Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees*

Hazem Elmeleegy[#], Ahmed K. Elmagarmid[#], Emmanuel Cecchet[‡], Walid G. Aref [#], Willy Zwaenepoel[§]

[#]Computer Science Department, Purdue University
{hazem,ake,aref}@cs.purdue.edu
[‡]Computer Science Department, University of Massachusetts
cecchet@cs.umass.edu
[§]EPFL
willy.zwaenepoel@epfl.ch

ABSTRACT

Continuous "always-on" monitoring is beneficial for a number of applications, but potentially imposes a high load in terms of communication, storage and power consumption when a large number of variables need to be monitored. We introduce two new filtering techniques, swing filters and slide filters, that represent within a prescribed precision a time-varying numerical signal by a piecewise linear function, consisting of connected line segments for swing filters and (mostly) disconnected line segments for slide filters. We demonstrate the effectiveness of swing and slide filters in terms of their compression power by applying them to a real-life data set plus a variety of synthetic data sets. For nearly all combinations of signal behavior and precision requirements, the proposed techniques outperform the earlier approaches for online filtering in terms of data reduction. The slide filter, in particular, consistently dominates all other filters, with up to twofold improvement over the best of the previous techniques.

1. INTRODUCTION

Continuous monitoring in distributed environments is widely applied in many contexts including sensor networks, moving objects, stock market, computer networks and distributed systems. Continuous queries over the incoming data streams are posed through a central Data Stream Management System (DSMS) to obtain useful information from the raw data. In addition to the benefits of online monitoring, it is often desirable to store the results for later offline analysis.

The number of data streams can get quite large because of the many objects that may need to be monitored. A high sampling frequency is desirable as it helps provide a detailed and accurate model of the monitored signal. For large and complex systems, where continuous monitoring is most useful, the combination of these requirements leads to a very large volume of monitoring data, imposing a substantial burden on the network and the repository used for storing the monitoring data. Even more seriously, in sensor network applications, the sensors' battery lifetime (and hence the lifetime of the whole sensor network) is predominantly dependant on the amount of transmitted data [14].

Much work has addressed the problem of compressing time series data by a given ratio, while attempting to minimize the approximation error (See [22] for a review of time series approximations). Less attention, however, has been given to the dual problem of guaranteeing a given error bound, while attempting to maximize the compression ratio. For both problems, piece-wise linear approximation has been one of the most widely used and accepted methods [22]. We generally refer to the techniques used to solve the second problem as *filtering techniques*. Roughly speaking, the filter predicts future data values, and if the actual measured value falls within the error bound around the predicted value, no new recording is made. If during steady-state operation the data follows a certain pattern, filtering can substantially reduce the amount of monitoring data that needs to be transmitted and recorded.

In this paper we present two novel filtering techniques: *swing filters* and *slide filters*. Their compression power exceeds by a large margin the best of the previous filtering techniques, with up to twofold improvement for the slide filter. They also impose a low overhead, which makes them very practical for overhead-sensitive applications like sensor networks.

Essentially, the newly proposed filters approximate time-varying numerical signals by a piecewise linear function, consisting of connected line segments in the case of the swing filter, and (mostly) disconnected line segments in the case of the slide filter. At any point in time, each of the filters maintains *a set of possible line segments*, all obeying the invariant that they represent the data observed so far. As each new data point arrives, the set is reduced to maintain this invariant.

Swing and slide filters improve over earlier cache and linear filters in that the latter two only maintain a single line segment at any given time, while the former two maintain a set of such line segments. As a result, swing and slide filters can capture more future data points within their approximation, and thus further reduce the number of recordings that need to be made. By allowing disconnected line segments, slide filters can capture an even larger set of future data points, at the expense of two recordings per line segment instead of one in the case of the swing

[*] Part of this work was done while the first and third authors were at INRIA Rhône-Alpes, France. Walid G. Aref's research is partly supported by NSF under Grant Number IIS-0811954.

filter, which always produces connected line segments.

While the compression power of our proposed filters comes from the fact that they postpone their selection of line segments as long as possible, this postponement also introduces a *lag* between the transmitter and receiver. We thus allow applications to set an upper bound for this lag by limiting the maximum number of data points a transmitter can process locally before updating the receiver. In other words, applications can choose the tradeoff point between the compression ratio and the length of the lag.

In practice, many applications do not consider the timeliness of data delivery as their top priority. For example, online stock quotes and foreign exchange rates are usually lagging a few minutes behind the actual market data. Also, it is very common in the area of sensor networks to give higher priority to data reduction (which leads to transmission rate reduction and ultimately power conservation) over the timeliness of data delivery (e.g. [9,18,19]).

In our work, we focus on the class of applications which can tolerate a bounded error and a bounded lag in the received data points (where both bounds can be set by the application), in return of a higher compression ratio.

Moreover, we observed that when multiple monitored signals are *correlated*, compressing them together as a multi-dimensional signal is more effective compared to compressing each signal independently. This was like an extra bonus because our design for the swing and slide filters was general enough to enable processing multi-dimensional as well single-dimensional signals.

We have implemented swing and slide filters, in addition to cache and linear filters, and applied them to a real data set from the oceanography domain and a wide variety of synthetic data sets. In summary, the contributions of this paper are:

1. The design and implementation of two new types of filters for the online piece-wise linear approximation of multi-dimensional data streams: swing and slide filters.
2. A theoretical analysis including the proofs of correctness of the two proposed filtering techniques.
3. An extensive experimental study showing the effect of various combinations of signal behavior and precision requirements on the effectiveness of the filtering techniques, using both real and synthetic data sets.
4. The demonstration that slide filters do a better job of data reduction than swing filters, which in turn generally outperform previously introduced cache and linear filters. Because of the slightly lower overhead of the swing filters, it may be favored over the slide filter for applications that are extremely overhead-sensitive.

The outline of the rest of this paper is as follows. Section 2 introduces the problem and provides some background on filtering by piecewise linear functions. Sections 3 and 4 give the design details of the swing and slide filters, respectively. Section 5 presents the experiments and results. Related work is discussed in Section 6. Finally, Section 7 concludes the paper.

2. ONLINE COMPRESSION

2.1 Problem Statement and Notations

Given a data signal in the form of an on-line sequence of discrete data points (t_j,X_j), where $j\in[1,n]$ and X_j is a d-dimensional vector $(x_{1j},x_{2j},\ldots x_{dj})$, we wish to approximate this signal using a piece-wise linear function, such that the error for each dimension x_i in each of the original data points does not exceed some preset value ε_i representing the precision width, $i\in[1,d]$. The goal is to record only the successive line segments, and not the individual data points, thereby reducing the overhead of recording the signal.

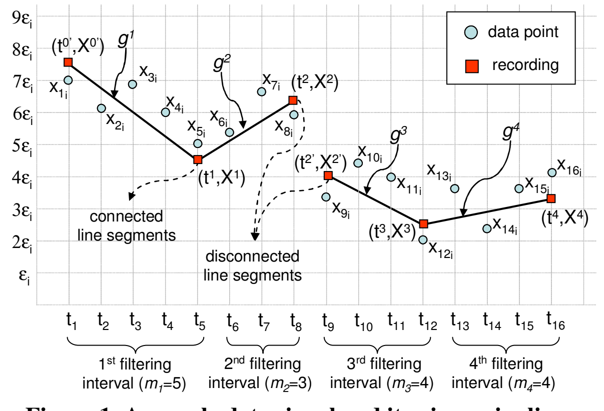
Moreover, if the approximated signal is to be sent from a transmitter to a receiver, the receiver should not be *lagging* behind the transmitter by a number of data points more than m_{max_lag}.

Note that the error constraint we choose (L_∞ metric) guarantees a certain quality level for *each* data point. This constraint is commonly used in the literature on online filtering techniques (e.g. [10,15,16,18,21]).

We assume that K line segments $(g^1,g^2,\ldots g^K)$ will be generated, where g^1 can represent the data points $((t_1,X_1),(t_2,X_2),\ldots(t_{j_1},X_{j_1}))$ and g^k, where $k\in[2,K]$, can represent the data points $((t_{j_{k-1}+1},X_{j_{k-1}+1}),(t_{j_{k-1}+2},X_{j_{k-1}+2}),\ldots(t_{j_k},X_{j_k}))$, and hence $j_K=n$. We denote the value j_1 as m_1 and the value $(j_k - j_{k-1} + 1)$ as m_k (i.e. m_k is the number of data points approximated by g^k).

We distinguish between two classes of the piece-wise linear functions used for approximation. These functions can either be in the form of *connected line segments* or *disconnected line segments.* In the former case, only one recording needs to be made per line segment, unlike the latter case, where two recordings are needed to define each of the segments. Disconnected line segments have, however, the potential to represent the original variable with fewer segments, (and thus fewer recordings), since they have an added degree of flexibility in choosing the starting location of each line segment.

We refer to the points of the original signal as the *data points.* We refer to the endpoints of the line segments as the *recordings*. If $g^{(k-1)}$ and g^k are disconnected, $k\in[2,K]$, then the recording at the beginning of g^k is denoted by $(t^{(k-1)'},X^{(k-1)'})$ and that at the end of g^k is denoted by (t^k,X^k). If $g^{(k-1)}$ and g^k are connected, then there is one recording for g^k at its end denoted by (t^k,X^k). The two recordings for g^1 are denoted by $(t^{0'},X^{0'})$ and (t^1,X^1). When a data point is not recorded, we say that it is *filtered* out. We refer to the interval during which the observed data points can be represented by a particular line segment as a *filtering interval.* There are K filtering intervals, where the kth interval is defined by $[t_1,t_{j_1}]$ when k=1 and $[t_{j_{k-1}+1},t_{j_k}]$ when $k\in[2,K]$. Finally, we use the notation $V_d(i,v)$ to denote a d-dimensional vector whose all dimensions are zeroes except the x_i dimension whose value is v. For example $V_4(3,5)$ = (0,0,5,0), and (9,9,9,9)- $V_4(3,5)$ = (9,9,4,9). Figure 1 shows a sample signal and a possible piece-wise linear approximation illustrating most of the notations described above.



