Periods. The notion of a period of a time series is well understood[17]. Given a time series V, its period is T if $V = T^{\ell}T'$ where T' is a prefix of T and ℓ is some positive integer (typically taken to be at least 2). Informally, V is a repetition of non-overlapping copies of T. For example, 113 is a period of 11311311311311. However, rarely do time series data have exact period as defined above. So, in reality, one needs a relaxed notion of a period. We define a relaxed period of V to be T' if the sequence generated by repeating T' to the extent needed results in a sequence V' whose distance from V is the smallest possible in some distance measure, say, in sum of squares distance, for concreteness. (This will be formally defined in Section 3). According to this notion, the relaxed period of 213123213132213 will be 213.If there was a period for T, that will also be its relaxed period. ∎

Finding the Average Trend. Consider a time series $V[1 \cdots n]$ and a length l. V consists of $\lceil \frac{n}{l} \rceil$ disjoint subsequences of length l (except possibly the last one which may be shorter). One may ask which is the subsequence whose total distance to all the other subsequences is the smallest; again, we can fix the sum of squares as the distance function. We will call such a subsequence the average trend. For example, if $V = 113123213132113$ and $l = 3$, then we have 5 subsequences of interest, namely, 113, 123, 213, 132 and 113. The average trend is 123 which has a smaller total sum of squares distance than the others. ∎

Both relaxed period and average trend will be instances of representative trends we study in this paper. Representative trends suitable in different applications may differ, and many different notions of representative trends may be formalized and studied. For example, representative trends may not be global at all, but rather occur locally in significant ways and indeed our formulation below will enable us to express such variations. Our study will therefore be general, encompassing the many different notions of representative trends.

3 Definitions

Let $V[1, \ldots n]$ be a time series of length n. We adopt a vector notation and we refer to the time series V as a vector $\vec{V}$. We denote the i-th element of this vector by $V[i]$. For some integer T, we define $V(T) = \{\vec{v}_i\} = \{(V[iT+1], V[iT+2], \ldots V[iT+T])\}, 0 \leq i \leq \frac{n}{T} - 1$; if T does not divide n, the final subvector is not considered. Let $D(\vec{v}, \vec{u})$ be the distance between two vectors. In this paper, we will focus on L_2 which is a natural measure of distance between two vectors. We define

$$C^i(V(T)) = \sum_{j=0}^{\frac{n}{T}-1} D(\vec{v_i}, \vec{v_j}).$$

Definition 1 *Given a vector $\vec{V}$ and integers l and u, the* relaxed period *of $\vec{V}$ in range $[l, u]$ is the T, $l \leq T \leq u$ such that $C^0(V(T))$ is minimized.*

Definition 2 *Given vector $\vec{V}$ and integers l and u, the* average trend *of $\vec{V}$ in the range $[l, u]$ is the subvector $\vec{v_i}$ such that $C^i(V(T))$ is minimized for $l \leq T \leq u$.*

Both relaxed periods and average trends are examples of representative trends. In both these definitions, we have an interval $[l, u]$ of interest which specifies the length of the trend of interest. Note that if

$C^0(V(T)) = 0$, then T is the exact period of the sequence as is well understood [17]. Other variants of representative trends will be of interest as well.

4 Exact Algorithms

Let $\vec{V}$ be a time series and $[l, u]$ a range of interest in $\vec{V}$. Let us consider the exact algorithms to identify relaxed periods and average trends. Let $V(T) = \{\vec{v_i}\}$ be a set of vectors as defined in section 3, for $l \leq T \leq u$ and $0 \leq i \leq \frac{n}{T} - 1$. The brute force algorithm for identifying relaxed periods exactly is an $O(n^2)$ algorithm: for each T from l to u, for each i from 0 to $n/T - 1$, determine the minimum value of $D(\vec{v_0}, \vec{v_i})$. It can be implemented efficiently for large time series data, by assuming that for every $T \in [l, u]$ two vectors of size T can be stored in memory at any time and $u - l + 1$ counters can be maintained in memory. Under these assumptions the computation can be performed by a single pass on $\vec{V}$, and is likely to be processor bound for realistic sizes of $\vec{V}$ and $[l, u]$.

The brute force algorithm for identifying average trends exactly, is as follows: for each T from l to u, for each i from 0 to $n/T - 1$, for each j from 0 to $n/T - 1$, determine the minimum value of $D(\vec{v_i}, \vec{v_j})$. It is an $O(n^3)$ algorithm. In the worst case, we have to scan the data set for every value of $T \in [l, u]$ and every $i, 0 \leq i \leq \frac{n}{T} - 1$. In the best case, $\vec{V}$ fits in memory and for every value of T a quadratic number of evaluations of the distance function $D(.)$ between vectors of length T has to take place.

Application of the above algorithms in large scale time series data sets is formidably time consuming. Even for vectors of a few thousand points the above algorithms are very inefficient. In the next section, we introduce a technique than can lead to the design of efficient solutions for both problems even for very large data sets.

5 Sketching Approach

We will develop algorithms for finding relaxed periods and average trends which will be faster than the algorithms in the previous section, but will only provide an approximate answer. This is based on a sketch based approach which (1) is general and it applies to other notions of representative trends as will be discussed in Section 8, (2) gives guaranteed approximation performance, with high probability, as we prove. We will present our overall approach in steps:

1. First, we will define the *sketch* of a vector and state its properties. This will be useful in computing the distance between two subvectors efficiently.

