Streaming Time Series Summarization Using User-Defined Amnesic Functions

Themis Palpanas, Michail Vlachos, Eamonn Keogh, and Dimitrios Gunopulos, *Member, IEEE*

Abstract—The past decade has seen a wealth of research on time series representations, because the manipulation, storage, and indexing of large volumes of raw time series data is impractical. The vast majority of research has concentrated on representations that are calculated in batch mode and represent each value with approximately equal fidelity. However, the increasing deployment of mobile devices and real-time sensors has brought home the need for representations that can be incrementally updated and can approximate the data with fidelity proportional to its age. The latter property allows us to answer queries about the recent past with greater precision, since in many domains, recent information is more useful than older information. We call such representations *amnesic*. While there has been previous work on amnesic representations, the class of amnesic functions possible was dictated by the representation itself. In this work, we introduce a novel representation of time series that can represent arbitrary user-specified amnesic functions. For example, a meteorologist may decide that data that is twice as old can tolerate twice as much error and thus specify a linear amnesic function. In contrast, an econometrist might opt for an exponential amnesic function. We propose online algorithms for our representation and discuss their properties. Finally, we perform an extensive empirical evaluation on 40 data sets and show that our approach can efficiently maintain a high-quality amnesic approximation.

Index Terms—Time series, streaming data, time-decaying, amnesic, approximation, summarization.

1 Introduction

Time series is one of the most frequently encountered forms of data. Many applications in diverse domains produce voluminous amounts of time series [40], [33]. The sheer number and size of the time series we need to manipulate in many of the real-world applications mentioned above dictates the need for a more compact representation of time series than the raw data itself, and a plethora of representations have been proposed to that effect [22].

The problem of approximating time series becomes more interesting and challenging in the context of streaming time series, where data values are continuously generated, potentially forever. Furthermore, most current time series representations treat every point of the time series equally. This means that when computing the approximation, the time position of a point does not make a difference in the fidelity of its approximation. This may be desirable for some applications such as archiving; however, there exist many real world situations where we would like to take into account the time dimension in the approximation of the time series. The intuition behind this requirement may be stated as follows: While we are willing to accept some margin of error in the approximation, we would like the most recent data to have a low error, and we would be more forgiving of error in older data. We call this kind of time series approximation *amnesic*, since the fidelity of approximation decreases with time, and it therefore requires less memory for the events further in the past.

The potential utility of such a representation has been documented in many domains. Consider the following motivating examples:

- The Environmental Observation and Forecasting System [33] is a large-scale distributed system designed to monitor, model, and forecast wide-area physical processes such as river systems. They note that in their current model, the loss of a repeater station results in the loss of real-time information. Allowing the stations to record some data to a buffer can mitigate this problem. However, since the station does not know how long it will be offline and has a finite buffer, amnesic approximation is the only logical way to record the data.
- NASA is developing robots to be used in an urban setting [18]. Typical applications include search and rescue and inspection of hazardous environments. In many situations, information about the path traversed must be known if the robot is to back up to a more promising avenue of exploration after reaching a dead end. Power and size constraints prohibit the robot from storing all the data with perfect fidelity, so the utility of an amnesic approximation has been noted for this domain [18].

Although this work suggests that the usefulness of data can diminish with age, we note that the rate at which its utility decays depends on the application. The function that determines the amount of error we can tolerate at each point in the time series is called an *amnesic* function. Ideally, we would like to allow arbitrary amnesic functions, so that we can match the requirements of a wide variety of applications. For example, a meteorologist may decide that

- *T. Palpanas is with DIT, University of Trento, Via Sommarive 14, Povo, TN Italy 38050. E-mail: themis@dit.unitn.it.*
- *M. Vlachos is with the IBM T.J. Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532. E-mail: vlachos@us.ibm.com.*
- *E. Keogh and D. Gunopulos are with the Department of Computer Science and Engineering, University of California-Riverside, Riverside, CA 92521. E-mail: {eamonn, dg}@cs.ucr.edu.*

Manuscript received 18 Oct. 2006; revised 9 July 2007; accepted 29 Nov. 2007; published online 7 Dec. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0479-1006.
Digital Object Identifier no. 10.1109/TKDE.2007.190737

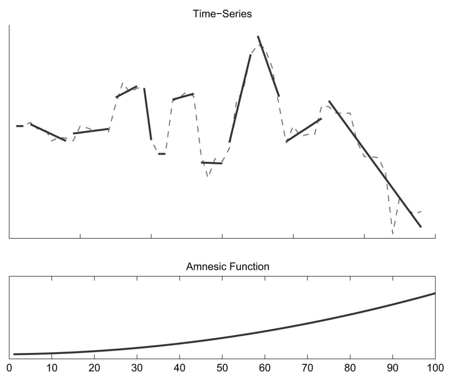


Fig. 1. Depiction of an amnesic approximation, using the piecewise linear approximation technique.

data that is twice as old can tolerate twice as much error and thus specify a linear amnesic function. In contrast, an econometrist using classic models might well specify an exponential amnesic function. Fig. 1 depicts an amnesic approximation of a static time series and the amnesic function that was used. Note that as we get to older points (to the right), the approximation gets coarser.

In this paper, we describe a framework for online amnesic approximation of streaming time series. We characterize the different classes of amnesic functions and present corresponding algorithms for performing amnesic approximation. We study two distinct cases of the problem. First, the case when we are interested in approximating the entire time series seen so far. We refer to this case as the *unrestricted window*. Second, the *sliding-window* case, where at any point in time, we are only interested in a fixed number of the last values of the time series. We present efficient algorithms that solve the problem in both the above cases, given a constraint on the amount of memory that can be used for the approximation. Furthermore, we also discuss a variation of the problem that allows the user to specify the maximum allowable error for the approximation of the time series. This formulation is useful when the application requires quality guarantees for the approximation of the time series. The algorithms we propose for this variation operate for both the unrestricted- and the sliding-window cases but do not have a set bound for the amount of memory they will need for the approximation.

While some recent work [10], [6] has proposed tools and techniques for computing special cases of amnesic approximations of time series, as we discuss in Section 7, these solutions are specific and rather restrictive in the variety of applications they can accommodate. In particular, the representation schemes used by these techniques dictate the form of the amnesic functions and restrict those to a very limited set. In contrast, our framework is general and able to operate with a wide class of amnesic functions, which are defined by the *user*.

Our contributions can be summarized as follows:

- We introduce the notion of general amnesic functions. We present a taxonomy of these functions, discuss their properties, and describe how they affect the solution of the problem of online amnesic approximation.
- We formulate the above problem as optimization problems, where we wish to either minimize the reconstruction error given the available amount of memory for the approximation or minimize the amount of memory required for the approximation given the maximum allowable error for the reconstruction. We study important variations of the above problems, namely, the unrestricted- and the sliding-window cases.
- We propose efficient algorithms for solving the above optimization problems. The time complexity of the algorithms we propose is independent of the size of the time series. The time to process each new point is essentially constant (logarithmic on the number of segments used in the approximation). These are the first algorithms proposed for solving the general case of the problem.
- We present an extensive experimental evaluation of our techniques, using more than 40 synthetic and real data sets. The experiments show the applicability of our approach and the quality of solutions of our algorithms.

The rest of the paper is organized as follows: In Section 2, we give the necessary background. In Section 3, we introduce some new terminology and formally define the problems we study. The algorithms we propose are presented in Sections 4 and 5. Section 6 discusses the experimental evaluation. Section 7 reviews related work, and Section 8 concludes the paper.