$9\varepsilon_i$
$8\varepsilon_i$
$7\varepsilon_i$
$6\varepsilon_i$
$5\varepsilon_i$
$4\varepsilon_i$
$3\varepsilon_i$
$2\varepsilon_i$
ε_i
data point
recording
$(t^{0'},X^{0'})$
g^1
g^2
g^3
g^4
x_{1_i} x_{2_i} x_{3_i} x_{4_i} x_{5_i} x_{6_i} x_{7_i} x_{8_i} x_{9_i} x_{10_i} x_{11_i} x_{12_i} x_{13_i} x_{14_i} x_{15_i} x_{16_i}
(t^2,X^2)
(t^1,X^1)
$(t^{2'},X^{2'})$
(t^3,X^3)
(t^4,X^4)
connected line segments
disconnected line segments
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16}
1st filtering interval (m_1=5)
2nd filtering interval (m_2=3)
3rd filtering interval (m_3=4)
4th filtering interval (m_4=4)

Figure 1. A sample data signal and its piece-wise linear approximation (projected on the *t*-x_i plane)

2.2 Earlier Approaches

For piece-wise constant approximation, the simplest solution is to use a *cache filter*. A cache filter predicts that the next incoming data point will have the same values as the previous one, within the error bound of ε_i for each dimension i, $i \in [1,d]$. As long as the incoming data points satisfy the error constraint, the prediction is considered valid, and no new recordings are made. An incoming data point is recorded only if it violates the error constraint. This approach was the one considered in [21], while two of its variations were presented in [18], where each generated horizontal line segment is determined by either the midrange or the mean of the data points it represents rather than only by the first of these data points.

For piece-wise linear approximation, the intuitive approach is to use what we refer to as a *linear filter*. The idea of the linear filter was presented in [10,15,16]. Instead of predicting that new data points have values close to those of the previous ones, a linear filter predicts that they will always fall in the proximity of a line segment, which is not necessarily horizontal. The slope of the line is defined by the first two data points it represents. Whenever a new data point falls more than ε_i units away from the predicted line segment, for any dimension i, $i \in [1,d]$, a new line segment is started. Linear filters can produce connected or disconnected line segments. In the connected case, the current line segment is terminated by the point predicted by that line segment at the time of the last data point that it approximates, and that point and the new data point form the next line segment. In the disconnected case, the current line segment is terminated as before, but the new line segment is defined by the new data point and the next.

2.3 New Compression Mechanisms

We propose two new types of filters that produce superior results to cache and linear filters. The two new filtering algorithms we propose address both classes of approximating functions. *Swing filters* generate connected line segments, while *slide filters* generate a mixture of connected and disconnected line segments. The slide filter first attempts to get the benefits of disconnected line segments, and then, whenever possible, it generates connected line segments that do not sacrifice any of those benefits.

Both types of filters maintain *a set of candidate line segments* to approximate the current set of data points. The intuition is to postpone the selection of the line segment that represents these data points as long as possible. By doing so, the filter increases the probability that further data points can be represented without a new recording being necessary.

3. SWING FILTERS

In this section, we show the mechanisms used in the swing filters for filtering out incoming data points, and for selecting the best possible points for recording.

3.1 Filtering Mechanism

We explain the intuition behind the filtering mechanism of swing filters by contrasting them with linear filters.

As mentioned in Section 2.2, the linear filter always maintains a single line segment to approximate the data points. In contrast, the swing filter maintains a *set* of line segments for each filtering interval k, all starting from the same initial point. Along each dimension x_i, all the line segments lie between an upper hyperplane u_i^k and a lower hyperplane l_i^k, which are both perpendicular to the t-x_i plane. Therefore, each of the hyperplanes can be defined using two points only. Each line segment in the set can represent all the data points observed so far, within the specified error constraints ε_i, $i \in [1,d]$. Each time a new data point occurs whose x_i value lies between u_i^k and l_i^k or at most ε_i units above u_i^k or below l_i^k, *for every* $i \in [1,d]$; the data point is filtered out, and the set is reduced to maintain the invariant that all line segments in the set can represent all data points, including the new one. If a new data point with an x_i value falling outside the specified region, *for any* $i \in [1,d]$, a new recording is made and a new filtering interval is started.

Example 3.1

We consider the first five data points of a signal of the form (t_j,X_j), $j \in [1,5]$. Since, for each data point, the filtering mechanism is applied independently for each dimension, we only consider the x_i values of the five data points shown in Figure 2a. We assume, without loss of generality, that for these data points, x_i values are *always* the cause for starting a new filtering interval, regardless of which filter type is used. With the linear filter, after data points (t_1,X_1) and (t_2,X_2) have occurred, the approximating line is defined. (t_3,X_3) falls within ε_i units from the defined line, but (t_4,X_4) does not and thus requires a new recording (see Figure 2b).

In contrast, rather than immediately settling on one line when (t_2,X_2) arrives, a swing filter maintains a set of lines, bounded by upper and lower hyperplanes along each dimension (the hyperplanes for the x_i dimension, u_i^1 and l_i^1, appear as lines in the t-x_i plane). u_i^1 is defined by the pair of points (t_1,X_1) and $(t_2,X_2+V_d(i,\varepsilon_i))$, while l_i^1 is defined by the pair of points (t_1,X_1) and $(t_2,X_2-V_d(i,\varepsilon_i))$ (see Figure 3a). Any line segment between u_i^1 and l_i^1 can represent the first two data points in the ith dimension.

When (t_3,X_3) arrives, in order to maintain the invariant that all lines within the set can represent all data points so far, l_i^1 needs to be "swung up", and u_i^1 needs to be "swung down" --- hence the name "swing filter". The new l_i^1 is defined by the pair of points (t_1,X_1) and $(t_3,X_3-V_d(i,\varepsilon_i))$ (see Figure 3b). Lines below this new l_i^1 cannot represent point (t_3,X_3). Similarly, the new u_i^1 connects the pair of points (t_1,X_1) and $(t_3,X_3+V_d(i,\varepsilon_i))$. Lines above this new u_i^1 cannot represent (t_3,X_3).

While the linear filter of Figure 2b cannot represent (t_4,X_4), the swing filter can do so by "swinging down" u_i^1 (see Figure 3c), such that it connects (t_1,X_1) and $(t_4,X_4+V_d(i,\varepsilon_i))$. The lower line l_i^1 need not be changed to maintain the invariant for (t_4,X_4). To complete the example, (t_5,X_5) cannot be represented by the current set of lines, and thus a new recording needs to be made. □

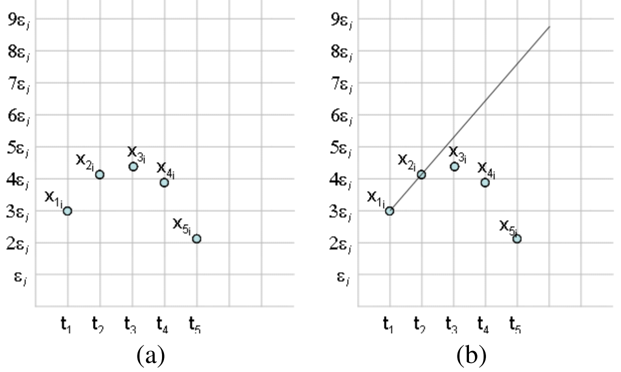


Figure 2. Data points pattern and the linear filter

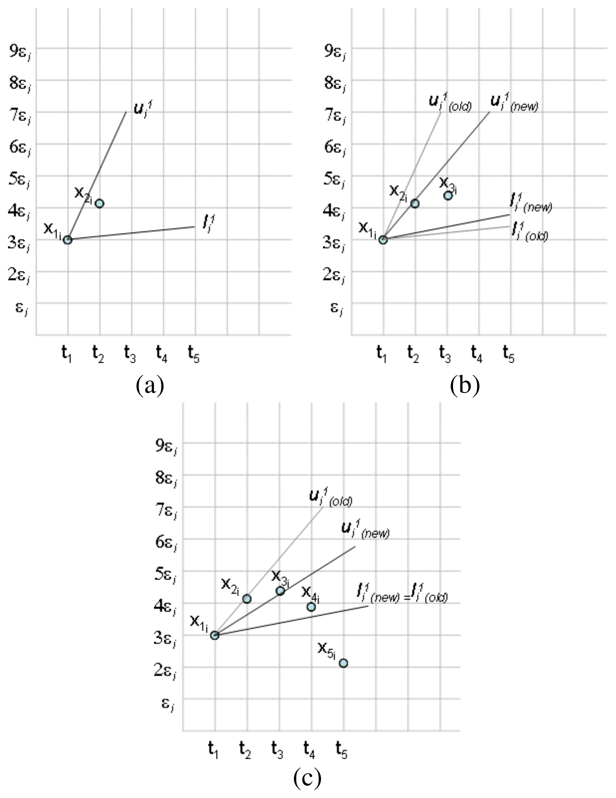


Figure 3. Filtering mechanism in swing filter

3.2 Recording Mechanism

Initially, the swing filter records the first incoming data point. Any later recording should represent the end point of the current approximating line segment and the start point of the new one. Hence, connected line segments are produced. A straightforward approach would be to simply record the last data point observed in each filtering interval. Instead, however, we choose a recording such that the generated line segment, g^k, for the just completed k^{th} filtering interval minimizes the mean square error for the data points observed in that interval. In that sense, after ensuring that we satisfied the error constraint and did our best effort in compression, we attempt to minimize the error further, as a secondary objective, where compression is the primary objective.

More formally, it is required to find the slope of g^k (call it a_{ig}^k) in the t-x_i plane, $i\in[1,d]$, such that (i) g^k minimizes the mean square error in the x_i dimension for the data points observed in the k^{th} filtering interval $((t_{j_{k-1}+1},X_{j_{k-1}+1}),(t_{j_{k-1}+2},X_{j_{k-1}+2}),\ldots(t_{j_k},X_{j_k}))$, (ii) g^k passes through the previous recording (t^{k-1},X^{k-1}), and (iii) a_{ig}^k occurs between the slopes of u_i^k and l_i^k (call them a_{iu}^k and a_{il}^k respectively). Once a_{ig}^k is known, and with the knowledge of the previous recording (t^{k-1},X^{k-1}), the new recording (t^k,X^k) can be obtained.