2. We will then present an algorithm for finding the sketch of all subvectors of a given width T efficiently. This step will rely on polynomial convolutions.

3. We will then show how to determine the sketch of all subvectors of width in a given range. In order to do this efficiently, this will involve preprocessing the given vector into a pool of sketches for a chosen subset of subvectors, and then we will show how to extract the sketch of any subvector from the pool efficiently.

4. We will show how to find the representative trends of our interest using sketches. This will show the improvement in performance over the exact algorithms in the previous section as well as the guarantee on the loss in accuracy.

5.1 Sketch of a Vector

Given a vector $\vec{t} = t[1 \cdots \ell]$, we present an algorithm to construct its *sketch* vector $\vec{S}(t)$. $\vec{S}(t)$ is of size k. We generate $S(t)[i]$ as follows. We pick a random vector $v_i[1 \cdots \ell]$ by picking each component $v_i[j]$ to be an independent random variable with normal distribution $N(0, 1)$ and the entire vector $\vec{v_i}$ is normalized to 1 in magnitude. We define

$$\vec{S}(t)[i] = \vec{t} \cdot \vec{v_i} = \sum_j t[j].v_i[j]$$

This is the well known inner product between two vectors.

Example 5.1 Say $\vec{t} = (2, 1, 3, 1)$ and suppose we wish to construct a sketch vector of size two. We choose two vectors $\vec{v_1} = (-0.45, -0.09, 0.10, 0.87)$ and $\vec{v_2} = (-0.19, 0.73, -0.61, 0.21)$ and compute the inner product. The sketch vector $S(t)$ is $(0.18, -1.28)$. ∎

The sketch of a vector has many nice properties of which the one that interests us the most is the following which follows from the Johnson-Lindenstrauss Lemma [11]; hence we do not include its proof here.

Lemma 5.1 *For any given set L of vectors of length ℓ, for fixed $\epsilon < 1/2$, if $k = \frac{9 \log |L|}{\epsilon^2}$, then for any pair of vectors $\vec{u}, \vec{w} \in L$*

$$(1 - \epsilon)||\vec{u} - \vec{w}||^2 \leq ||\vec{S}(u) - \vec{S}(w)||^2 \leq (1 + \epsilon)||\vec{u} - \vec{w}||^2$$

with probability $1/2$. Here $||U - V||^2$ is the L_2 distance between two vectors U and V.

This lemma has the additional property that by increasing k, one can increase the probability of success as needed.

5.2 Fixed Window Sketches

In this section, we focus on computing the sketch for each subvector of a given length ℓ in $T[1 \cdots n]$. That is, we need to compute the sketch of $\lceil \frac{n}{\ell} \rceil$ vectors: $\vec{t}_1 = T[1 \ldots \ell], \vec{t}_2 : T[2 \ldots \ell + 1], \ldots, \vec{t}_{n-\ell+1} = T[n - \ell + 1, \ldots, n]$. Let us fix our attention on one of components, say, $S(t_i)[j]$ for each such vector $\vec{t}_i$.

The straightforward method would be as follows. We first generate random vector $\vec{v}_j$. We then consider each of the vectors $\vec{t}_i$ and compute $T[i, \ldots, i + \ell - 1] \cdot \vec{v}_j[1 \cdots \ell]$ directly. This takes $O(n\ell)$ time since there are $n - \ell + 1$ such vectors and for each we perform $O(\ell)$ work computing the inner product. Now we can repeat the whole procedure for the other components of the k sized sketch, in all taking $O(n\ell k)$ time all together. While this algorithm is practical for small ℓ, later, we will need sketches for rather large time windows, that is, large ℓ up to $O(n)$. For such cases, the straightforward algorithm above takes time $O(n^2 k)$ which is prohibitive for large n such as the ones we consider in our applications.

Our key observation here is that we can compute all such sketches fast by using Fast Fourier Transforms. Again, let us focus on one of the components of the sketch, say, $S(t_i)[j]$ for all i. The key observation is that *the problem of computing the sketches of all subvectors of length ℓ simultaneously is precisely the problem of computing the polynomial convolution of the two vectors $\vec{t}$ and $\vec{v}_j$.* This observation is evident when one considers the definition of the polynomial convolution.

Definition 3 *Given two vectors $A[1 \cdots a]$ and $B[1 \cdots b]$, $a \geq b$, their convolution is the vector $C[1 \cdots a + b]$ where $C[k] = \sum_{1 \leq i \leq b} A[k - i] \times B[i]$ for $2 \leq k \leq a + b$, with any out of range references assumed to be 0.*

For example, if $A = [1, 10, 2, 4]$ and $B = [7, 2]$, we have $C = [7, 72, 34, 32, 8]$. Polynomial convolution of two vectors can be computed in $O(a \log b)$ time using Fast Fourier Transforms. Here, we observe that

Lemma 5.2 *Sketches of all subvectors of length ℓ can be computed in time $O(nk \log \ell)$ using polynomial convolution.*

Proof. Consider reversing $\vec{v}_j$ and performing the polynomial convolution of $\vec{t}$ and $\vec{v}_j$. The output we are interested in is precisely $C[b + 1] \cdots C[a + 1]$ of the convolution. Repeat for all k components of the sketch. ∎

Figure 1 presents an example of the use of convolutions to compute sketches. A vector $(2, 1, 3, 1)$ is convolved two times with normalized normal vectors and the same coordinates of all three sketches of length two are computed at the same time.