2 Time Series Approximation

A time series $T[i]$ is a series of data points, each one arriving at a distinct time instance t_i. $T[i..j]$ defines a range of data points. When the total number of data points in the time series N is known in advance, we call the time series *static*, and we say that is has length N. When data points are arriving continuously, in a streaming fashion, the value of N represents the number of data points seen in the time series so far, and we call the time series *streaming*. The focus of our work is on streaming time series.

Several techniques have been proposed in the literature for the approximation of time series, including *Discrete Fourier Transform (DFT)* [30], [13], *Discrete Cosine Transform (DCT)*, *Piecewise Aggregate Approximation (PAA)* [38], *Discrete Wavelet Transform (DWT)* [28], [9], *Adaptive Piecewise Constant Approximation (APCA)* [8], [25], *Piecewise Linear Approximation (PLA)* [23], *Piecewise Quadratic Approximation (PQA)* [17], and others. Before we consider which of these representations is best suited for the task at hand, it is natural to ask which is best, simply in terms of reconstruction accuracy. In order to answer this question, we experimentally compare the above approaches using many real-world data sets. We conducted such an experiment on 40 diverse time series from the UCR Time Series Data Mining Archive [1].

For our experiment, we randomly extracted a subsequence of length 512 from each time series and approximated it with each of the representations under consideration, using a 16-to-1 compression ratio. This was a fair comparison, using the same amount of memory for each representation. That is, we reduced the 512 raw data points to 32 wavelet coefficients, 16 complex *DFT* coefficients, 32 *DCT* coefficients, 32 *PAA* segments, 12 *PLA* segments, 8 *PQA* segments, and 16 *APCA* segments. Note that we carefully used all possible optimizations for all representations. For example,

TABLE 1
Comparison among Various Techniques for Time Series Approximation

DFT	DCT	PAA	DWT (Haar)	DWT (Daub12)	APCA	PLA	PQA
0.951	0.923	0.948	0.948	0.902	0.893	0.940	0.927

we used the complex conjugate property of *DFT* [30], and because the sequences were normalized to have zero mean, we did not use the first coefficient for the wavelet and *DFT* approaches (they must be zero). However, for the piecewise polynomial approaches, the optimal representation requires quadratic time to produce, and we used a well-known near-linear-time algorithm instead [20], [23]. We measured the quality of the approximation using the root-mean-square error. We repeated this procedure 100 times, averaged the results, and normalized the performance of each representation by dividing by the best performing approach. Finally, we averaged all 40 scores, as shown in Table 1.

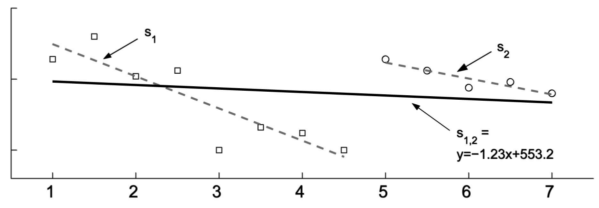
The results may appear surprising, because there is little difference between all the approaches. In fact, similar results have been documented elsewhere as well [22], [8], [37]. The overall conclusion from this experiment is the following. If we want to choose a representation for the task of approximating time series, then we should not choose the representation based on approximation fidelity but rather on other features.

When considering the alternative representations in the context of amnesic approximation, it is not obvious how some of them can accommodate the requirements of this new environment. The *DWT* representation is intrinsically coupled with approximating sequences whose length is a power of two, which severely restricts the choices of amnesic functions. Using wavelets with sequences that have other lengths requires ad hoc measures that reduce the fidelity of the approximation and increase the complexity of the implementation. While *DFT* has been successfully adapted to incremental computation [40], it is not clear that it can be adapted to perform amnesic approximation, since each *DFT* coefficient corresponds to a global contribution to the entire time series. The same is true for *DCT* as well.

In contrast to the above, the piecewise polynomial methods offer several desirable properties for the task at hand. Much is already known about their incremental calculation, and because each segment is independent of each other, we can reduce the fidelity of "older" segments simply by merging them with their neighbors, without affecting "newer" segments. The only question remaining is which piecewise polynomial technique to use. We decide on *PLA* for the following reasons. PLAs are already widely used and accepted in the medical and financial domains [19], [24], [34]. There are many useful distance measures defined on *PLA*, including weighed measures [23], time warping [34], Markov-model-based measures [14], and lower bounding approximations to the euclidean distance. Moreover, many applications like anomaly detection algorithms [21] and rule discovery algorithms [27] use the *PLA* representation.

2.1 Properties of *PLA* Approximation

In *PLA*, we approximate the data points in a time series using a number of linear segments whose ends need not be contiguous [23]. The *PLA* approximation scheme has some desirable properties that allow incremental computation of the solution. These properties are necessary in order for the algorithm to be able to operate efficiently on large data sets. In the following paragraphs, we present these properties in the form of theorems, and we discuss their applications in Section 4.



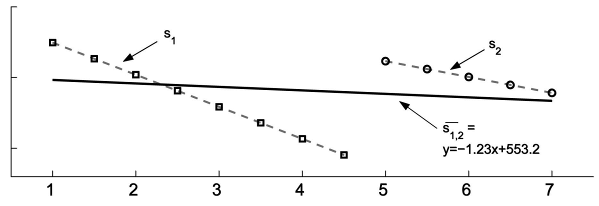


Fig. 2. Combining two regression lines.

Assume that we have N data points of a time series $T[i]$, $1 \leq i \leq N$, and we use them to fit two line segments (using least squares). Let the first line s_1 approximate points 1 to n, $n < N$, and the second line s_2 approximate points $n+1$ to N (there is no restriction on n). In addition, suppose we use a single line segment to approximate all the points 1 to N and call it $s_{1,2}$. The above three lines are depicted in Fig. 2a. Related to these three lines are the errors $E(s_1)$, $E(s_2)$, and $E(s_{1,2})$. The error of a segment s is computed according to the formula $E(s) = \sum_{j \in s}(T[j] - s[j])^2$, where j ranges over all the points in segment s, $T[j]$ is the value of point j in the time series, and $s[j]$ is the estimate for point j given by segment s.

Now, imagine that we keep s_1 and s_2 and throw away the original N points and that we want to use a single line segment to approximate all the original points. The construction of this new line $\overline{s_{1,2}}$ can be based only on the information in s_1 and s_2, and we prove that $\overline{s_{1,2}}$ is the same as $s_{1,2}$. Since we no longer have the original points, we assume that all N points lie *on* line segments s_1 and s_2, and we build $\overline{s_{1,2}}$ based on this assumption. This situation is depicted in Fig. 2b. The residual error of this new line is $\hat{E}(\overline{s_{1,2}})$. Unlike the previous cases, this is the error between the points on line $\overline{s_{1,2}}$ and the points on lines s_1 and s_2. (Remember that line $\overline{s_{1,2}}$ is not calculated based on the original points of the time series.) It turns out that we can also calculate $E(s_{1,2})$ without the need to refer to the original N points.

We can now prove the following theorems regarding the process of merging two line segments into one.

Theorem 1 (Computing the New Line Segment). *The line segment $\overline{s_{1,2}}$, built from the two line segments s_1 and s_2, is the same as the line segment $s_{1,2}$, built from the original points of the time series.*[1] *That is, $s_{1,2} = \overline{s_{1,2}}$.*

Theorem 2 (Computing the New Error). *The error of the line segment approximating all the original data points can*

1. A similar result has also appeared elsewhere [10].

be computed as the sum of the errors of the two individual line segments and the error between those two line segments and the line calculated based on those two. That is, $E(s_{1,2}) = E(s_1) + E(s_2) + \hat{E}(\overline{s_{1,2}})$.