To get a_{ig}^k, consider that the equation for g^k in the t-x_i plane is

$$\hat{x}_i = a_{ig}^k t + b_{ig}^k \tag{1}$$

Since g^k has to pass through (t^{k-1},X^{k-1}), then

$$\hat{x}_i = a_{ig}^k (t - t^{k-1}) + x_i^{k-1} \tag{2}$$

where x_i^{k-1} is the x_i dimension of X^{k-1}. The mean square error can be minimized in each dimension independently. For the x_i dimension, it is given by

$$E_i^k = \sum_{j=j_{k-1}+1}^{j_k} (x_{ij} - \hat{x}_{ij})^2 \tag{3}$$

where $\hat{x}_{ij}$ is the value of $\hat{x}_i$ at $t=t_j$. Substituting (2) into (3) gives

$$E_i^k = \sum_{j=j_{k-1}+1}^{j_k} (x_{ij} - (a_{ig}^k (t_j - t^{k-1}) + x_i^{k-1}))^2 \tag{4}$$

By differentiating E_i^k w.r.t a_{ig}^k and equating to zero, and considering that $a_{ig}^k\in[a_{il}^k,a_{iu}^k]$, we get

$$a_{ig}^k = \max(\min(A_{ig}^k, a_{iu}^k), a_{il}^k) \tag{5}$$

where

$$A_{ig}^k = \frac{\sum_{j=j_{k-1}+1}^{j_k} (x_{ij} - x_i^{k-1})(t_j - t^{k-1})}{\sum_{j=j_{k-1}+1}^{j_k} (t_j - t^{k-1})^2} \tag{6}$$

Calculating the desired value of x_i^k at $t=t^k$ which minimizes E_i^k can now be performed by substituting (5) and (6) into (2). Note that the two summations in (5) can be calculated incrementally as each new data point is observed. Thus, there is no need to maintain the data points themselves; i.e. the space needed is O(1).

By repeating the process for all dimensions, we can find the optimal recording (t^k,X^k).

3.3 Algorithm and Analysis

With the above description of the filtering and recording mechanisms in the swing filter, the whole algorithm can be now outlined (Algorithm 1). getNext() is function that reads the next data point, and returns null when no more data points exist.

The state information that needs to be maintained by the swing filter is the initial point in the current filtering interval, k; the last observed data point; and the slopes of u_i^k and l_i^k, $i\in[1,d]$. In other words, the swing filter algorithm is O(1) in time and space complexity.

Algorithm 1: Swing Filter

```
// initialization
1. $(t_1,X_1)$ = getNext();$(t_2,X_2)$ = getNext();
2. Make a recording: $(t^{0'},X^{0'}) = (t_1,X_1)$;
3. Start a new filtering interval with $u_i^1$ passing through $(t_1,X_1)$
   and $(t_2,X_2+V_d(i,\varepsilon_i))$; and $l_i^1$ passing through $(t_1,X_1)$ and $(t_2,X_2-$
   $V_d(i,\varepsilon_i))$, for every dimension $x_i$, $i\in[1,d]$;
4. set $k$ = 1;
//main loop
5. while (true)
6.     $(t_j,X_j)$ = getNext();
7.     if $(t_j,X_j)$ is null or $(t_j,X_j)$ is more than $\varepsilon_i$ above $u_i^k$ or below $l_i^k$
       in the $x_i$ dimension for any $i\in[1,d]$    //recording mechanism
8.         Make a new recording: $(t^k,X^k)$, such that $t^k=t_{j-1}$, $x_i^k$ falls
           between $u_i^k$ and $l_i^k$, and $x_i^k$ minimizes $E_i^k$, for every
           dimension $x_i$, $i\in[1,d]$;
9.         Start a new filtering interval with $u_i^{(k+1)}$ passing through
           $(t^k,X^k)$ and $(t_j,X_j+V_d(i,\varepsilon_i))$; and $l_i^{(k+1)}$ passing through $(t^k,x^k)$
           and $(t_j,X_j-V_d(i,\varepsilon_i))$;
10.        set $k = k+1$;
11.        if $(t_j,X_j)$ is null    //end of signal
12.            return;
13.    else    //filtering mechanism
14.        for each dimension $x_i$, $i\in[1,d]$
```

```
15.        if $(t_j,X_j)$ falls more than $\varepsilon_i$ above $l_i^k$ in the $x_i$ dimension
16.            "Swing up" $l_i^k$ such that it passes through $(t^k,x^k)$ and
               $(t_j,X_j-V_d(i,\varepsilon_i))$;
17.        if $(t_j,X_j)$ falls more than $\varepsilon_i$ below $u_i^k$ in the $x_i$ dimension
18.            "Swing down" $u_i^k$ such that it passes through $(t^k,x^k)$
               and $(t_j,X_j+V_d(i,\varepsilon_i))$;
```

We note that if the number of data points observed during a certain filtering interval reaches the maximum allowable value by the receiver m_{max_lag}, then the swing filter can simply drop its maintained set of candidate line segments except for one (e.g. the line segment minimizing the mean square error). The filter will then update the receiver with the line segment it kept, and proceeds as a standard linear filter until the end of the filtering interval. For the next interval, it will switch back to proceeding as described in Algorithm 1.

3.4 Proof of Correctness

Theorem 3.1 *All the original data points of a signal compressed using the swing filter occur within the error constraint from the generated piece-wise linear approximation.*

Proof. It is obvious that for every filtering interval k, the first two data points are within ε_i from each of u_i^k, l_i^k, and g^k (since g^k is guaranteed to occur between u_i^k and l_i^k, as indicated in line 8 in Algorithm 1) for every dimension x_i, $i\in[1,d]$. If we assume that the first m data points in the k^{th} filtering interval are within ε_i from u_i^k, l_i^k, and g^k, then based on the method used to adjust u_i^k and l_i^k when the $(m+1)^{\text{th}}$ data point arrives (lines 14-18 in Algorithm 1), we can conclude that the new versions of u_i^k and l_i^k will be within ε_i from the $(m+1)^{\text{th}}$ data point, and will both occur between the old versions of u_i^k and l_i^k, thereby they will also be within ε_i from the first m data points. Consequently, g^k will still be guaranteed that it is within ε_i from the first m data points. By mathematical induction, all data points observed in any filtering interval k will be within ε_i from g^k for every dimension x_i, $i\in[1,d]$. □

4. SLIDE FILTERS

Slide filters are different from swing filters in that they may generate disconnected line segments as an approximation for the original data points. This gives them more flexibility when choosing line segments, at the expense of having to make two recordings for a single line segment if it is disconnected from its neighboring segments. In what follows, we explain the filtering and recording mechanisms used by the slide filters.

4.1 Filtering Mechanism

Similar to the swing filter, the slide filter maintains a set of lines which occur between an upper hyperplane u_i^k and a lower hyperplane l_i^k for each dimension x_i and filtering interval k. Unlike the swing filter, the lines need not start from the end point of the previous line segment. This allows the slide filter to have a larger set of lines and thus a higher probability to accommodate more incoming points, without the need for a new recording.

Also similar to the swing filter, a new data point is filtered out if it occurs between u_i^k and l_i^k, is above u_i^k by at most ε_i, or is below l_i^k by at most ε_i in the x_i dimension, for every $i\in[1,d]$. Otherwise, a recording is made and a new filtering interval is started. With the arrival of each new data point, u_i^k and l_i^k are potentially adjusted, $i\in[1,d]$.

The following two lemmas provide the foundation for finding the new u_i^k and l_i^k when a new data point is observed, $i\in[1,d]$.

Lemma 4.1 *Consider a sequence of m data points $((t_{j_1},X_{j_1}),(t_{j_2},X_{j_2}),...(t_{j_m},X_{j_m}))$, where there exists a hyperplane that is perpendicular to the t-x_i plane and within ε_i from all the m data points in the x_i dimension. If u_i (l_i) is a hyperplane with the following properties*

(P1) perpendicular to the t-x_i plane

(P2) passing through a pair of points $(t_{j_h},X_{j_h}-V_d(i,\varepsilon_i))$ and $(t_{j_l},X_{j_l}+V_d(i,\varepsilon_i))$ ($(t_{j_h},X_{j_h}+V_d(i,\varepsilon_i))$ and $(t_{j_l},X_{j_l}-V_d(i,\varepsilon_i))$), such that $t_{j_1}\le t_{j_h}<t_{j_l}\le t_{j_m}$

(P3) having the minimum (maximum) slope (i.e. dx_i/dt) among all hyperplanes having properties (P1) and (P2)

Then, u_i (l_i) also has the following two properties

(P4) within ε_i from all m data points in the x_i dimension

(P5) higher (lower) than any other hyperplane having properties (P1) and (P4) in the x_i dimension for any $t>t_{j_m}$

Proof. Assume that u_i has the properties (P1)-(P3), but not (P4). Let (t_j,X_j) be some data point, where u_i is more than ε_i below or above it in the x_i dimension. If $t_j<t_{j_l}$ and u_i is more than ε_i below (t_j,X_j) in the x_i dimension, or $t_j>t_{j_h}$ and u_i is more than ε_i above (t_j,X_j) in the x_i dimension, then there exists a hyperplane $u_i^{k'}$ with properties (P1) and (P2) that has a smaller slope than that of u_i^k. In particular, $u_i^{k'}$ will pass through points $(t_j,X_j-V_d(i,\varepsilon_i))$ and $(t_{j_l},X_{j_l}+V_d(i,\varepsilon_i))$, or $(t_{j_h},X_{j_h}-V_d(i,\varepsilon_i))$ and $(t_j,X_j+V_d(i,\varepsilon_i))$ respectively. This is a contradiction to property (P3) for u_i^k.

If $t_j<t_{j_h}$ and u_i^k is more than ε_i above (t_j,X_j) in the x_i dimension, or $t_j>t_{j_l}$ and u_i^k is more than ε_i below (t_j,X_j) in the x_i dimension, then there will not exist any hyperplane with property (P1) that is within ε_i from data points (t_h,X_h), (t_l,X_l) and (t_j,X_j), which is a contradiction to the description of the m considered data points. From the previous two contradictions, we conclude that data point (t_j,X_j) does not exist, and u_i^k has the property (P4).