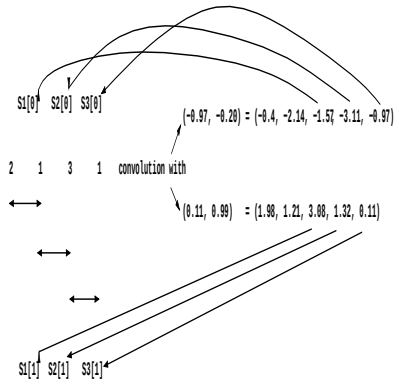


Figure 1: Using convolutions to compute sketches

5.3 Computing Sketches for Range of Subvectors

In this section, we consider the most general problem of computing the sketch for any subvector of length between l and u, of a given long vector. The most straightforward approach would be to consider all possible subvectors, and compute the sketch of each directly. There are $O(\sum_{l \leq i \leq u} \frac{n}{i})$ such subvectors. In the worst case, this is $O(n^2)$ subvectors and the majority of them are of size $\Theta(n)$, hence, the entire algorithm will take time $O(n^3)$ which is prohibitive. A less straightforward approach would be to apply our algorithm from the previous section with all possible values of ℓ. Since there are $O(u - l)$ possible values of ℓ, this algorithm will take $O(n^2 \log^2 n)$, which is better, but still prohibitive. In this section, we propose an algorithm to perform this task significantly more efficiently.

Our algorithm has the following structure. We will carefully construct a *pool* of sketches which we will store; this will be a small subset of the set of all sketches we need. Following this preprocessing, we will be able to determine the sketch of any subvector in the original vector in $O(1)$ time fairly accurately. In what follows, we will explain this procedure in more detail.

First we focus on the preprocessing. We choose $l \leq \ell \leq u$ such that ℓ is a power of 2. For each such ℓ, we compute the sketch of all subvectors of $\vec{t}$ of that length using our algorithm in Section 5.2. In fact, we compute *two* versions of sketches, each using different random variables; they are called S^1 and S^2. The resulting set of sketches is what we call the *pool*. This is of size $O(n \log(u - l)k)$ altogether and it takes time $O(n \log u \log(u - l)k)$ to compute; this is $O(n \log^3 n)$ in the worst case, hence, this algorithm scales well with the input size.

Second, we focus on determining the sketch for any subvector $\vec{t}_i$. Let us fix a particular component, say, $S(t_i)[j]$ for now. Two possibilities exist:

- $L = 2^r$, in which case, we have the sketches for this subvector in our pool, so we merely lookup

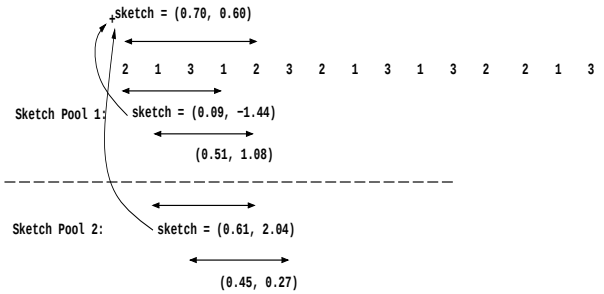


Figure 2: Synthesizing sketches for a subsequence of length 5 from sketches of subsequences of length 4 the desired sketch.

- $2^r < L < 2^{r+1}$. In that case, we compute as follows: $S'(T[i,\ldots,i+L-1])[j] = S^1(T[i,\cdots,i+2^r-1])[j] + S^2(T[i+L-2^r,\ldots,i+L-1])[j])$ (both the terms on the right belongs to our pool). See Figure 2 for an example.

We claim that S' satisfies a property very similar to the one in Lemma 5.1, namely:

Theorem 5.1 *For any given set L of vectors of length ℓ, for fixed $\epsilon < 1/2$, if $k = \frac{9\log|L|}{\epsilon^2}$, then for any pair of vectors $\vec{u}, \vec{w} \in L$*

$$(1-\epsilon)||\vec{u}-\vec{w}||^2 \leq ||\vec{S'}(u)-\vec{S'}(w)||^2 \leq 2(1+\epsilon)||\vec{u}-\vec{w}||^2$$

with probability $1/2$.

Note the additional factor 2 in the second inequality.

Proof. The idea of the proof is to use *stochastic dominance*. Recall that for two random variables X and Y we say that Y *dominates* X (or $X \leq Y$), if for any z we have $Pr[Y > z] \geq Pr[X > z]$. The dominance relation is known to be preserved under monotone functions, i.e., if $X_1 \leq Y_1, \ldots, X_l \leq Y_l$, then $f(X_1,\ldots,X_l) \leq f(Y_1,\ldots,Y_l)$ if f is a monotone function. We will also use the additive property of the normal distribution, namely if $X_1,\ldots,X_l$ are independent variables with $N(0,1)$ distribution, then for any sequence $a_1,\ldots,a_l$ of real numbers the variable $X = \sum_i a_i X_i$ has distribution $N(0,\sum_i a_i^2)$. Finally, for any interval $I \subset \{1\ldots l\}$, define 1_I to be a vector of length l containing 1's at positions belonging to I and zeros elsewhere.