Another interesting property of *PLA* is that for the computation of the error $\hat{E}(\overline{s_{1,2}})$, we do not need to process individually all the points corresponding to line segment $\overline{s_{1,2}}$. We can instead avoid the linear complexity of this procedure and compute the value of $\hat{E}(\overline{s_{1,2}})$ in constant time, according to the following lemma.

Lemma 1 (Computing the Error between Two Segments). *The error* $\hat{E}(\overline{s_{1,2}})$ *of a line segment* $\overline{s_{1,2}}$*, which was constructed from two line segments,* s_1 *and* s_2*, can be computed with a closed-form formula in time* $O(1)$*, regardless of the length of the line segments.*

Proof. We want to prove that the error between two line segments can be computed using a closed-form formula. Let l_1 and l_2 be the two line segments corresponding to the same set of $M+1$ points, $0, \dots, M$. Let $L = |l_1[0] - l_2[0]|$, $R = |l_1[M] - l_2[M]|$, $c = min(L, R)$, and $\Delta = (max(L, R) - c)/M$. In order to compute the error E, we need to sum the squares of the pairwise distances for the $M+1$ points of the line segments. The main observation is that each one of those pairwise distances differs from its neighbors by Δ. The shortest distance is c, the next one is $c + \Delta$, etc., and the last one is $c + M\Delta$. Then, we can compute E as follows:

$$E = \sum_{i=1}^{M+1} (c + (i-1)\Delta)^2$$
$$= \dots = (M+1)\left(c^2 + cM\Delta + \frac{\Delta^2 M(2M+1)}{6}\right).$$

The above analysis assumes that either l_1 and l_2 do not intersect or they intersect at one of their ends (point 0 or point M). If the two line segments intersect at any other point, then we consider the parts of the segments on either side of the intersection point separately and apply the above formula twice. □

The properties of *PLA*, presented in Theorems 1 and 2 and Lemma 1, form the basis for the design of the online algorithms we propose. These properties enable our algorithms to merge two line segments and calculate exactly the resulting line segment along with its residual error in constant time.

3 Problem Formulation

In the following paragraphs, we establish some additional terminology necessary for the rest of the paper. Then, we formally define the problems that we address with this work.

3.1 Amnesic Functions

As we mentioned earlier, we need a way to specify for each point in time the amount of error allowed for the approximation of the time series. In order to achieve this goal, we use the *amnesic function* $A(x)$, which returns the acceptable approximation error for point $x = t_N - t_i$, where t_N is the current time, and t_i is the time that point $T[i]$ arrived. The time t_N refers to the time when the last data point arrived and corresponds to position $x = 0$ of the amnesic function. Note that the function $A(x)$ is only defined for $x \geq 0$, since $t_i \leq t_N$.

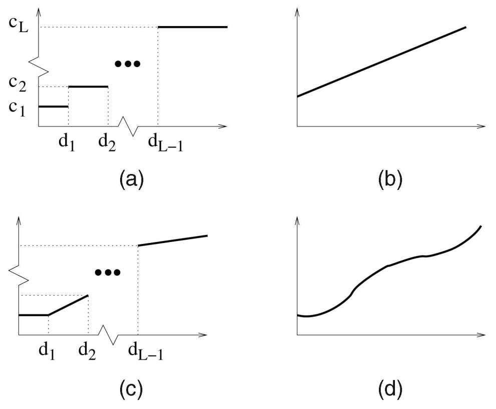


Fig. 3. The different classes of amnesic functions. (a) Piecewise constant. (b) Linear. (c) Piecewise linear. (d) Continuous.

A key property that an amnesic function has to satisfy is the *monotonicity* property.

Definition 1 (Monotonic Amnesic Functions). *A function* $A(x)$ *is called monotonic if* $A(x) \leq A(x+1)$ *for every value of* x *in its domain.*

The approximation of a time series is a lossy compression technique, which by definition is irreversible. Thus, the monotonicity property poses a natural restriction in our setting. It ensures that if at time t, we can tolerate some error in the approximation of point $T[i]$, $E^t(T[i])$, then we will not request an approximation of the same point $T[i]$ with error $E^{t'}(T[i]) < E^t(T[i])$, at any time $t' > t$.

We now define a taxonomy of amnesic functions (refer to Fig. 3). The constant amnesic function represents a trivial case, and we do not discuss it here. As we will discuss in Section 3.2, each class in the taxonomy has its own special characteristics, which have to be taken into account when designing an efficient algorithm for the amnesic approximation of time series.

Piecewise Constant. The *piecewise constant function* has the following general form:

$$A(x) = \begin{cases} c_1, & 0 \leq x < d_1, \\ \dots \\ c_L, & d_{L-1} \leq x, \end{cases}$$

where $c_1, \dots, c_L$ are constants such that $0 < c_1 < \dots < c_L$. We refer to each step of the function as a *section* to distinguish it from the segments used in the approximation.

Linear. A *linear function* has the general form $A(x) = \alpha x + \beta$, α, $\beta > 0$.

Continuous Piecewise Linear. The general form of a *piecewise linear function* with L sections is given as follows:

$$A(x) = \begin{cases} \alpha_1 x + \beta_1, & 0 \leq x < d_1, \\ \dots \\ \alpha_L x + \beta_L, & d_{L-1} \leq x, \end{cases}$$

where $\alpha_j \geq 0$, $1 \leq j \leq L$, $\beta_1 > 0$, and

$$\beta_2 = \alpha_1 d_1 + \beta_1 - \alpha_2 d_1, \dots, \beta_L = \alpha_{L-1} d_{L-1} + \beta_{L-1} - \alpha_L d_{L-1}.$$

Continuous. The amnesic functions of this class can take any form not subsumed by the previous classes. The only restriction is that the function is monotonic (according to Definition 1). We do not require that these functions have a closed-form formula.

We also define two forms of amnesic functions, namely, the *relative* $RA(x)$ and the *absolute* $AA(x)$ amnesic functions.

Relative. A relative amnesic function RA determines the relative approximation error we can tolerate for every point in the time series. When we use a relative amnesic function, we essentially weigh the error of a data point by the inverse of the amnesic function corresponding to that point so that the weighted error for point x is $E(x)/RA(x)$. For example, the relative amnesic function $RA(x) = x + 1$ specifies that when we approximate a point that is twice as old, we will accept twice as much error. When we use relative amnesic functions, we fix the number of linear segments that we are allowed to use for the approximation of the data.

Absolute. An absolute amnesic function specifies, for every point in the time series, the *maximum* allowable error for the approximation. The error $E(x)$, at point x, should satisfy the inequality $E(x) \leq AA(x)$. When we use absolute amnesic functions, we allow the approximation to use as many linear segments as necessary in order to meet the error bounds.

When we have to apply an amnesic function to a segment s, we pick a single point from the segment, on which we apply the amnesic function. Nevertheless, this computation refers to the entire segment. The reason we do this is that we do not store the error of each individual point represented by each segment, and we only have available the error of the entire segment. For the rest of this paper, we assume that segment s is represented by its most recent point, $T[i_s]$. Then, when we want to apply an amnesic function to s, we simply consider the point of the amnesic function corresponding to point $T[i_s]$. We can also apply more elaborate schemes. For example, we could consider taking the average value of the amnesic function corresponding to the first, middle, and last points of s. In any case, the algorithms we propose do not need to change.

3.2 Problems for Amnesic Approximation

Under the assumptions discussed above, we want to maintain a *PLA* model Q with K segments for a streaming time series with an unrestricted window. More formally, we define the following two problems.