Now assume that u_i^k has the properties (P1)-(P3), but not (P5). Then, from the description of the m considered data points, there has to exist another hyperplane u'_i that has properties (P1) and (P4) and is higher than any other hyperplane with properties (P1) and (P4) (including u_i) in the x_i dimension for *some* $t>t_{j_m}$. If u'_i does not have the property (P2), then we can obtain another hyperplane u''_i by rotating u'_i counter-clockwise (w.r.t the t-x_i plane) around the $t=t_i$ axis, for any $t_i\in[t_{j_1},t_{j_m}]$ such that u''_i does not pass through any points of the form $(t_{j_w},X_{j_w}-V_d(i,\varepsilon_i))$, where $t_{j_1}\le t_{j_w}<t_j\le t_{j_m}$, or of the form $(t_{j_w},X_{j_w}+V_d(i,\varepsilon_i))$, where $t_{j_1}\le t_j<t_{j_w}\le t_{j_m}$. u''_i will have the properties (P1) and (P4) and will be higher than u'_i in the x_i dimension for any $t>t_{j_m}$, which is a contradiction. Thus, u'_i must have the property (P2).

Furthermore, since u'_i has the property (P4), then at $t=t_{j_h}$, u'_i is higher than or equal to u_i in the x_i dimension. Since u_i has the minimum slope among hyperplanes having properties (P1) and (P2), then the slope of u'_i is greater than or equal to that of u_i. Since u'_i is different from u_i, then if they have the same slope, u'_i must be higher than u_i at $t=t_{j_h}$ and $t=t_{j_l}$ in the x_i dimension, which contradicts the property (P4) for u'_i. Then the slope of u'_i must be greater than that of u_i. However, this implies that u'_i is higher than u_i at $t=t_{j_l}$ in the x_i dimension, which also contradicts the property

(P4) for u'_i. Therefore, u'_i does not exist and u_i has the property (P5). The proof that if l_i has the properties (P1)-(P3), then it also has the properties (P4) and (P5) is quite similar. □

Lemma 4.2 *Referring to the properties defined in Lemma 4.1, given a sequence of m data points $((t_{j_1},X_{j_1}),(t_{j_2},X_{j_2}),...(t_{j_k},X_{j_m}))$, if there exists a hyperplane u_i (l_i) with the properties (P1), (P2), and (P4), then u_i (l_i) also has the properties (P3) and (P5)*

Proof. Assume that u_i has the properties (P1), (P2), and (P4), but not (P3). Let u'_i be a hyperplane that has properties (P1) and (P2) (i.e. it passes through a pair of points $(t_{j_{h'}},X_{j_{h'}}-V_d(i,\varepsilon_i))$ and $(t_{j_{l'}},X_{j_{l'}}+V_d(i,\varepsilon_i))$, such that $t_{j_1}\leq t_{j_{h'}}<t_{j_{l'}}\leq t_{j_m}$), and that the slope of u'_i is smaller than that of u'_i. Since u_i has the property (P4), then it has to be higher than or equal to u'_i at $t=t_{j_{h'}}$ and lower than or equal to u'_i at $t=t_{j_{l'}}$ in the x_i dimension. However, this implies that the slope of u_i is smaller than or equal to that of u_i, which is a contradiction. Thus, u'_i does not exist and u_i has the property (P3).

Now, assume that u_i has the properties (P1), (P2), and (P4), but not (P5). Let u''_i be a hyperplane that has properties (P1) and (P4), and is higher than u_i for *some* $t>t_{j_m}$. Since u''_i has the property (P4), then u''_i has to be higher than or equal to u_i at $t=t_{j_h}$ and lower than or equal to u_i at $t=t_{j_l}$ in the x_i dimension. However, this implies that the slope of u''_i is smaller than or equal to that of u_i. Thus, at $t>t_{j_m}$, u''_i must be lower than or equal to u_i, which is a contradiction. Thus, u_i does not exist and u_i has the property (P5). The proof that if l_i has the properties (P1), (P2), and (P4), then it also has the properties (P3) and (P5) is quite similar. □

Considering the kth filtering interval, Lemma 4.1 shows how to limit the search space for u_i^k (l_i^k). In particular, u_i^k (l_i^k) is the hyperplane with the minimum (maximum) slope (property (P3)) in the set of hyperplanes defined by properties (P1) and (P2). We will refer to this limited set as U_i^k (L_i^k). Lemma 4.2 shows that if the new data point is within ε_i in the x_i dimension from the existing u_i^k (l_i^k), then u_i^k (l_i^k) need not be adjusted, and thus the search in U_i^k (L_i^k) is not even needed.

We will shortly show how we can narrow the search space even further. However, we first explain the details of the filtering mechanism based on Lemmas 4.1 and 4.2 through an example.

Example 4.1

We again consider the pattern of data points shown in Figure 2a. We also only consider the x_i dimension, for the same reasons explained in Example 3.1.

After the data points (t_1,X_1) and (t_2,X_2) arrive, the sets U_i^1 and L_i^1 contain one line each, being u_i^1 and l_i^1 respectively. u_i^1 is defined by the two points $(t_1,X_1-V_d(i,\varepsilon_i))$ and $(t_2,X_2+V_d(i,\varepsilon_i))$, while l_i^1 is defined by $(t_1,X_1+V_d(i,\varepsilon_i))$ and $(t_2,X_2-V_d(i,\varepsilon_i))$ (see Figure 4a). After the arrival of (t_3,X_3), the lines u_{i1}^1 and u_{i2}^1 are added to U_i^1, where u_{ij}^1 connects $(t_j,X_j-V_d(i,\varepsilon_i))$ and $(t_3,X_3+V_d(i,\varepsilon_i))$, $j\in[1,2]$. Based on Lemma 4.1, the new u_i^1 is selected as the line with the minimum slope among u_i^1, u_{i1}^1, and u_{i2}^1, which is u_{i2}^1 in this case. Similarly, the new l_i^1 is selected as the highest of l_i^1, l_{i1}^1 and l_{i2}^1 (constituting the new L_i^1), where l_{ij}^1 is the line defined by the pair of points $(t_j,X_j+V_d(i,\varepsilon_i))$ and $(t_3,X_3-V_d(i,\varepsilon_i))$, $j\in[1,2]$. l_{i1}^1 is selected in this case (see Figure 4b). Adjusting the lines u_i^1 and l_i^1 does not involve rotations around the initial point, and thus they rather “slide” than “swing” --- hence the name “slide filter”.

The data point (t_4,X_4), as seen in Figure 4c, can already be

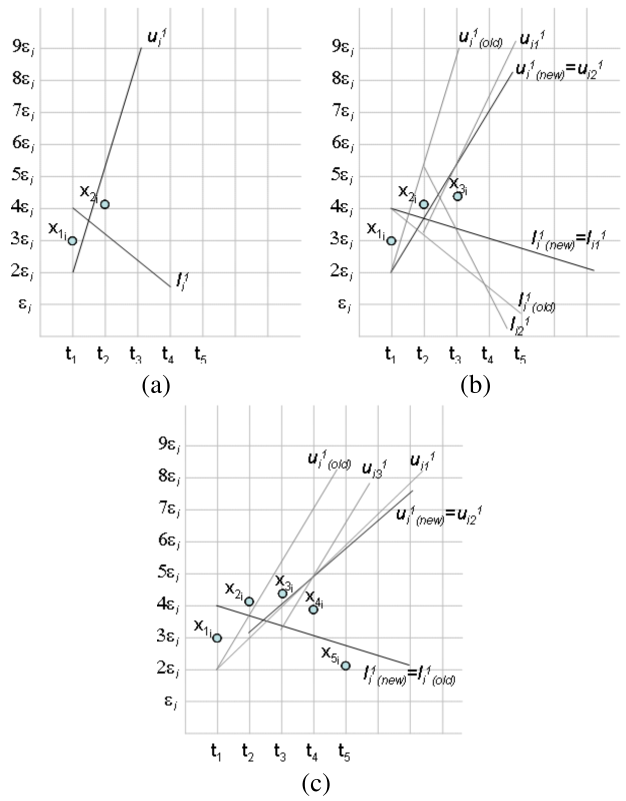


Figure 4. Filtering mechanism in slide filter

represented by l_i^1. l_i^1 has the properties (P1), (P2) and (P4), thereby, based on Lemma 4.2, it can be directly used as the new l_i^1. It is also guaranteed to have the maximum slope among all the lines in L_i^1. u_i^1, however, needs to be adjusted to represent (t_4,X_4). In the same way as described above, the lines u_{ij}^1, $j\in[1,3]$ are constructed, and then u_{i2}^1, being the lowest of them and u_i^1, is selected as the new u_i^1. Finally, (t_5,X_5) is less than ε_i below l_i^1 in the x_i dimension (see Figure 4c), and thus can be represented by it. Recall that (t_5,X_5) could not be represented by the swing filter (see Figure 3c). □

Optimization: The strategy for updating u_i^k (l_i^k) described so far involves checking *all* the data points observed in the current filtering interval, whenever a new data point arrives and invalidates the current u_i^k (l_i^k). It turns out that we can do much better. In fact, it is sufficient to check the points on the *convex hull* of the observed data points, as will be shown in the following lemma. The significance of this optimization is that the number of points on the convex hull can be dramatically smaller than the total number of data points observed during a filtering interval.

Lemma 4.3 *To update u_i^k (l_i^k) during the kth filtering interval of the slide filter, such that the new u_i^k (l_i^k) satisfies properties (P1), (P2), and (P3), defined in Lemma 4.1; it is sufficient to check the points on the convex hull of the data points observed during that filtering interval along the ith dimension, $i\in[1,d]$.*

Proof. We will only prove the lemma for the case when we are searching for the new u_i^k on the arrival of a new data point (t_j,X_j) which invalidates the old u_i^k. The proof for the case of l_i^k should be similar. According to Lemma 4.1, the new u_i^k should be the minimum-slope hyperplane (P3) chosen from the old u_i^k and all hyperplanes, which are perpendicular to the t-x_i plane (P1) and passing through $(t_{j'},X_{j'}-V_d(i,\varepsilon_i))$ and $(t_j,X_j+V_d(i,\varepsilon_i))$ (P2), where $(t_{j'},X_{j'})$ is a data point observed in the current filtering interval.