We are now ready to prove the theorem. Define $I_1 = \{i,\ldots,i+l-1\}$ and $I_2 = \{i,\cdots,i+2^r-1\}$. Note that $1_{I_1} + 1_{I_2}$ contains only 1's and 2's (no zeros), since $I_1 \cup I_2 = \{1,\ldots,l\}$. Let v_i^b be the vectors used for sketch S^b. Let $x = u - w$ and observe that $S'(u)-S'(w)][i] = S'(x)[i] = v_i^1\cdot x + v_i^2\cdot x = (v_i^1+v_i^2)\cdot x$. By additivity property, the jth coordinate of $v_i^1 + v_i^2$ has distribution $N(0,2)$ if $j \in I_1 \cap I_2$ and $N(0,1)$ otherwise. Thus $S'(x)[i]$ has distribution $N(0,D)$, where $D = \sum_j (1_{I_1} + 1_{I_2})[j](x[j])^2$. For comparison, note that $S(x)[i]$ (as in Lemma 5.1) has distribution $N(0,\sum_j (x[j])^2)$ and $\sqrt{2}S(x)[i]$ has distribution $N(0,2\sum_j (x[j])^2)$. Therefore (by monotonicity) $||S(x)|| \leq ||S'(x)|| \leq \sqrt{2}||S(x)||$. The theorem then follows from Lemma 5.1. ∎

5.4 Using Sketches to Compute Representative Trends

In this section, we will show how to compute the representative trends of our interest, namely, relaxed periods and average trends, using sketches.

Let us first consider finding relaxed periods. First, we will preprocess the given vector $\vec{V}$ as in the previous section so that the sketch of any subvector can be computed in $O(1)$ time. This takes time $O(n\log(u-l)k\log u)$ if we know $[l,u]$ or $O(nk\log^2 n)$ in the worst case. As in the exact algorithm, we will now consider all T, $l \leq T \leq u$, and for each T, we compute $C^0(V(T))$ by considering each $\vec{v}_i$ in turn and computing $D(\vec{v}_0,\vec{v}_i)$. The key is that we can now estimate $D(\vec{v}_0,\vec{v}_i)$ in $O(k)$ time by determining the sketch of $\vec{v}_0$ and $\vec{v}_i$. This process takes time $O(\sum_T \frac{n}{T}k) = O(nk\log n)$ time in the worst case (here we have used the harmonic series summation formula). Choosing $k = \frac{9\log n}{\epsilon^2}$ from Lemma 1, it follows that the relaxed period we find will be at most a factor of $2+\epsilon$ away from the true relaxed period with high probability (this probability can be made essentially as high as we need by choosing larger k, for example, by picking k to be twice as much, the probability that we find a relaxed period more than $2+\epsilon$ away from the true relaxed period is at most $1/n$, a tiny quantity!).

The algorithm for computing the average trend using sketches is a similar modification to that of finding exact average trend: wherever $D(\vec{v}_i,\vec{v}_j)$ is needed, we use the estimate for that distance derived from the sketch of $\vec{v}_i$ and $\vec{v}_j$. Again, we will find a $(2+\epsilon)$ approximation with very high probability.

6 Implementation Issues

Making our proposed sketching technique practical involves addressing important issues such as constraints in main memory size, efficient retrieval of sketches from the pool, and the impact of these constraints on the overall performance of our technique.

6.1 Computing Sketches

Assume the data set consists of n points, the sketch window is ℓ and the size of each sketch is k points. If no main memory constraints exist, an application of our technique would involve reading the data set of N points in memory, allocating $k*(n-\ell+1)$ space for the sketches and performing k convolutions to compute the sketches each requiring time $O(n\log n)$. With realistic data sizes and a fixed memory size, it is unlikely that the data set will fit in main memory. In that case, we have to read the data set in pieces, compute a batch of sketches each time and insure that we have enough space to keep both the computed batch of sketches as

well as the data points of the data piece we are working on, in memory.

Let B be the total memory available for our approach. Let M be the size of the data piece that we have to read from disk. Applying our sketching on M points will produce $M - \ell + 1$ sketches, each of size k. Thus, we have to guarantee that $M + k*(M-\ell+1) \leq B$ and the value of M can be derived. Once the first batch of sketches is computed in main memory we can write it to disk and continue, by reading the next piece and constructing the next sketch batch. This process is repeated until we read the entire data set. A data piece of size M is loaded into memory and the sketches are computed for each subsequence of size k. Notice that construction of the first batch of sketches can be performed independently from the construction of the second batch. We only need to maintain the last $k-1$ points from the first piece of size M, in memory, and continue with the construction of the second batch, Thus, sketches can be computed with a single scan of the underlying data set. The fact that sketch batches can be computed almost independently, makes sketching ideal for a parallel implementation, either on an SMP or distributed memory environment. Only a small amount of information needs to be communicated across processors.

6.2 Retrieving Sketches from the Pool

Once sketches are computed and the sketch pool is materialized on disk, we have to access the pool and retrieve sketches required for computing sketches of sequences in suitable range $[l, u]$. We discuss the following two cases:

Relaxed Periods. Under the assumption that we have space in memory to store two vectors of size at most u as well as $u - l + 1$ counters in memory, we can compute the corresponding clustering for every length in the range with a single pass of the sketch pool. Clustering C^0 evaluates the distance of the first sketch to all the others, so we only need to maintain the first sketch for each length in the range in memory and accumulate the value of the clustering for each length in memory.