Problem 1 (Unrestricted Window with Relative Amnesic (URA)]) *Given the number of segments K and a relative amnesic function $RA(x)$, find an approximation Q using K segments that minimizes the approximation error of the time series $\sum_{j=1}^{K}(E(s_j)/RA(t_N - t_{s_j}))$.*

Problem 2 (Unrestricted Window with Absolute Amnesic (UAA)). *Given an absolute amnesic function $AA(x)$, construct a model Q with the minimum number of segments K, subject to the constraints $E(s_j) \leq AA(t_N - t_{s_j})$, $1 \leq j \leq K$.*

We are looking for online algorithms that, when a new point arrives, update the approximation model in sublinear time on the number of segments. Note that in the *URA* and *UAA* problems, the optimization objective is different. In the *URA* problem, we seek to minimize the approximation error given the memory space used by *PLA*, while in the *UAA* problem, we want to minimize the space used in the approximation given the maximum error allowed.

Following the definition of the problems for the unrestricted window, we now define the corresponding problems for the case where we consider the sliding-window model.

Problem 3 (Sliding window with Relative Amnesic (SRA)). *Given a sliding window of length W, the number of segments K, and a relative amnesic function $RA(x)$, find an approximation Q using K segments that minimizes the approximation error of the time series within the sliding window $\sum_{j=1}^{K}(E(s_j)/RA(t_N - t_{s_j}))$, $t_{N-W+1} \leq t_{s_j} \leq t_N$.*

Problem 4 (Sliding window with Absolute Amnesic (SAA)). *Given a sliding window of length W and an absolute amnesic function $AA(x)$, construct a model Q with the minimum number of segments K, subject to the constraints $E(s_j) \leq AA(t_N - t_{s_j})$, $t_{N-W+1} \leq t_{s_j} \leq t_N$, $1 \leq j \leq K$.*

4 Algorithms for Relative Amnesic Functions

We now describe algorithms for the *URA* and *SRA* problems. In the experimental evaluation, we show that our algorithms perform very close to optimal. At the end of the section, we briefly discuss solutions for *UAA* and *SAA*.

4.1 Unrestricted Window with Relative Amnesic

4.1.1 Optimal Solution

The optimal solution for the *URA* problem can be obtained using dynamic programming [5]. The objective of the algorithm is to minimize $ApErr(b, k)$, which is the error resulting from the approximation of data points $b, \ldots, N$ with $k < K$ segments. The recursion for the dynamic programming solution is described by the following formula:

$$ApErr(b,k) = \min_{b \leq j \leq N}(E(T[b \ldots j]) + ApErr(j+1, k-1)). \quad (1)$$

The algorithm starts by computing the approximation error $E(T[b \ldots j])$, for $1 \leq b \leq N$ and $b \leq j \leq N$. Then, at each iteration, it computes the optimal solution by minimizing the total approximation error. The minimum error for approximating data points $b, \ldots, N$ with k segments is given by the sum of the approximation error of points $b, \ldots, j$ with one segment and the error of the optimal approximation of points $j+1, \ldots, N$ with $k-1$ segments. Finally, the algorithm picks the assignment of segments that leads to the least overall approximation error.

Note that in order to get the optimal solution in a streaming environment, we have to run the dynamic programming algorithm every time that a new data point arrives. The reason is that we cannot reuse the computations made during the previous step, because the amnesic function causes the approximation error of each point and their interrelationships to change at every time step. The time complexity for the dynamic programming algorithm is $O(N^2K)$, which renders this approach inapplicable for the online version of the problem. Nevertheless, in the experimental section, we show that our algorithms always find a solution that is very close to optimal.

```
1 let $H$ be a min-priority queue on the approximation errors
      resulting from merging each pair of consecutive segments;
2 let $EQ = \emptyset$ be a time-event queue;
3 procedure GrAp-R()
4    when a new point, $T[i]$, of the time series arrives at time $t_N$
5        pick the minimum element from $H$, and merge the
             corresponding segments, $s_m$ and $s_{m+1}$, into a new
             segment $s_{m,m+1}$;
6        update $H$ with the errors of merging $s_{m,m+1}$ with its two
             neighboring segments;
7        assign a new segment, $s_{T[i]}$, to the newly arrived
             point, $T[i]$;
8        update $H$ with the error of merging $s_{T[i]}$ with its
             neighboring segment;
9        ManageEvents($H$, $EQ$, $t_N$, $s_m$, $s_{m+1}$, $s_{m,m+1}$);
10   return;
```

Fig. 4. The skeleton of the *GrAp-R* algorithm.

4.1.2 The GrAp-R Algorithm

In this section, we present the skeleton of our algorithm, *GrAp-R*, for solving the *URA* problem.

At each time step, the algorithm merges the consecutive pair of segments whose merge will result in the least approximation error among all possible merges. The pair of segments that should be merged, s_m and s_{m+1}, is given by the heap structure H. We merge those in one segment $s_{m,m+1}$ according to Theorems 1 and 2. Then, we compute the approximation error that would result by merging the new segment with each one of its two neighbors s_{m-1} and s_{m+2} according to Lemma 1. We use these values for the errors to update the heap H, in order to reflect the new set of possible merges. This merge results in a spare segment, which we assign to the newly arrived point of the time series. Once again, we have to compute the approximation error when merging this segment with its neighbor and update the heap H. A high-level description of the algorithm is depicted in Fig. 4.

The *GrAp-R* algorithm also makes use of queue *EQ*. This structure keeps track of the way that the dependencies among the segments used for the approximation change as a result of the amnesic function. The procedure that manages these dependencies is *ManageEvents()*, and we describe it in more detail in the next paragraphs.

In the following sections, we elaborate on the way the framework of the *GrAp-R* algorithm described above changes when we consider the different classes of amnesic functions. We discuss the specific details of each case and present the time and space complexities of the solutions we propose.

4.1.3 Piecewise Constant Amnesic Functions

When the amnesic function belongs to the class of piecewise constant functions, a change to the relative ordering of the pair of segments that should be merged during the next step of the algorithm only happens when a segment crosses a discontinuity between two sections of the amnesic function.

Example 1. Assume that we have the amnesic function $RA(x) = 1,\ 0 \leq x < 10$, and $RA(x) = 4,\ x \geq 10$. Let $s_{1,2}$ and $s_{3,4}$ be two pairs of segments, candidates for merging, that, at the current time, are at positions $x = 7$ and $x = 2$ and have errors $E(s_{1,2}) = 4$ and $E(s_{3,4}) = 2$, respectively (Fig. 5a). Then, their relative errors are $E(s_{1,2})/RA(7) = 4$ and $E(s_{3,4})/RA(2) = 2$, which means that $s_{3,4}$ is the first candidate for merging. However, after three time instances, when $s_{1,2}$ first gets to the point $x = 10$, its error becomes $E(s_{1,2})/RA(10) = 1 < E(s_{3,4})/RA(5) = 2$ (Fig. 5b). Thus, $s_{1,2}$ is now the candidate pair for merging.

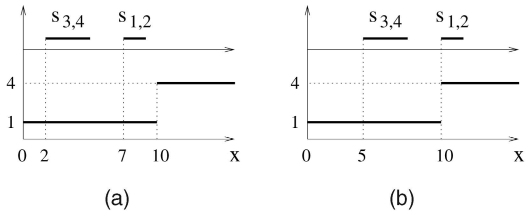


Fig. 5. Event example for piecewise constant.

In order to keep track of these changes, we need to maintain the heap H and, in addition, a time-event queue *EQ*. The heap H determines the next pair of segments that should be merged. The queue *EQ* flags the times at which the segments cross a discontinuity in the amnesic function (remember that during these computations we assume that each segment is represented by its most recent point). When this happens, we update the position of the segment in the heap, and we compute the next time that it will cross a discontinuity. Fig. 6 shows the *ManageEvents()* procedure for the case of piecewise constant amnesic functions. The *GrAp-R* algorithm remains as discussed earlier.