To see why only the convex hull of the observed data points is relevant to us, let us first denote that hull in the ith dimension by H_i, and the convex hulls of the points of the form $(t_{j'},X_{j'}-V_d(i,\varepsilon_i))$ and $(t_{j'},X_{j'}+V_d(i,\varepsilon_i))$ by H_i^- and H_i^+ respectively.

Now, if point $(t_{j'},X_{j'})$ occurs inside H_i, then its corresponding u_i^k (call it $u_{ij'}^k$) which passes through $(t_{j'},X_{j'}-V_d(i,\varepsilon_i))$ and $(t_j,X_j+V_d(i,\varepsilon_i))$ can always be rotated around $(t_j,X_j+V_d(i,\varepsilon_i))$ clockwise (to decrease its slope) until it touches a vertex in H_i^- (call it $(t_{j''},X_{j''}-V_d(i,\varepsilon_i))$). The corresponding u_i^k (call it $u_{ij''}^k$) has a smaller slope than $u_{ij'}^k$, thereby overriding it. Thus, there is no need to check (or maintain) the data points observed inside H_i. □

Following from the proof of the above lemma, we can further conclude that even across the data points occurring at the vertices of H_i, we are only interested in one whose corresponding u_i^k is *tangent* to H_i^-. In particular, the one where that tangent cannot be rotated clockwise any further.

Hence, the filtering mechanism for the slide filter reduces to solving two key problems: the incremental maintenance of H_i, and finding the tangent to H_i^- from an outside point. Both problems are well-known in the area of computational geometry [3].

The incremental convex hull algorithm can be summarized as follows. Points on H_i are divided into two lists representing an *upper hull* and a *lower hull*, where the points in each list are sorted by time. The two lists overlap in their first and last points, being the first- and last-observed data point in the current filtering interval. When a new data point arrives, it is inserted at the end of both lists. Then each list is updated separately.

Updating a list is achieved by examining streaks of three consecutive points starting with the most recent, and then moving backwards. If the direction of the "turn" made at the middle point of the three examined points is opposite to the original turning direction for the list (it should be clockwise for the upper hull and anti-clockwise for the lower hull as we move forward in time), then that middle point is removed from the list. Once a streak of three points is reached where the middle point is not removed, the update process stops for that list. For more details about this algorithm, the reader is referred to [3].

To find the tangent to H_i^-, we can simply scan its vertices until we find the vertex that minimizes the slope of u_i^k. An even more efficient algorithm can be found in [6].

4.2 Recording Mechanism

For each filtering interval k, the set of candidate line segments, that can represent all the data points observed in that interval, are those segments occurring between u_i^k and l_i^k, for every $i\in[1,d]$. In other words, a candidate line segment must pass through the intersection of u_i^k and l_i^k, $i\in[1,d]$. For the first filtering interval $[t_1,t_{j_1}]$, the generated line segment g^1 is chosen such that it minimizes the mean square error for the data points observed during that interval along each dimension x_i, $i\in[1,d]$. This is achieved exactly in the same way described in Section 4.1, where the slope of g^1 is decided independently for each dimension. The start point of g^1 occurs at $t=t_1$, while its end point is only decided after the second filtering interval $[t_{j_1+1},t_{j_2}]$ ends. By delaying that decision until the end of the second filtering interval, we might be able to generate two connected line segments rather than two disconnected ones. The criteria for generating connected line segments and the way the connection point is chosen will be described shortly. If the two line segments (g^1 and g^2) could not be connected, then g^1 will end at $t=t_{j_1}$ and g^2 will start at $t=t_{j_1+1}$, such that it minimizes the mean square error of the data points observed in the second filtering interval. The generated line segments for the following filtering intervals are chosen in the same manner, where the end point of g^K occurs at $t=t_{j_K}=t_{j_n}$.

When the kth filtering interval ends at $t=t_{j_k}$, $k\in[2,K]$, we need to determine whether g^k can be chosen such that it intersects with $g^{(k-1)}$ or not. By that time, the start point and slope of $g^{(k-1)}$ are known. For each dimension x_i, there can be an interval $[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$ where g^k can intersect with $g^{(k-1)}$, such that they can represent all the data points in the $(k-1)$th and kth filtering intervals within an error bounded by ε_i in that dimension. The intersection point can be chosen at any time $t^{(k-1)}$ in the interval $[\alpha^{(k-1)},\beta^{(k-1)}]$ (if exists), which is the intersection of all the intervals $[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$, $i\in[1,d]$.

The following lemma shows when the interval $[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$ exists, and how to calculate it for every dimension x_i, $i\in[1,d]$. Before presenting the lemma, we will define some variables, which are also illustrated in Figures 5a and 5b. Let (1) $(t_i^k,V_d(i,x_i^k))$ be a point on the intersection of u_i^k and l_i^k, (2) $s_i^{(k-1)}$ and $q_i^{(k-1)}$ be the hyperplanes perpendicular to the t-x_i plane, passing through the intersection of u_i^k and l_i^k, and intersecting with $l_i^{(k-1)}$ and $u_i^{(k-1)}$ respectively at $t_{j_{(k-1)}}$, (3) c_i^k and $c_i^{k'}$ be the intersection times of $g^{(k-1)}$ with u_i^k and l_i^k respectively, (4) d_i^k and $d_i^{k'}$ be the intersection times of $g^{(k-1)}$ with $s_i^{(k-1)}$ and $q_i^{(k-1)}$ respectively (5) e_i^k and $e_i^{k'}$ be $\max(c_i^k,d_i^{k'})$ and $\max(c_i^{k'},d_i^{k'})$ respectively, and (6) f_i^k and $f_i^{k'}$ be the intersection times of $g^{(k-1)}$ with l_i^k and u_i^k respectively.

Lemma 4.4 *If $(t_i^k,V_d(i,x_i^k))$ is below (above) $g^{(k-1)}$, f_i^k ($f_i^{k'}$) is less than $t_{j_{(k-1)}}$, and l_i^k is above $l_i^{(k-1)}$ (u_i^k is below $u_i^{(k-1)}$) at $t=t_{j_{k-1}}$ in the x_i dimension, then there exists $\alpha_i^{(k-1)}=e_i^{(k-1)}$ ($\alpha_i^{(k-1)}=e_i^{(k-1)'}$) and $\beta_i^{(k-1)}=f_i^{(k-1)}$ ($\beta_i^{(k-1)}=f_i^{(k-1)'}$), such that g^k can be chosen to intersect with $g^{(k-1)}$ at any time $t^{(k-1)}\in[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$, while $g^{(k-1)}$ is within ε_i in the x_i dimension from all the data points in the interval $[t_{j_{(k-2)}+1},t^{(k-1)}]$ and g^k is within ε_i in the x_i dimension from all the data points in the interval $[t^{(k-1)},t_{j_k}]$*

Proof. We will only consider the case where $(t_i^k,V_d(i,x_i^k))$ is below $g^{(k-1)}$. The proof for the opposite case is quite similar. Let $(t^{(k-1)},X^{(k-1)})$ be the intersection point of $g^{(k-1)}$ and g^k, such that $t^{(k-1)}\in[e_i^{(k-1)},f_i^{(k-1)}]$, and consequently $t^{(k-1)}<f_i^{(k-1)}<t_{j_{(k-1)}}$. It follows that $g^{(k-1)}$ is used to approximate the data points in the interval $[t_{j_{(k-2)}+1},t^{(k-1)}]$, while g^k is used to approximate the data points in the interval $[t^{(k-1)},t_{j_{(k-1)}}]$ and those in the interval $[t_{j_{(k-1)}+1},t_{j_k}]$. By definition, $g^{(k-1)}$ is within ε_i in the x_i dimension from all the data points in the interval $[t_{j_{(k-2)}+1},t_{j_{(k-1)}}]$, which includes the interval $[t_{j_{(k-2)}+1},t^{(k-1)}]$. Since g^k intersects with $g^{(k-1)}$ at a time ($t^{(k-1)}$) between the intersection times of u_i^k and l_i^k with $g^{(k-1)}$ (c_i^k and f_i^k respectively), and since all of g^k, u_i^k and l_i^k intersect at a *later* time (t_i^k), then g^k is guaranteed to always occur between u_i^k and l_i^k. Therefore, g^k is within ε_i in the x_i dimension from all the data points in the interval $[t_{j_{(k-1)}+1},t_{j_k}]$. If $t^{(k-1)}>t_{j_{(k-1)}}$, then the interval $[t^{(k-1)},t_{j_{(k-1)}}]$ does not exist. Otherwise, since g^k intersects with $g^{(k-1)}$ at $t=t^{(k-1)}$, then g^k is between $u_i^{(k-1)}$ and $l_i^{(k-1)}$ at $t=t^{(k-1)}$. Since $(t_i^k,V_d(i,x_i^k))$ is below $g^{(k-1)}$ in the x_i dimension, then g^k has a smaller slope than those of $g^{(k-1)}$ and $u_i^{(k-1)}$, and thus is lower than $u_i^{(k-1)}$ in the x_i dimension at $t=t_{j_{(k-1)}}$. Also, since $t^{(k-1)}>\max(c_i^k,d_i^k)$, then g^k is higher than the highest of u_i^k and

$s_i^{(k-1)}$. But since $s_i^{(k-1)}$ intersects with $l_i^{(k-1)}$ at $t=t_{j_{(k-1)}}$, then g^k is guaranteed to be higher than or equal to $l_i^{(k-1)}$ in the x_i dimension at $t=t_{j_{(k-1)}}$. Therefore, g^k occurs between $u_i^{(k-1)}$ and $l_i^{(k-1)}$ in the interval $[t^{(k-1)},t_{j_{(k-1)}}]$, and thus is within ε_i in the x_i dimension from all the data points in that interval. □

Figure 5a shows the x_i dimension of a signal where $g^{(k-1)}$ and g^k cannot be connected because $(t_i^k,V_d(i,x_i^k))$ is below $g^{(k-1)}$ and $f_i^k<t_{j_{(k-1)}}$, but opposite to the requirement of Lemma 4.4, l_i^k is below $l_i^{(k-1)}$ at $t=t_{j_{(k-1)}}$. In contrast, $g^{(k-1)}$ and g^k can be connected in Figure 5b, where all the requirements of Lemma 4.4 are met: $(t_i^k,V_d(i,x_i^k))$ is below $g^{(k-1)}$, $f_i^k<t_{j_{(k-1)}}$, and l_i^k is above $l_i^{(k-1)}$ at $t=t_{j_{(k-1)}}$.