If this assumption does not hold, we have to perform random disk accesses and retrieve the required sketches from secondary storage. However, clusterings for a number of consecutive lengths in the range can be evaluated at the same time. More specifically for $l \leq m \leq m+1 \leq u$ the offsets of required sketches are at j sketches away for the case of the first sketch pool and $j+1$ away for the case of the second, $0 \leq j \leq \frac{N}{m}$. Thus, if for each value of m and j we prefetch j $(j+1)$ sketches from the first (second) pool we would be able to compute the sketches for the sequence of length $m+1$ entirely in memory, saving the disk accesses. Thus, by employing selective prefetching between successive values of m we can reduce the total number of disk accesses by half. Subsequently, additional IO savings are possible if one is willing to prefetch more aggressively. The amount of prefetching depends on the memory constraints which in turn determine the number of successive values of m one can evaluate simultaneously. This observation provides a nice tradeoff between main memory usage and disk access time.

Average Trend. Evaluating average trends involves application of C^i clustering. A single scan of the pools is not enough, as all pairwise evaluations of distance between sketches corresponding to sequences of the same length is necessary. Constructing sketches by random accesses for every length is needed. This way, we are able to bring all sketches corresponding to a specific length in memory and evaluate C^i entirely in memory. Prefetching can be applied in this case as well, to save disk accesses by devoting more memory to store sketches.

7 Experimental Evaluation

We implemented the proposed sketching technique as well as the proposed algorithms to check for various kinds of representative trends and in this section we present detailed experimental results evaluating our approach. We begin our description of the experiments by describing the data used in our evaluation and then we continue with the presentation of experimental results from each of the classes of experiments we performed. With the experiments in this section we are interested in evaluating the time to construct sketches as well as the time to compute relaxed periods and average trends. In all experiments in this section whenever we retrieve sketches from the pool to synthesize sketches we do so by **random IO** accesses to the sketch pool. Our experiments were run on a SUN sparc Ultra Enterprise 8 processor SMP. The IO transfer rate in our configuration was approximately 10MB/sec.

Due to space limitations only a small subset of our experimental evaluation is presented. More discussion is available elsewhere [9].

7.1 Description of Datasets

All the datasets used in our performance study are real, extracted from one of the warehouses we maintain at AT&T Labs. All the data sets are time series containing utilization information of one of the services AT&T provides to customers. The performance data are collected in the granularity of a second and we range the time duration from a month (approximately 16MB of data) to a year (approximately 256MB).

7.2 Evaluating Time to Compute Relaxed Periods

We compare the performance of the proposed sketching technique, as applied in the identification of re-

laxed periods, with that of the brute force algorithm. Our treatment of the brute force algorithm is favorable since we assume that given a range of length p, we have the space to store two vectors of size up to p as well as space to maintain p variables, all in main memory. Thus we can simultaneously evaluate the clustering for all candidate relaxed periods in memory with a single scan of the time series.

In Figure 3(a), we search for the best relaxed period using our proposed sketching technique. The data set size is 16MB and we are interested in finding the best relaxed period, within specified ranges of values. The impact of our approach is most evident when ranges are large enough, so we start at offset 512 in the sequence. We vary the range from 512 to 128K. Thus, the first range is [512,1024], the second, [512, 1536] etc. Figure 3(a) presents the results of two experiments. The first one uses a sketch size equal to the logarithm of the number of data points in the data set and the second uses a sketch twice that size. For each range of values we compute the sketches necessary, from scratch. We present, for each range, the time required to construct all the necessary sketches plus the time to search for the best relaxed period within that range. For example, checking for the best relaxed period in the range [512,1536] consists of constructing sketches with a window of 512 points, using these sketches to check the lengths in the range [512,1023], and then constructing sketches with a window of 1024 points, and checking the lengths in range [1024,1536]. As we increase the size of the sketch, the time to perform the computation increases as is evident in Figure 3(a). This is because, the number of required convolutions doubles, and thus the required processor time.

Figure 3(b) presents the results of a scalability experiment plotting the total time to search for the best relaxed period in a range of size 8K as the size of the data set increases. The total time includes the time to construct the sketches on a suitable sketch window and the time to search for the best relaxed period in that range. Again, we report two experiments, using sketches of size $\log(n)$ and $2 * \log(n)$ were n is the number of points in each data set. Our sketching approach requires time linear to the size of the data set. Moreover, the time approximately doubles by doubling the data set size. This is expected as the number of sketches almost doubles by doubling the data set size and thus the processor time required for their construction. In addition more time is spent reading the data set, but IO time is only a small fraction of the computation.

As a comparison, figure 4(a) presents the total time (in minutes) requested by the brute force algorithm to compute the best relaxed period in a range p as the range increases from 512 to 128K for a 16MB data set. The time to read the data is constant for each of the ranges since the computation for the entire range is performed by a single scan of the time series. Thus, the computation performed by the brute force algorithm is processor bound. In figure 4(b) we present the time to compute the best relaxed period for a range of 8K, as the data size increases from 16MB to 256MB.