The following theorem states the space and time complexity of the algorithm.

Theorem 3. *The space complexity of* GrAp-R *with a piecewise constant amnesic function is $O(K)$, and the time complexity to process each new point is $O(L \log K)$.*

Proof. The algorithm needs $O(K)$ space to store the K segments used in the approximation. A heap structure is used to determine the pair of segments that will be merged at each step of the algorithm. The heap requires $O(K)$ space to store the $K - 1$ adjacent pairs of segments. Finally, we must keep track of the times when segments cross a discontinuity of the amnesic step function. At each point in time, we only need to maintain in the time-event queue one such event for every segment. Therefore, the queue has a worst space complexity of $O(K)$, and $O(K)$ is the overall space complexity of the algorithm as well.

At each time unit, the algorithm can pick from the heap the pair of segments to merge and identify in the time-event queue the segments that cross a discontinuity in $O(1)$ time. The time to merge two segments is constant, because of Theorems 1 and 2 and Lemma 1. The time to update the heap is $O(\log K)$, and since the size of the time-event queue is $O(K)$, the time to insert or delete an event from the queue is $O(\log K)$ (when the queue is implemented using skiplists [29] or any other equivalent data structure that offers logarithmic search times). Thus, the overall time complexity for each iteration, when there is only one segment crossing a

```
1 proc ManageEvents($H$, queue $EQ$, time $t$, segments
      $s_m$, $s_{m+1}$, $s_{m,m+1}$)
2   remove from $EQ$ any events corresponding to segments $s_m$
        and $s_{m+1}$;
3   if (next event $e$ in $EQ$ is scheduled for time $t < t_e \leq t+1$)
4     remove $e$, related to segments $s_{e,1}$ and $s_{e,2}$, from $EQ$;
5     update in $H$ the position of the pair $s_{e,1}$ and $s_{e,2}$;
6     compute the new time when the pair $s_{e,1}$ and $s_{e,2}$ will
          cross a discontinuity;
7     insert in $EQ$ the new event (if any);
8   insert in $EQ$ any new dependencies identified concerning
        $s_{m,m+1}$;
9   return;
```

Fig. 6. The *ManageEvents()* procedure for piecewise constant amnesic functions.

discontinuity, is $O(\log K)$. In the worst case, for a particular iteration, different segments may be crossing all L discontinuities of the amnesic function, so the worst-case time complexity is $O(L \log K)$. □

Note that the time complexity mentioned in the above theorem refers to the worst case, when segments cross all the discontinuities of the amnesic function at the same time. In practice, we do not expect this situation to arise often. Actually, it never occurred in our experiments with an extensive set of real data sets.

4.1.4 Linear Amnesic Functions

In the case of linear amnesic functions, each event in EQ specifies the time at which the relative ordering of the merging error of two pairs of segments changes. It turns out that if we know the approximation error of each segment and the closed formula of the amnesic function, we can compute the times at which these changes will occur. We refer to those times as the *crosspoints*.

Example 2. Assume that we have the amnesic function $RA(x) = x + 1, x \geq 0$. Let $s_{1,2}$ and $s_{3,4}$ be two pairs of segments, candidates for merging, that were created at the current time, at positions $x = 6$ and $x = 2$, and have errors $E(s_{1,2}) = 24$ and $E(s_{3,4}) = 12$, respectively (Fig. 7a). Then, their relative errors are $E(s_{1,2})/RA(6) = 3.4$ and $E(s_{3,4})/RA(2) = 4$, which means that $s_{1,2}$ is the first candidate for merging. However, after four time instances, when $s_{1,2}$ first gets to the point $x = 10$, its error becomes $E(s_{1,2})/RA(10) = 2.2 > E(s_{3,4})/RA(6) = 1.7$ (Fig. 7b). Thus, $s_{3,4}$ is now the candidate pair for merging.

Consider the general case, where we have a linear relative amnesic function $RA(x) = \alpha x + \beta$, and we want to compute the time when the relative ordering of segments s_1 and s_2 will change. (In fact, each one of s_1 and s_2 represents the merge of a pair of segments.) Let $E(s_1)$ and $E(s_2)$ be the approximation errors for s_1 and s_2, respectively. Finally, assume that t_{s_1} is the time when s_1 was created. This time is defined as the time when the most recent point of s_1 arrived. We define t_{s_2} in a similar way. Then, their crosspoint t_c is given by the following equation:

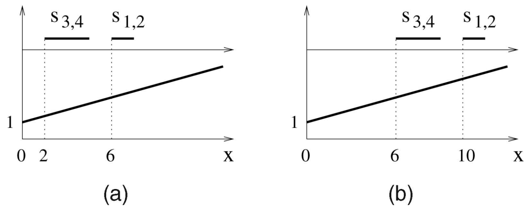


Fig. 7. Event example for piecewise constant.

$$\frac{E(s_1)}{\alpha \cdot (t_c - t_{s_1}) + \beta} = \frac{E(s_2)}{\alpha \cdot (t_c - t_{s_2}) + \beta}, \text{ or}$$
$$t_c = \frac{(\alpha \cdot t_{s_2} - \beta) \cdot E(s_1) - (\alpha \cdot t_{s_1} - \beta) \cdot E(s_2)}{(E(s_1) - E(s_2)) \cdot \alpha}. \quad (2)$$

We only consider the positive solutions of this equation. Note that it may be the case that their relative ordering never changes, that is, there is no positive solution. Furthermore, we do not need to compute the crosspoint of each segment with all the others. It suffices to consider only the segments stored in the neighboring nodes in the heap H and maintain these dependencies up to date as the heap changes. All these computations can be performed in constant time according to (2).

The *ManageEvents()* procedure for the case of linear amnesic functions is depicted in Fig. 8.

The problem of keeping track of the crosspoints is reminiscent of the work in the area of *kinetic data structures* [4]. However, the above work examines only linear motion and does not apply to our problem.

The complexity of the algorithm is given in the following theorem.

Theorem 4. *The space complexity of* GrAp-R *with a linear amnesic function is $O(K)$, and the time complexity to process each new point is $O(\log K)$.*

Proof. The algorithm requires $O(K)$ space to store the K segments and the heap. The time-event queue also requires $O(K)$ space, since it stores one event for every adjacent pair of segments.

At each iteration, the time to find the pair of segments to merge and the segment that has reached a crosspoint is $O(1)$. We need $O(\log K)$ time to update the heap after those changes. We also need to update the queue, which takes $O(\log K)$ time. Therefore, the overall time complexity for each iteration is $O(\log K)$. □

4.1.5 Piecewise Linear Amnesic Functions

Assume that the amnesic function is comprised of L sections. Then, we treat each section separately, as in the case of linear amnesic functions discussed above. We maintain L heaps, one for each section, and a single time-event queue. The time-event queue, in addition to keeping track of all the crosspoints, also maintains the times at which a segment moves from one section to another. The above L heaps carry local information, as to which is the best pair of segments to merge within each section. Then, at each iteration of the algorithm, it is easy to determine the overall best pair of segments to merge, either by performing a linear scan of the top element of the L heaps or by maintaining a heap of those L elements. For all practical

```
1 proc ManageEvents($H$, queue $EQ$, time $t$, segments
      $s_m$, $s_{m+1}$, $s_{m,m+1}$)
2   remove from $EQ$ any events corresponding to segments $s_m$
        and $s_{m+1}$;
3   if (next event $e$ in $EQ$ is scheduled for time $t < t_e \leq t+1$)
4     remove $e$, related to segments $s_{e,1}$ and $s_{e,2}$, from $EQ$;
5     swap in $H$ the positions of $s_{e,1}$ and $s_{e,2}$;
6     compute crosspoints between $s_{e,1}$ and $s_{e,2}$ and all their
          new neighbors (i.e., parent and children nodes) in $H$;
7     insert in $EQ$ events for any new crosspoints identified;
8   insert in $EQ$ any new crosspoints identified concerning
      $s_{m,m+1}$;
9   return;
```

Fig. 8. The *ManageEvents()* procedure for linear amnesic functions.

purposes, L is relatively small, in the order of a few dozens. Therefore, a linear scan is sufficiently fast and avoids the need for maintaining the extra heap structure, which in the worst case has time complexity $O(L \log L)$. For the rest of this work, we only consider the linear-scan approach.