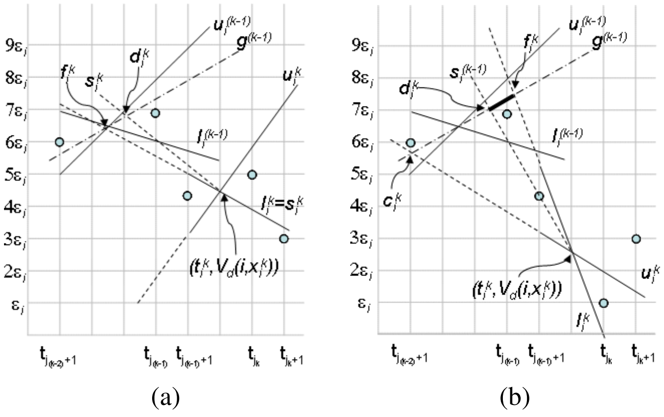


Figure 5. Recording mechanism in slide filter

4.3 Algorithm and Analysis

Based on the above discussion, we can now outline the algorithm of the slide filter (Algorithm 2).

Algorithm 2: Slide Filter

```
//initialization
1. $(t_1,X_1)$ = getNext();$(t_2,X_2)$ = getNext();
2. Start a new filtering interval with $u_i^1$ passing through $(t_1,X_1-V_d(i,\varepsilon_i))$ and $(t_2,X_2+V_d(i,\varepsilon_i))$, and $l_i^1$ passing through $(t_1,X_1+V_d(i,\varepsilon_i))$ and $(t_2,X_2-V_d(i,\varepsilon_i))$, for every $i\in[1,d]$;
3. **set** $k$=1;
//main loop
4. **while**(true)
5.    $(t_j,X_j)$ = getNext();
6.    **if** $(t_j,X_j)$ is null or $(t_j,X_j)$ falls more than $\varepsilon_i$ above $u_i^k$ or below $l_i^k$ in the $x_i$ dimension, for *any* $i\in[1,d]$;    //recording mechanism
7.       **if** ($k$>1)
8.          Calculate the interval $[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$ for each dimension $x_i$, $i\in[1,d]$, as described in Lemma 4.3;
9.          Calculate the interval $[\alpha^{(k-1)},\beta^{(k-1)}]$ as the intersection of all the intervals $[\alpha_i^{(k-1)},\beta_i^{(k-1)}]$, $i\in[1,d]$;
10.         **if** the interval $[\alpha^{(k-1)},\beta^{(k-1)}]$ exists
11.            **for each** dimension $x_i$, $i\in[1,d]$
12.               **let** $z_i^k$ be any point on the intersection of $u_i^k$ and $l_i^k$;
13.               **if** $z_i^k$ falls below $g^{(k-1)}$
14.                  Adjust $u_i^k$ and $l_i^k$ to intersect $g^{(k-1)}$ at $t=\alpha^{(k-1)}$ and $t=\beta^{(k-1)}$ respectively, while $u_i^k$ and $l_i^k$ still pass through $z_i^k$
15.               **else if** $z_i^k$ falls above $g^{(k-1)}$
16.                  Adjust $u_i^k$ and $l_i^k$ to intersect $g^{(k-1)}$ at $t=\beta^{(k-1)}$ and at $t=\alpha^{(k-1)}$, while $u_i^k$ and $l_i^k$ still pass through $z_i^k$;
17.      Calculate $a_{ig}^k$ (the slope of $g^k$) such that it is between $a_{iu}^k$ and $a_{il}^k$ and minimizes $E_i^k$, for *every* $i\in[1,d]$ ;
18.      **if** ($k$>1) and the interval $[\alpha^{(k-1)},\beta^{(k-1)}]$ exists
19.         Make a recording: $(t^{(k-1)},X^{(k-1)})$, which is the intersection point of $g^k$ and $g^{(k-1)}$ ;
20.      **else if** ($k$>1) and the interval $[\alpha^{(k-1)},\beta^{(k-1)}]$ does not exist
21.         Make two recordings: $(t^{(k-1)},X^{(k-1)})$, which is the point on $g^{(k-1)}$ at $t=t_{i_{(k-1)}}$ and $(t^{(k-1)'},X^{(k-1)'})$, which is the point on $g^k$ at $t=t_{j_{(k-1)}+1}$;
22.      **else if** ($k$=1)
23.         Make a recording: $(t^{0'},X^{0'})$, which is the point on $g^1$ at $t=t_1$;
24.      **if** $(t_j,X_j)$ is null   //end of signal
25.         Make a recording: $(t^k,X^k)$, which is the point on $g^k$ at $t=t_{(j-1)}$;
26.         **return**;
27.      **else**
28.         $(t_{j+1},X_{j+1})$ = getNext();
29.         Start a new filtering interval with $u_i^{(k+1)}$ passing through $(t_j,X_j-V_d(i,\varepsilon_i))$ and $(t_{j+1},X_{j+1}+V_d(i,\varepsilon_i))$, and $l_i^{(k+1)}$ passing through $(t_j,X_j+V_d(i,\varepsilon_i))$ and $(t_{j+1},X_{j+1}-V_d(i,\varepsilon_i))$, for *every* $i\in[1,d]$;
30.         **set** $k$=$k$+1;
31.   **else**    //filtering mechanism
32.      **for each** dimension $x_i$, $i\in[1,d]$
33.         Update the convex hull $H_i$;
34.         **if** $(t_j,x_j)$ falls more than $\varepsilon_i$ above $l_i^k$
35.            Construct $l_{ij'}^k$, for every point $(t_{j'},X_{j'})$ that is a vertex on $H_i$, such that $l_{ij'}^k$ passes through $(t_{j'},X_{j'}+V_d(i,\varepsilon_i))$ and $(t_j,X_j-V_d(i,\varepsilon_i))$;
36.            Adjust $l_i^k$ to be the highest of $l_i^k$ and $l_{ij'}^k$ for $t$>$t_j$, for every $j'$, where $(t_{j'},X_{j'})$ is a vertex on $H_i$;
37.         **if** $(t_j,x_j)$ falls more than $\varepsilon_i$ below $u_i^k$
38.            Construct $u_{ij'}^k$, for every point $(t_{j'},X_{j'})$ that is a vertex on $H_i$, such that $u_{ij'}^k$ passes through $(t_{j'},X_{j'}-V_d(i,\varepsilon_i))$ and $(t_j,X_j+V_d(i,\varepsilon_i))$;
39.            Adjust $u_i^k$ to be the lowest of $u_i^k$ and $u_{ij'}^k$ for $t$>$t_j$, for every $j'$, where $(t_{j'},X_{j'})$ is a vertex on $H_i$;
```

During each filtering interval, the slide filter needs to maintain the slopes of u_i^k and l_i^k, in addition to the data points representing the vertices of the convex hulls of the data points observed so far in that interval – one convex hull for each dimension. Our experiments have shown that the number of such vertices typically remains very small regardless of how many data points are observed in the same filtering interval. If we denote this number by m_H, then the time and space complexity of the slide filter are both $O(m_H)$ (recall that the incremental update of the convex hull is linear in its number of vertices).

We note that if the number of data points observed since the last receiver update reaches the maximum value m_{max_lag}, then the slide filter can handle this situation in the same way described for the swing filter.

4.4 Proof of Correctness

Theorem 4.1 *All the original data points of a signal compressed using the slide filter occur within the error constraint from the generated piece-wise linear approximation.*

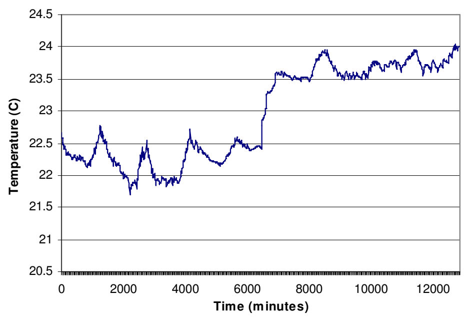


Figure 6. Sea surface temperature

Proof. Considering disconnected line segments only, it is obvious that for every filtering interval *k*, the first two data points are within ε_i from each of u_i^k, l_i^k, and g^k (since g^k is guaranteed to occur between u_i^k and l_i^k) for every dimension x_i, $i\in[1,d]$. If we assume that the first *m* data points in the k^{th} filtering interval are within ε_i from u_i^k, l_i^k, and g^k, then based on the method used to adjust u_i^k and l_i^k (lines 36 and 39 respectively in Algorithm 2) and Lemmas 4.1-4.3, we can conclude that the first *m*+1 data points will also be within ε_i from u_i^k, l_i^k, and g^k. By mathematical induction, all data points observed in any filtering interval *k* will be within ε_i from g^k for every dimension x_i, $i\in[1,d]$. Considering connected line segments, the slide filter connects the line segments $g^{(k-1)}$ and g^k only when the conditions specified in Lemma 4.3 are met (lines 8-10 in Algorithm 2), and their intersection point is selected also as specified in Lemma 4.3 (lines 11-19 in Algorithm 2). Thus, Lemma 4.3 guarantees that all the data points in the filtering intervals *k*-1 and *k* are within ε_i from either $g^{(k-1)}$ or g^k. □

5. EXPERIMENTS AND RESULTS

5.1 Experimental Setup

In our experimental study, we use both real data and synthetic data to evaluate the effectiveness of the different filters. The real data is obtained from the oceanography domain. It consists of 1285 data points for the sea surface temperature sampled at a 10 minutes interval [20]. Moreover, using the synthetic data allowed us to carefully study the impact of certain properties, which the data signals may exhibit, on the effectiveness of the filters.