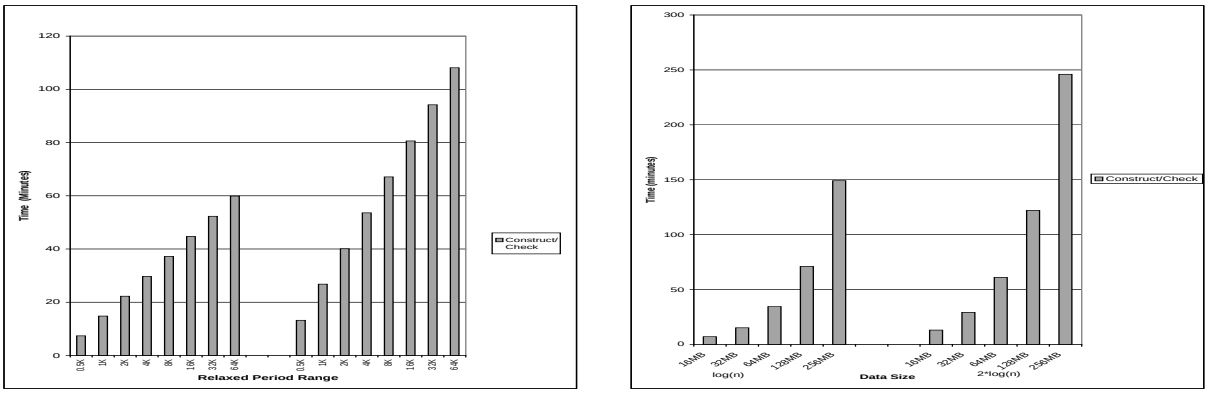
From the experiments in this section, it is evident that the proposed sketching technique offers important savings during the search for the best relaxed period, when compared with a brute force approach. Contrasting figures 4(a) and 3(a) two major performance trends are evident. First, as the range of lengths increases the performance benefits of our sketching approach become continuously better. For the 0.5K range, the time required by our sketching algorithm is slightly higher than that of brute force, when sketches are computed from scratch. Sketching is not beneficial for testing relaxed periods, in very small ranges, if sketches have to be constructed from scratch, since the overhead of constructing them becomes higher than the time required by the brute force algorithm to perform the exact computation. This observation makes sketching the method of choice for larger ranges. The exact performance cross over point between the two approaches is difficult to quantify since it depends of the machine characteristics. In our case, for any length above 512 points, sketching is always beneficial. Second, by increasing the sketch size, the cross over point essentially moves to the right, but again quickly sketching becomes the method of choice. For the range of values shown in the figure, sketching becomes almost an order of magnitude faster. The performance benefits increase as larger ranges are considered.

Contrasting figures 4(b) and 3(b) one can make a similar observation. Sketching appears 4 times faster than brute force for a sketch size of $\log(n)$ and almost 2 times faster for $2 * \log(n)$ sketch size. These performance benefits will increase when a larger range of lengths in considered, than the 8K range in the figures. In particular the performance benefits of sketching approximately double by doubling the range of lengths over which we search.

The application of the proposed technique to the identification of average trends offers much larger performance benefits when compared to brute force. Details are available elsewhere [9].

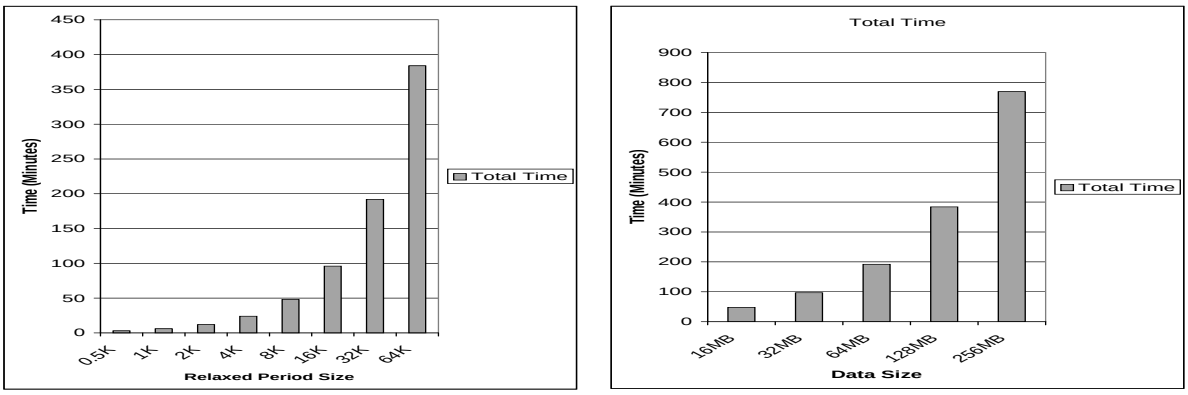
7.3 Evaluating the accuracy of the approximation

The proximity of the resulting approximation to the optimal relaxed period or average trend depends on the sketch size. We showed analytically the exact dependency of the sketch size to the approximation value ϵ. In this section we experimentally evaluate the influence of the sketch size to the resulting approximation. We conducted the following experiment: for a specific data set, we determine using the brute force algorithm



(a) Increasing Relaxed Period Range Data Size 16MB(b) Increasing Data size, window size 8K

Figure 3: Approximate relaxed period computation



(a) Increasing Relaxed Period Size, Data Size 16MB(b) Increasing Data size, range 8K

Figure 4: Brute Force Algorithm

the optimal relaxed period and the corresponding L_2 error. We then use our technique to find the best relaxed period as the sketch size increases and we measure the absolute relative error between that of the optimal L_2 value and the L_2 value resulting from our approximation. Let O be the optimal L_2 value and $O^{'}$ the approximated. The absolute relative error (ARE) is defined as:

$$ARE = \frac{|O - O^{'}|}{O} \quad (1)$$

Figure 5 presents the results of this experiment. It shows the results for two data sets containing 1M points, representing utilization information of an AT&T service. We include the results for two data sets with different statistical characteristics. Let n be the size of the data set. In both cases we can observe that using a sketch size between $\log(n)$ or $2\log(n)$ provides reasonable accuracy, close to the optimal value. This behavior was consistent over a large collection of data sets we used in our experiments and we experimentally recommend these values. If guaranteed accuracy is required, one should select the sketch size according to Theorem 5.1 as a function of ϵ, the error one is willing to tolerate.