The following theorem gives the space and time complexity of the algorithm.

Theorem 5. *The space complexity of* GrAp-R *with a piecewise linear amnesic function is $O(K)$, and the time complexity to process each new point is $O(L + L \log \frac{K}{L} + \log K)$.*

Proof. We assume that an equal number of segments correspond to each section of the amnesic function.[2] The algorithm requires $O(K)$ space for storing the K segments and the L heaps (since all the heaps combined store $O(K)$ values). The time-event queue stores an event for every adjacent pair of the K segments and also an event for the segments that will move from one section to the next. Therefore, we need $O(K)$ space in total.

In terms of time, the algorithm at each iteration needs $O(\log K)$ time to update the time-event queue, $O(L \log \frac{K}{L})$ time to update the L heaps, and $O(L)$ time to pick the best pair of segments to merge. □

4.1.6 Continuous Amnesic Functions

When the amnesic function is continuous, we identify two cases. First, the amnesic function has a closed-form formula. In this case, we can compute the crosspoints of the segments, and we proceed as with the linear amnesic functions. Second, when the amnesic function does not have a closed-form formula, we replace the continuous function with a PLA using L sections. Then, we proceed as with the piecewise linear amnesic functions. We construct L heaps and search in those for the best pair of segment to merge. Since the resulting amnesic function is an approximation of the original function, instead of examining only the top element from each heap, we consider the top-q elements. We calculate the exact error (i.e., based on the continuous amnesic function) of those elements and pick the best pair of segments among them. This technique proves to work very well, even for a small q.

The following theorem gives the space and time complexity of the algorithm.

Theorem 6. *Assume that we approximate a continuous amnesic function with L piecewise linear sections. Further, assume that we consider the top-q elements of each heap in order to identify the best pair of segments to merge. Then, the space complexity of* GrAp-R *with a continuous amnesic function is $O(K)$, and the time complexity to process each new point is $O(qL + L \log \frac{K}{L} + \log K)$.*

Proof. We assume that an equal number of segments correspond to each section of the amnesic function. The algorithm requires $O(K)$ space for storing the K segments and the L heaps (since all the heaps combined store $O(K)$ values). The space required by the time-event queue is $O(K)$, to store events for all adjacent pairs of segments and for the segments crossing a section. Therefore, we need $O(K)$ space in total.

In terms of time, the algorithm at each iteration needs $O(\log K)$ time to update the time-event queue, $O(L \log \frac{K}{L})$ time to update the L heaps, and $O(qL)$ time to pick the best pair of segments to merge. □

4.2 Sliding Windows with Relative Amnesic

In this section, we discuss algorithms that solve the online amnesic approximation problem for a sliding window of a streaming time series. Assume a sliding window of size W and that we use *PLA* to build an approximation model Q with K segments. We refer to the side of the sliding window from which new points enter the window as the *start* of the sliding window. We call the *end* of the sliding window the side from where points exit and *last* segment the segment of Q that approximates the points of the series at the end of the sliding window.

The skeleton of the algorithms for the sliding-window case is the same as the one presented in Section 4.1 for the amnesic approximation of time series in an unrestricted window. The only difference is that we now have to adjust the approximation such that there is no segment that refers to data points beyond the end of the sliding window. In order to achieve this goal, we simply discard the last segment as soon as it gets entirely out of the sliding window, and we reuse it at the start of the window. Observe though that the amnesic function is more tolerable to the approximation error toward the end of the sliding window. Then, a question that arises naturally is whether it is possible for the last segment to continue growing by merging with the second to last segment and consequently never fall out of the boundaries of the sliding window. The following lemma addresses this question.

Lemma 2. *The last segment of model Q will not grow to represent the entire set of points beyond the end of the sliding window.*

Proof. We will prove this statement by contradiction. Assume that the last segment s_K never falls out of the sliding window. This necessarily means that the error of s_K, $E(s_K)$, is not always the largest among the errors of all the segments in the sliding window (so that it gets

2. This assumption is realistic because of the following observation. The sections of the amnesic function that refer to the newer values of the time series will tend to be of finer granularity and encompass a smaller portion of the time series than the sections referring to the older values. Nevertheless, they will require a higher ratio of segments per data point, since the requirements for accuracy in the newer data points is higher than that for the older ones.

picked to be merged with the second to the last segment s_{K-1}). If s_K never completely falls out of the window, then it represents all the points of the time series beyond the end of the sliding window. As the window moves forward, more points are being added to it (or otherwise s_K would fall outside the sliding window). As more points are added to s_K, $E(s_K)$ keeps increasing. If we assume an infinite data stream, then $\lim_{t\to+\infty} E(s_K) = +\infty$. The above equation holds even when we take into account the amnesic function. Thus, for a sufficiently large time point, $t_L : E(s_K) > E(s_i)$, $1 \le i < K$, $\forall t > t_L$, since all the other segments represent a bounded number of data points and therefore have a bounded error smaller than $E(s_K)$. This violates our assumption above that $E(s_K)$ is not always the largest among all the segments in the window. □

The above lemma guarantees that a sliding-window amnesic approximation will never degenerate to an unrestricted-window approximation of the time series but does not give us a bound on the size of the last segment. In Section 6, we experimentally show that the size of the last segment is always relatively small.

```
1 let $EQ = \emptyset$ be a time-event queue;
2 procedure **GrAp-A**()
3    when a new point, $T[i]$, of the time series arrives at time $t_N$
4        assign a new segment $s_{T[i]}$ to the new point;
5        let $s_m$ be a hypothetical segment resulting from the merge of
             $s_{T[i]}$ and its neighboring segment;
6        if ($E(s_{T[i]}) < AA(t_N)$)
7            accept segment $s_m$ in the approximation model;
8        else
9            let $s_m = s_{T[i]}$;
10       let $s_{m+1}$ be the one segment adjacent to $s_m$;
11       **ManageEvents**($EQ$, $t_N$, $s_m$, $s_{m+1}$);
12   return;

13 proc **ManageEvents**(queue $EQ$, time $t$, segments $s_m$, $s_{m+1}$)
14    insert in $EQ$ an event for the time when $s_m$ and $s_{m+1}$ will be
          able to merge;
15    if (next event $e$ in $EQ$ is scheduled for time $t$)
16        remove $e$, related to segments $s_{e,1}$ and $s_{e,2}$, from $EQ$;
17        merge segments $s_{e,1}$ and $s_{e,2}$ into segment $s_e$;
18        let $s_{e-1}$ and $s_{e+1}$ be the segments adjacent to $s_e$;
19        insert in $EQ$ events for the times when $s_{e-1}$ and $s_e$, and
              $s_e$ and $s_{e+1}$ will be able to merge;
20    return;
```

Fig. 9. The skeleton of the *GrAp-A* algorithm.