In the experiments, we compare between four different types of filters: (1) cache filters, (2) linear filters (generating connected segments), (3) swing filters, and (4) slide filters.

We report the compression ratio achieved by each filter, which is calculated by dividing the number of recordings needed when no filtering is used by that when filtering is used. We also report the average error of the signals generated by each filter. The average error is computed as the sum of errors for each sample divided by the number of samples. Finally, we present an experiment, which shows the processing time needed per data point when the different types of filters are used.

We studied the effect of several parameters, including (1) the prescribed precision width, which is measured as a percentage of the signal's range (difference between maximum and minimum values), (2) the signal behaviour (e.g. the degree of monotonicity and the magnitude of change per data point), and (3) the dimensionality (e.g. the number of dimensions and the degree of correlation between the different dimensions). In our graphs, we generally use a logarithmic scale for the x-axis whenever we wish to examine a wide range of values for the parameter under study.

The experiments were conducted on a Pentium 4 machine with a 3 GHz processor and 1GB RAM. In general, we have set m_{max_lag} to a large value, to be able to assess the filters' full compression power, especially for applications that give higher priority to compression over timeliness. Still, however, other lag-sensitive applications can set m_{max_lag} to any arbitrary value.

5.2 Effect of Precision Width

In this experiment, we show the effect of varying the precision width on the filters' compression ratio and average error for the signal representing the sea surface temperature. Figure 6 shows the original signal. As can be observed, it continuously goes up and down with no regular pattern.

The results shown in Figure 7 indicate that the slide filter is superior to the other filters in terms of the compression ratio. Its improvement over the filter with the lowest compression ratio (linear filter) ranges from 21% to an astounding 1867% when the precision width is 10% of the range. The swing filter follows the slide filter in performance. The cache filter comes next preceding the linear filter. This is because the value of the sea surface temperature remains fixed frequently enough to give an advantage to the cache filter. Note that the compression ratio is always above 1 even though it may not be clear in the figure.

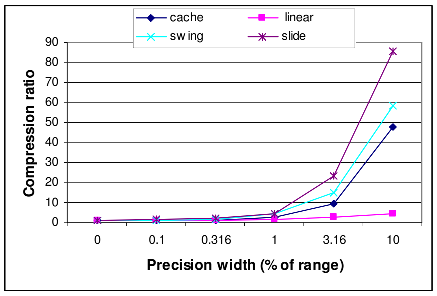


Figure 7. Compression ratio for the sea surface temperature

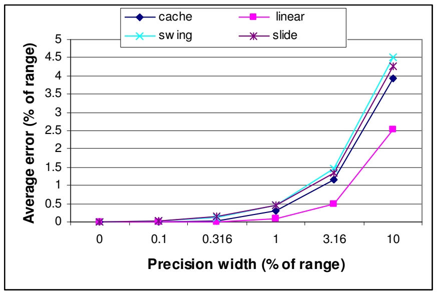


Figure 8. Average error for the sea surface temperature

Figure 8 shows that the average error for the slide, swing, and cache filters is almost identical, and is a little lower for the linear filter (which also has the least compression ratio). We further note that the average error for all the filters is generally far below the prescribed precision width. For example, when the prescribed precision width is 10% of the range, the average error for the swing filter (highest across all filters) is only 4.5% of the range.

5.3 Effect of Signal Behavior

This set of experiments uses synthetic data to show the effect of varying the signal behavior on the compression ratio when the

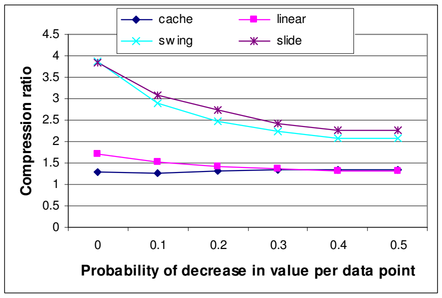


Figure 9. Effect of the degree of monotonicity

different filters are used. We generated the synthetic signals such that they follow a random-walk-like model. The value for each data point can be lower than or higher than that of the previous data point according to the probabilities *p* and (1-*p*) respectively. The magnitude of increase/decrease in the value is given by a uniform distribution U(0,*x*), where *x* is a configurable parameter.

Figure 9 shows the effect of the degree of the signal's monotonicity on the compression ratio. The probability *p* is varied from 0 to 0.5, while *x* is set to 400% of the precision width. At the two extremes of the graph, the signal is either monotonically increasing or continuously oscillating. The figure clearly shows that the slide and swing filters achieve higher compression ratios than the linear and cache filters. The improvement of the slide filter (best) over the cache filter (worst) ranges from 70% when *p*=0.5 to about 200% when *p*=0. The cache filter is the least sensitive to the fluctuations in the signal's value, whereas the other filters perform better when the value is mostly changing in the same direction since such behavior is closer to the linear behavior they expect.

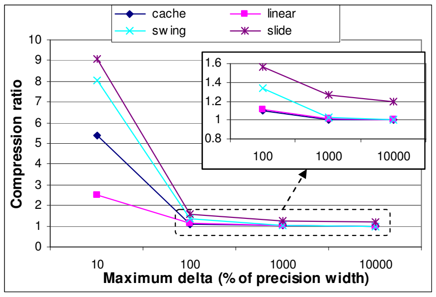


Figure 10. Effect of the magnitude of change per data point

Figure 10 shows the effect of varying the magnitude of maximum change per data point (*x*) from 10% to 10,000% of the precision width, where *p* is set to 0.5. This implies that the variable oscillates up and down with equal probability. As *x* increases, it becomes more difficult to represent many data points using the same line segment, and so the compression ratio decreases. However, the figure shows that the slide and swing filters consistently outperform the cache and linear filters. In terms of improving the compression ratio, the slide filter achieves an improvement over the linear filter ranging from 266% when *x*=10% down to 19.5% when *x*=10,000%. We note that when *x* is less than the precision width (e.g. *x*=10%), the cache filter performs better than the linear filter. In this case, the signal can keep oscillating around the same horizontal line segment without violating the error constraint, which is good for the cache filter. Moreover, the reason behind the high resilience of the slide filter to the sharp fluctuations in the signal's value (i.e. even when *x* is large) compared to the other filters is as follows. Even though the number of required segments increases with such fluctuations, the chances of connecting neighboring segments also increase.

5.4 Effect of Dimensionality

In this set of experiments, we study the effect of dimensionality on the filters' compression ratio. We also use synthetic data, where we consider signals having more than one dimension. The values for each dimension are generated in the same way as in Section 5.3.

Figure 11 shows that as the number of dimension increases, the achieved compression ratio decreases. This is expected because a new line segment has to be generated once the value in *any* dimension x_i is more than ε_i above or below the current line segment. With more dimensions, the likelihood that this event occurs gets higher (especially when the dimensions are completely independent as in the case of Figure 11). It is observed that the slide and swing filters still achieve the highest compression ratios, even with high dimensionality.

For the experiment reported in Figure 12, we generated a 5-dimensional signal, and varied the correlation between its five dimensions from 0.1 to 1. As expected, as the correlation increases, the dimensions tend to vary in a similar way. Thus, the likelihood that one of them requires starting a new line segment and not the others decreases. This results in generating less number of line segments, and thus a higher compression ratio. Figure 13 also demonstrates that the slide and swing filters still consistently outperform their counterparts.

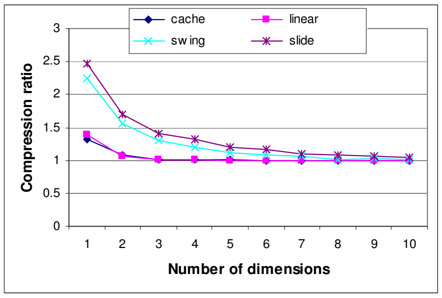


Figure 11. Effect of the number of dimensions

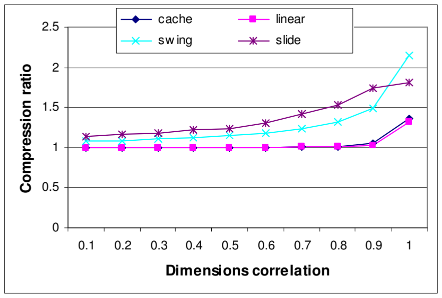


Figure 12. Effect of the correlation between dimensions

An interesting question is whether it is more effective to compress each dimension independently, or to compress the multiple dimensions together. In fact, it depends on how correlated they are. For example, from Figure 11, we find that compressing a single dimension independently using the slide filter can result in a compression ratio of 2.47. However, since independent compressions require recording the time information for the points generated for each dimension. In effect, this reduces the compression ratio. If we assume that the size of the time field is equal to the size of the dimension value, x_i, then for a d-dimensional signal, the compression ratio resulting from independent compressions should be the ratio for a single dimension multiplied by $(d+1)/2d$ to account for the redundancy in recording the time information. Thus for a 5-dimensioanl signal, the compression ratio for independent compressions should be $2.47\times(5+1)/(2\times5)=1.48$. From Figure 12, we find that when the correlation is above 0.7, the compression ratio exceeds 1.48; i.e., compressing the multiple dimensions together becomes more effective than compressing each dimension independently.

5.5 Filtering Overhead

To measure the filtering overhead, we used the sea surface temperature data, where we loaded all the data points into memory and then fed them into our filtering system once without performing any filtering and once for each filter type. In all cases, the total time for processing all the data points, repeated 10,000 times, is measured. Finally, we subtract the time taken when no filtering is applied from the time for each filter and divide by the number of data points processed to get the processing overhead per data point.