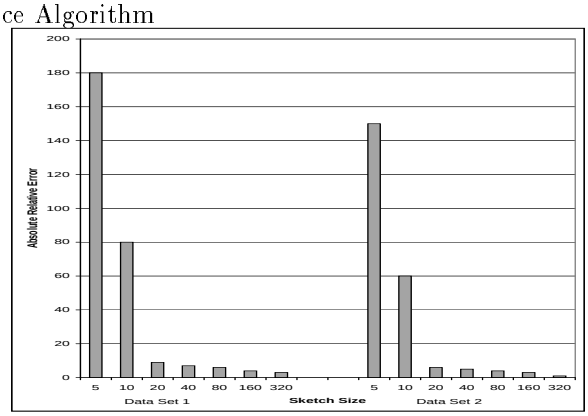


Figure 5: Evaluating the accuracy as a function of the sketch size

7.4 Comparison with Fourier Transform Based Approach

In the context of times series data, the use of fourier transform has been proposed in the community for dimensionality reduction and subsequent query processing [5]. This approach consists of the following steps: (a) compute the fourier transform of the time series segment, (b) maintain a number of fourier transform coefficients (usually the first few coefficients), and (c)

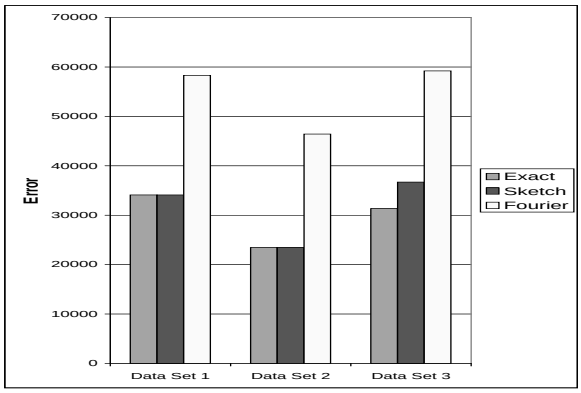


Figure 6: Error for exact relaxed period, relaxed period based on sketches, and relaxed period based on fourier coefficients

use the coefficient vector during query processing. Notice that performing such a fourier analysis of the time series data does not provide us with relaxed periods or average trends directly. However, a natural way to employ this approach for finding representative trends is to use fourier transforms as sketches. A major technical hurdle for this approach is that we are not aware of a method to combine fourier coefficients (in the spirit of Theorem 5.1 for our proposed sketching approach) and derive the fourier coefficients of a larger time series segment. This means that fourier-transforms based approach, at least in its natural version, will be inefficient since we need to recompute the fourier transform for every sketch window. Nevertheless, in order to understand the relative merits of fourier transform based approach for finding representative trends, we focused only on comparing the accuracy of the two methods (our approach vs the fourier transform based approach) without regard to their respective efficiency. We performed the following experimental study. For the case of fourier transform, for each sketch window, we computed the fourier transform and maintained as many leading coefficients as the size of the sketch. (Notice that we explicitly compute the fourier transform for each sketch window). For all the experiments we performed, our proposed sketching technique, was almost always able to identify correctly the relaxed period of the average trend in contrast with the use of fourier coefficients that failed consistently to do so. In the cases that the proposed sketching technique failed to identify the exact relaxed period, the relaxed period returned was more accurate than the one returned with the use of fourier coefficients. Results for three sample datasets are presented in figure 6.

7.5 Experimental Observations on Real Datasets

We used the proposed sketching technique to analyze large time series data sets in order to identify relaxed periods and average trends. Our analysis revealed several unexpected findings. For example in some data sets, over a very large period of time, the relaxed period was at the granularity of a week. Moreover, in other data sets, the relaxed period was at the granularity of a month. We also discovered data sets, having both a relaxed period and an average trend at the granularity of a single day. Different services that provided the data sets had particular bias: day, week or month, etc. While one expects the "periods" to be aligned with the natural time, it is intriguing why some data sets align with days and others with weeks or months.

8 Extensions

The approach of using sketches is quite powerful. We can formulate many other notions of representative trends and compute them using sketches; the results will of course be an approximation, but guaranteed to be a small factor (such as $2+\epsilon$) with a very high probability. The idea in all such applications is that instead of using the distance $D(\vec{v}_i, \vec{v}_j)$, we can use its estimate from the sketch for $\vec{v}_i$ and $\vec{v}_j$. The process is extremely efficient because we can compute these sketches in $O(1)$ time after a preprocessing step of $O(n \log^2 n)$ in the worst case. As an example we can find local relaxed periods rather than global ones, that is, we can find all subvectors which have at least as many period repetitions as specified by a threshold, for a given window size T. In particular the framework presented in [7, 8] for partial periodic patterns, can be expressed using sketches on subsequences.

9 Related Work

A number of dimensionality reduction techniques have been proposed in the database literature including FastMap [4], applications of fourier transforms [1, 5], Singular Value Decomposition [16, 12], wavelet transforms [14, 18], and the cosine transform [13]. However, none of the above techniques has the property of our sketching technique, as proved in Theorem 5.1. This property is crucial for the algorithmic framework we presented in this paper. Indyk and Motwani [10] and Gionis et. al., [6] provide a framework based on lemma 5.1 for nearest neighbor search in multiple dimensions.