5 Algorithms for Absolute Amnesic Functions

In this section, we present algorithms for the *UAA* and *SAA* problems, that is, the online amnesic approximation of streaming time series with absolute amnesic functions. First, we discuss the algorithms for the unrestricted window and then extend the discussion for the sliding-window case.

5.1 Unrestricted Window

5.1.1 The GrAp-A Algorithm

When we use absolute amnesic functions, we do not know in advance the number of segments that will be needed for the approximation. Furthermore, we can calculate the time when a neighboring pair of segments will be eligible to merge. Hence, in this case, we do not have to keep track of the segments whose merge will result in the least additional error, and subsequently, there is no need to maintain a heap structure on the adjacent pairs of segments, as we did for the case of the relative amnesic functions.

Fig. 9 shows a high-level description of the *GrAp-A* algorithm, which we propose for the solution of the *UAA* problem. During each time step, the algorithm assigns the new data point of the time series to a new segment s_i by itself (line 4). Then, it tries to merge s_i with its adjacent segment (line 6). Note that at this time, there is only one segment adjacent to s_i, since s_i represents the last data point seen so far, at the end of the time series. If the segment that results from the merge has an error less than what is specified by the absolute amnesic function, then the merge is realized (line 7). In either case, the next step involves the update of the time-event queue EQ (line 11).

The procedure *ManageEvents()* keeps track of the merges we know in advance that should happen and updates EQ correspondingly. First, in line 14, it schedules in EQ an event specifying when the segment that was formed with the arrival of the new data point will be able to merge with its adjacent segment. Then, in lines 15-19, it processes any merges that are scheduled to happen at the current time and updates the queue EQ. In the following paragraphs, we elaborate on the above issue for the different kinds of amnesic functions.

5.1.2 Piecewise Constant Amnesic Functions

We observe that given two segments, we can precompute at which point in time (if any) we will be able to merge them. Moreover, we can simplify the problem by considering, as viable time points for merging two segments, only the exact times at which both segments have crossed a discontinuity of the amnesic function. Remember that the discontinuity points $d_1, \dots, d_{L-1}$ are specified in the definition of the amnesic function and therefore are known (see Section 3.1). Because of the form of the piecewise constant functions, we are certain that if two segments cannot merge once they change sections in the amnesic function, they will definitely not be able to merge before they change sections again.

For each adjacent pair of segments in our approximation, s_i and s_{i+1}, it suffices to take the following three steps:

1. Assume that segments s_i and s_{i+1} are merged into $s_{i,i+1}$ and calculate the error of $s_{i,i+1}$, $E(s_{i,i+1})$.
2. Compute the earliest point in time when $E(s_{i,i+1})$ becomes less than the specified amnesic absolute error and call this time t_{merge}. This computation is fast, because we only need to consider the discontinuity points that $s_{i,i+1}$ has not crossed yet. That is, in order to determine t_{merge}, we have to make the computation $L-1$ times in the worst case. It turns out that if we use the monotonicity property of the absolute amnesic functions, then we can reduce the computation effort. The monotonicity property says that as we move toward the past, the approximation error allowable by the amnesic function is increasing monotonically. This means that instead of a linear scan on the discontinuity points, we can employ a binary search, which will result in faster computation

of t_{merge}. However, for the purposes of this paper, we only use the linear-scan approach.

3. Schedule an event in queue EQ, regarding segments s_i and s_{i+1}, for time t_{merge}.

5.1.3 Linear Amnesic Functions

In the case of linear absolute amnesic functions, we can compute the time during which two neighboring segments will be eligible to merge based on a closed-form formula. This is the time when the resulting segment will have an approximation error equal to or below the error specified by the absolute amnesic function.

Consider the case where we are trying to compute the time t_{merge} when two existing segments s_1 and s_2 can be merged into segment s. Assume that the linear amnesic function is described by the equation $AA(x) = \alpha x + \beta$ and $E(s)$ is the approximation error for s and let t_s be the time when s was created. This is defined as the time when the most recent point in s arrived. Then, the time t_{merge} is given by the following equation:

$$E(s) = \alpha \cdot (t_{merge} - t_s) + \beta, \text{or } t_{merge} = \frac{E(s_1) - \beta}{\alpha}. \quad (3)$$

We only consider the positive solutions of this equation. (In this case, negative solutions mean that the merge should have occurred in the past.) Also, note that since the amnesic function is monotonic, t_{merge} indicates the earliest point in time when we can realize the merge. The merge may also happen at any time $t > t_{merge}$.

When we compute the time t_{merge} for a pair of segments, we insert this time in the time-event queue EQ. After a merge has occurred, we have to compute the times that the new segment will be able to merge with its two neighbors and update the time-event queue.

5.1.4 Piecewise Linear Amnesic Functions

Assume that there are L sections in the piecewise amnesic function. Then, we have to treat each section separately and, for each section, proceed as in the case of linear amnesic functions. We still need to maintain a single time-event queue EQ to handle the events from all L sections. The processing in this case is described as follows:

For section l_i, which is specified by the discontinuity points d_{i-1} and d_i, we compute for each pair of segments the time at which they could merge. If this time falls inside the interval $[d_{i-1}, d_i]$, then we insert in EQ a merge event with this time. Otherwise, we simply insert in EQ an event specifying that the corresponding pair of segments should be reexamined when it crosses over to the next section l_{i+1}. We do the same for the pairs of segments that span across two sections of the amnesic function. In this case, the relevant computations are performed as if they both belong to the section where their most recent point belongs to.[3]

5.1.5 Continuous Amnesic Functions

When the amnesic function is continuous, we can only calculate the times of the possible segment merges if the function has a closed-form formula. In the general case, where there is no closed-form formula available, we approximate the continuous amnesic function with a piecewise linear function with L sections. Then, we proceed by treating the amnesic function as a piecewise linear one, as described in Section 5.1.4.

Note, however, that we still have to compensate for the error due to the PLA of the continuous function. Consider a pair of segments, corresponding to section l_i of the piecewise linear function, that are candidates to merge into segment s. Assume that we have computed the merging time for s, t_{merge}, according to the PLA $\widetilde{AA}(x)$ of the amnesic function $AA(x)$. This means that $E(s) \leq \widetilde{AA}(t_{merge} - t_s)$, where t_s is the time when the most recent point of s arrived. We then check whether $E(s) \leq AA(t_{merge} - t_s)$, which is what we really want to hold. If the above inequality is true, then we update EQ with an event concerning s at time t_{merge} as usual. Otherwise, we schedule in EQ an event to reexamine the merging time for s when it crosses over to the next section of the piecewise linear function, l_{i+1}. We know that the merge is more likely to occur then, because of the monotonic property of the amnesic function.

We considered other approaches for dealing with this problem as well. When the inequality $E(s) \leq AA(t_{merge} - t_s)$ does not hold, we can test other possible time points for the merge, before s moves to the next section. On the other extreme, we can choose the PLA of the continuous amnesic function in such a way that $\widetilde{AA}(x) \leq AA(x)$, $\forall x$. In this case, we know that for any s, $E(s) \leq \widetilde{AA}(t_{merge} - t_s) \leq AA(t_{merge} - t_s)$. The approach that we propose is the middle ground between those two.

5.2 Sliding Window

We now turn our attention to algorithms that work on a sliding window of the time series stream. These algorithms are based on the corresponding ones for the unrestricted window with only minor modifications. Consider the example illustrated in Fig. 10. The two figures depict the amnesic approximation of the values of the time series that fall in the sliding window for two time instances (the absolute amnesic function remains the same across time). In the first time instance (Fig. 10a), two line segments are enough in order to produce an approximation that has less error than what is specified by the amnesic function. For the second time instance (Fig. 10b), we need five segments in order to meet the same approximation error requirements (albeit for a different set of values). Note that as time advances, the number of line segments used for the approximation may increase or decrease.