Note that the only parameter that may affect the processing time per data point is the size of the filtering interval in terms or how many data points it spans. Hence, to study the overhead, it is sufficient to run the filters on *any* signal while varying the precision width. This way, we will effectively be varying the average size of the filtering intervals – precisely what we need for this study. In other words, varying other parameters will not provide additional information. For example, varying the signal behavior will also ultimately result in varying the size of the filtering intervals. Moreover, if the signal is multi-dimensional, the same amount of work is done for each dimension. Correlated dimensions can only result in higher compression, which again implies larger filtering intervals on average.

Figure 13 shows how the processing time per data point changes by varying the precision width. In addition to showing the overhead of the four filters studied before, we also show here the overhead of the non-optimized slide filter (when the convex hull optimization is not used).

It is observed that all four filters, including swing and slide (the optimized version), are scalable w.r.t. the number of observed data points in the filtering interval. This was expected for the swing filter because its time complexity is O(1). For the slide filter, however, this is an interesting result because it shows that the number of vertices of its maintained convex hulls is almost constant regardless of how many data points are inside the hulls.

It is also worth noting that the overhead does not exceed 4µs per data point for the cache, linear, and swing filters, and about 8µs per data point for the slide filter. Again, this difference was expected because of the additional convex hull maintenance work the slide filter has to do. But more importantly, the two figures are sufficiently low for overhead-sensitive applications (e.g. sensor networks or cluster monitoring, where the wasted CPU cycles by the monitoring service should be minimal). Extremely overhead-sensitive applications may prefer the lower overhead of the swing filter over the higher compression power of the slide filter.

The figure also clearly shows the significance of optimizing the slide filter. In particular, its non-optimized version is not scalable with respect to the number of observed data points. It has

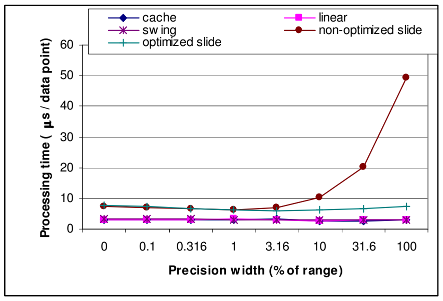


Figure 13. Filtering overhead for the sea surface temperature signal

to process each such data point whenever a new data point arrives, as opposed to processing the vertices of the convex hull only in the case of the optimized version.

6. RELATED WORK

The management of data streams resulting from monitoring and sensor network applications has been an active research area in the last few years. Much work has been directed towards finding techniques for data reduction in order to cope with the large sizes of collected data. Lazaridis et al. [18] propose an optimal on-line algorithm for constructing a piecewise constant approximation for a time series, as opposed to the more general piecewise linear approximation that we construct. The output of their algorithm corresponds to that of the cache filter presented in Section 2. Olston et al. [21] consider the problem of approximating aggregate values over multiple input streams. They propose an algorithm, which, given a desired precision for the aggregate value, adaptively adjusts the precision of the underlying individual input streams, such that the communication overhead is minimized. They only consider cache filters for filtering the input streams. Dilman et al. [10] propose two algorithms similar to the cache and linear filter algorithms for reducing the monitoring overhead in IP networks. They also study the statistical factors that affect the amount of savings for each monitored variable. In [15], Jain et al. propose using Kalman filters for approximating data streams. Kalman filters are a general framework for predicting the state of any process represented by the data stream, taking into consideration the measurement noise and uncertainty in state propagation. Kalman filters are general enough to model both the cache and linear filters, and even more complex models such as sinusoidal models. Choosing the most appropriate model requires, however, prior knowledge about the behavior of the monitored variable, which is not normally available. Kalman filters are also incapable of simulating the swing and slide filters since each of them maintain multiple prediction models simultaneously, i.e., the set of candidate line segments. The work in [23] is based on inserting load shedding operators inside the query execution plans for querying input data streams in order to handle peaks in the input data rates that the servers cannot cope with. They do not provide precision guarantees, but rather protect the servers from overwhelming data rates. Wu et al. [24] consider the approximation of financial data streams, where the data follows a repetitive pattern of waves. Therefore, the piece-wise linear approximation generated by their algorithm has a zigzag shape. The output is further pruned to get rid of noise-like line segments that are irrelevant to the stocks' general trends. Palpanas et al. [22] introduced the amnesic approximation of data streams, which allows arbitrary, user defined reduction of quality with time. The work in both [24] and [22] does not provide precision guarantees either. Keogh et al. [16] proposed the SWAB algorithm which merges an offline bottom-up technique for time series segmentation with an online technique similar to the linear filter. This work is complementary to our work as the swing and slide filters can replace the linear filter in the SWAB algorithm.

There have been other efforts for data reduction that do not directly depend on filtering. Deligiannakis et al. [9] attempt to find correlations between data streams collected from sensors, construct base signals that carry the important trends in them, and then only record the base signals and the relation between each stream and the base signals. The algorithm needs O($n^{1.5}$) time and O(n) space. It is assumed to run periodically after enough historical data is collected by the sensor. Guha et al [12] generalize the problem of histogram construction for infinite data streams. The goal of the histogram construction problem is to divide a data set into a given number of buckets and then represent the data set using the mean values of these buckets, such that the error in the approximation is minimized. The algorithm they propose is based on using a fixed-length sliding window of data points. In [4], Buragohain et al. also address the histogram construction problem. However they represent each bucket by a line segment rather than a single value. Madden et al. [19] introduce a new mechanism for in-network aggregation in ad-hoc sensor networks, where the execution of aggregate queries is distributed in the network, resulting in less communication overhead than the obvious centralized approach. The authors then extend their work to provide wavelet-based lossy compression of the data collected in sensor networks [14]. Again, the main difference between the above algorithms and ours is that they do not provide precision guarantees.

A significant number of Data Stream Management Systems have been introduced by the database community, including AURORA [1], COUGAR [25], NiagraCQ [7], NILE [13], TelegraphCQ [5] and STREAM [2]. Their common goal is to provide a general-purpose infrastructure for the efficient management of data streams. Several frameworks have been developed for system monitoring as well. Among them is WatchTower [17] which collects Windows performance counter data, and stores only the statistically interesting counters, or composite counters that summarize the behavior of many raw counters. Remos [11] is another system that collects and distributes resource information in grid environments across different querying entities. Pinpoint [8] is a monitoring system for J2EE applications that logs Java exceptions in J2EE application servers, and tries to derive from that information performance bottlenecks or component malfunctions. To the best of our knowledge, none of the currently available systems use techniques similar to ours for reducing the size of collected data.

7. CONCLUSIONS

We have presented two new filtering mechanisms that produce a piecewise linear approximation for an input multi-dimensional data stream with guarantees on both the quality of each data point and the lag between the transmitter and receiver. The two new mechanisms, the swing and slide filters, were shown to outperform previous methods of filtering by piecewise linear (and constant) approximation. We have evaluated the performance of these filters using a real data set from the oceanography domain, in addition to a wide variety of synthetic data sets to study the effect of the different types of signal behavior and precision requirements on the compression power of the proposed techniques. We have studied the effect of monotonic versus oscillatory behavior, smooth versus sharp fluctuations; and high-dimensionality versus low-dimensionality. We concluded that the slide filter provides the highest compression ratios in almost all the cases. We also showed that compressing highly-correlated dimensions together can be more effective than compressing each dimension independently. The overhead imposed by the filters was found to be minimal: a few microseconds per data point. Because of the relatively lower overhead of the swing filter compared to the slide filter, it (swing) can be more suitable for applications that are extremely overhead-sensitive. Finally, we have also proved, for both types of filters, that the error of each data point in the approximated signal is guaranteed to stay within the prescribed precision.

REFERENCES

[1] D. Abadiand et al. Aurora: A datastream management system (demonstration). In *SIGMOD* 2003.

[2] A. Arasu et al. STREAM: The Stanford stream data manager. *IEEE Data Engineering Bulletin,* 2003.

[3] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars. Computational Geometry. 3rd edition, Springer, 2008.

[4] C. Buragohain, N. Shrivastava, S. Suri. Space Efficient Streaming Algorithms for the Maximum Error Histogram. In *ICDE* 2007.

[5] Chandrasekaran et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR* 2003.

[6] B. Chazelle and D. P. Dobkin. Intersection of convex objects in 2 and 3 dimensions. Journal of the ACM, 34:1-27, 1987.

[7] J. Chen et al. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD* 2000.

[8] M. Chen et al. Pinpoint: Problem Determination in Large, Dynamic, Internet Services. In *ICDSN (IPDS Track),* 2002.

[9] A. Deligiannakis, Y. Kotidis, N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *SIGMOD* 2004.

[10] M. Dilman and D. Raz. Efficient reactive monitoring. In *Proc. IEEE INFOCOM*, 2001.

[11] P. Dinda et al. The Architecture of the Remos System. In *HPDC* 2001.

[12] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *ICDE* 2002.

[13] M. A. Hammad et al. Nile: A Query Processing Engine for Data Streams (demonstration). In *ICDE* 2004.

[14] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *IPSN* 2003.

[15] A. Jain, E. Y. Chang, Y. Wang. Adaptive Stream Resource Management Using Kalman Filters. In *SIGMOD* 2004.

[16] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An Online Algorithm for Segmenting Time Series. In *ICDM* 2001.

[17] M. Knop, J. Schopf, P. Dinda. Windows Performance Monitoring and Data Reduction Using WatchTower. In *SHAMAN* 2002.

[18] I. Lazaridis and S. Mehrotra. Capturing sensor generated time series with quality guarantees. In *ICDE* 2003.

[19] S. Madden, M. Franklin, J. Hellerstein, W. Hong. TAG: A tiny aggregation service for ad hoc sensor networks. In *OSDI* 2002.

[20] M. J. McPhaden. Tropical atmosphere ocean project, pacific marine environmental laboratory. http://www.pmel.noaa.gov/tao/.

[21] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD* 2003.

[22] T. Palpanas et al. Online amnesic approximation of streaming time series. In *ICDE* 2004.

[23] N. Tatbul et al. Load shedding in a data stream manager. In *VLDB* 2003.

[24] H. Wu, B. Salzberg, D. Zhang. Online Event-driven Subsequence Matching over Financial Data Streams. In *SIGMOD* 2004.

[25] Y. Yao and J. Gehrke. Query processing for sensor networks. In *CIDR* 2003.