Identifying representative trends in a time series database is a problem that, to the best of our knowledge, has not been addressed in the database literature. In the case of identifying relaxed periodic patterns, one can imagine the possible application of techniques based on fourier transforms [15]. Our work, specifically, for the problem of identifying relaxed periods, offers the following advantages: (a) it guarantees that the identified relaxed period is the best *internal* (consisting of a part of the data set) relaxed period, as opposed to, a sinusoidal approximation of it (b) it identifies relaxed periods under various metrics, as opposed to only L_2 as is the case for fourier transforms (c) since time series are approximately periodic (noisy

signals) an application of fourier transform will involve filtering of the time series before processing to guarantee accuracy. Even with state of the art filtering techniques, and even assuming that no phase shifts have occurred, fourier transforms cannot offer the type of approximation guarantees that the approach presented herein offers.

Han et. al., [8, 7] in their pioneering work, defined notions of partial periodicity in time series and proposed IO efficient algorithms to identify partial periods in time series datasets. Let p be a user specified period in a time series of length n. Han et. al., [8] defined partial periodicity as the existence of $f * \frac{n}{p}$ (f a user specified parameter) similar subsets (one for each of the $\frac{n}{p}$ segments), of maximum size p. Similarity between two subsets is defined in terms of equality in the corresponding elements. Subsequently, Han et.al., [7] generalized their framework to search for partial periods without specifying the period in advance. Our framework for partial periodicity is related to that of Han et.al. We do not consider similar subsets however, but similar subsequences. Two subsequences are considered similar if their distance (under some metric) is smaller than a user specified threshold. Been less strict in our definition of partial periodicity, we are able to scale our algorithms to massive datasets in an approximate way.

Techniques based on Hidden Markov Models [3] could be applied towards the type of time series analysis considered in this paper. We plan to investigate the relationship further in the future.

10 Concluding Remarks

We introduced the problem of identifying representative trends in massive time series data sets. Our work makes the following specific contributions. We presented processor and IO efficient algorithms to search for relaxed periods and average trends in large time series data bases. Our algorithms are based on a sketching technique where we showed how to efficiently construct a pool of sketches and combine them in a specific way such that other sketches can be derived from this combination efficiently. Our experiments with real data sets of realistic size showed that sketching is a fast approach for these problems and significant performance benefits are attainable. To the best of our knowledge our work is the first that addresses this important problem and provides efficient solutions for it. Given the flexibility sketching offers for approximate computations, it will be worthwhile to offer its applications to other problems of interest in time series analysis.

11 Acknowledgments

We wish to thank the referees for their comments and Dennis Shasha for overseeing the preparation of the final manuscript.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. *Proc. of the 4th Int'l Conference on Foundations of Data Organization and Algorithms*, pages 69–84, Oct. 1993.

[2] D. Barbara, C. Faloutsos, J. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. Sevcik. The new jersey data reduction report. *Data Engineering Bulletin*, Sept. 1996.

[3] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[4] C. Faloutsos and D. Lin. FastMap: A Fast Algorithm for Indexing, Data Mining and Visualization of Traditional and Multimedia Data Sets. *Proceedings of ACM SIGMOD, San Jose California*, pages 163–174, June 1995.

[5] C. Faloutsos, M. Ranganathan, and I. Manolopoulos. Fast Subsequence Matching in Time Series Databases. *Proceedings of ACM SIGMOD*, pages 419–429, May 1994.

[6] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. *Proceedings of VLDB, Endiburgh, England*, Sept. 1999.

[7] J. Han, G. Dong, and Y. Yin. Efficient Mining of Partial Periodic Patterns in Time Series Databases. *Proceedings of ICDE*, pages 106–115, Mar. 1999.

[8] J. Han, W. Gong, and Y. Yin. Mining Segment-Wise Periodic Patterns in Time Series Databases. *KDD*, pages 214–218, Aug. 1998.

[9] P. Indyk, N. Koudas, and S. Muthukrishan. Indentifying Representative Trends In Massive Time Series Data Sets Using Ske tches. *AT&T Labs Technical Report*, Feb. 2000.

[10] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *30th Symposium on the Theory of Computing*, Sept. 1998.

[11] W. B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert Space. *Contemporary Mathematics, Vol 26*, pages 189–206, May 1984.

[12] K. V. R. Kanth and A. Singh. Dimensionality Reduction For Similarity Searching In Dynamic Databases. *Proceedings of ACM SIGMOD*, pages 97–105, June 1998.

[13] J. Lee, D. Kim, and C. Chung. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. *Proc. of the 1999 ACM SIGMOD Intern. Conf. on Management of Data, June 1999.*

[14] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *Proc. of the 1998 ACM SIGMOD Intern. Conf. on Management of Data, Jun e 1998.*

[15] A. Oppenheim and A. Willsky. *Signals and Systems*. Prentice Hall, Signal Processing Series, Aug. 1992.

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

[17] D. Sankoff and J. Kruskal. *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, Mass.,, 1983.

[18] J. Vitter and M. Wang. Approximate Computation Of Multidimensional Aggregates On Sparse Data Using Wavelets. *Proceedings of SIGMOD*, pages 193–204, June 1999.