In the sliding-window context, we only need to maintain a representation of the values of the time series in the window. Consequently, we do not insert in the time-event queue EQ any events that refer to a time point past the end of the window. These are events about merges that cannot occur, since the corresponding segments will be dropped from the representation Q as soon as they fall out of the sliding window (it is an easy exercise to prove a lemma similar to Lemma 2).

3. This choice was made in accordance to the way we apply an amnesic function to a segment, described in Section 3.1.

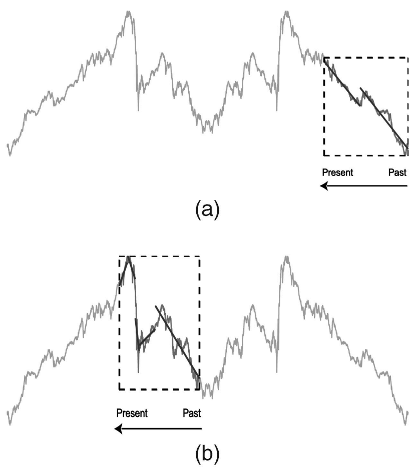


Fig. 10. Example of sliding-window amnesic approximation with absolute amnesic functions (two time instances shown for a random walk data set). (a) Time t_1. (b) Time $t_2 > t_1$.

5.3 On the Complexity of the Algorithms

The following theorem states the space and time complexity for all the variations of the *GrAp-A* algorithm discussed above.

Theorem 7. *Assume that we employ a piecewise constant amnesic function. Then, the space complexity is $O(K)$, and the time complexity to process each new point is $O(\log K)$.*

Proof. The algorithm requires $O(K)$ space to store the K segments used in the approximation. The queue EQ has space complexity $O(K)$, since at each point in time, we only need to maintain in the time-event queue only one event for every pair of segments. Therefore, the overall space required is $O(K)$.

At every time step, the algorithm has to insert an event for the new segment and process any events scheduled for the current time. Both these operations translate to inserting new events in EQ. The time to calculate the error of merging two segments is constant, and the same is true for calculating the time t_{merge} at which a merge becomes viable. (We can safely assume that the number of discontinuities $L-1$ of the amnesic function are far less than the number of segments and treat them as a constant. Therefore, t_{merge} can be computed in constant time.) The insertion of new events in EQ takes $O(\log K)$ time. Thus, the overall time complexity for each iteration is $O(\log K)$. □

In this section, we presented time and space complexity measures for the algorithms solving the *UAA* and *SAA* problems. These measures depend on the number of segments K used in the approximation. However, when using absolute amnesic functions, it is not possible to calculate in advance the value of K or even a range of values for K. The values that K is going to assume are determined by the data set and the amnesic function and can vary greatly. Nevertheless, we expect that in practice, the users will be able to make, for each application domain, judicious decisions about the absolute amnesic functions. These decisions will lead to reasonable values of K, and, subsequently, to small space and time complexity bounds.

6 Experimental Evaluation

We implemented our algorithms and conducted a series of experiments to evaluate their efficiency. We also implemented the optimal algorithm using dynamic programming and the traditional *BottomUp* algorithm for *PLA* [20], which is an offline algorithm, to compare against our techniques. Briefly, *BottomUp* works as follows: It starts by assigning a segment to each point in the time series. Then, at each consecutive step, it merges the two neighboring segments that will result in the least increase for the overall approximation error. For our experiments, we also make use of the amnesic functions, which are used to weigh the errors of the segments. Then, *BottomUp* operates on the weighted errors and proceeds as normal.

In order to evaluate our algorithms, we used an extensive set of real-world data sets. These are 40 data sets coming from diverse fields, including finance, medicine, biometrics, chemistry, astronomy, robotics, networking, and industry and covering the complete spectrum of stationary/nonstationary, noisy/smooth, cyclical/noncyclical, symmetric/asymmetric, etc. [1]. All the data sets have a length of 10,000 points and are studentized (i.e., they have zero mean and unit standard deviation). When not explicitly mentioned, the results reported are averages over all 40 data sets. For all the experiments shown here, we employed a piecewise linear amnesic function. The results for other amnesic functions are similar. In the following paragraphs, we first discuss the results for the relative amnesic functions and, subsequently, for the absolute amnesic functions.

6.1 Comparison to *BottomUp*

In the first set of experiments, we compare the performance of *GrAp* to *BottomUp*, which is essentially a comparison between an online and the corresponding offline algorithm.

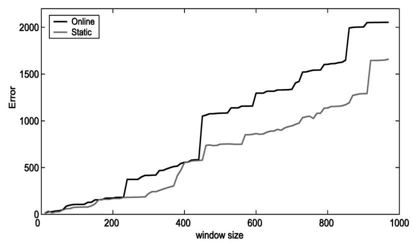
Fig. 11 depicts the approximation error and computation time for *GrAp-R* and *BottomUp* for a single data set (Space Shuttle STS-57). Similar trends are observed with all 40 data sets we used in our experiments. We use the unrestricted-window model and 10 segments, and we report the error and time as a function of the window size. Our online algorithm consistently provides approximations that are very close to those found by the offline algorithm. At the same time, our algorithm is much faster, requiring only constant time for processing every new point (actually, as we discussed in Section 4, the time is independent of N). On the other hand, *BottomUp* has time complexity $O(N \log N)$.

In the next set of experiments, we quantify the differences in the performance of the two algorithms. We report the cumulative relative error (CRE), which measures the relative increase in the cumulative error when using *GrAp-R*:

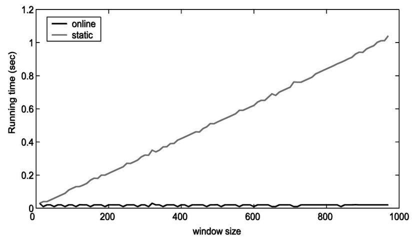
$$CRE = 100 \cdot \frac{\sum_{j=1}^{N}(E_{GrAp-R}(T[1..j]) - E_{BottomUp}(T[1..j]))}{\sum_{j=1}^{N} E_{BottomUp}(T[1..j])}.$$

The second measure of interest is the speedup, which measures how many times faster *GrAp-R* or *GrAp-A* is when compared to *BottomUp*:

$$Speedup = \frac{\sum_{j=1}^{N} Time_{BottomUp}(T[1..j])}{\sum_{j=1}^{N} Time_{GrAp}(T[1..j])}.$$

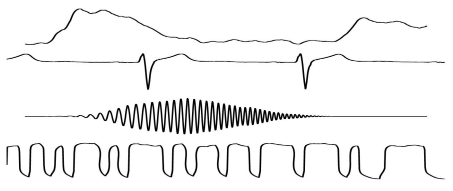


(a)



(b)

Fig. 11. Typical progression of (a) error and (b) time for *GrAp-R* and *BottomUp* (Space Shuttle STS-57 data set, unrestricted window).


(a)

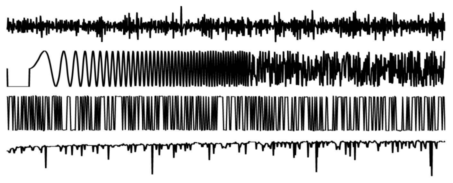

(b)

Fig. 12. Illustration of the data sets for which the quality of approximation achieved by *GrAp-R* is (a) closest to and (b) furthest away from that of *BottomUp*. (a) Top-four performers. (b) Bottom-four performers.

In Fig. 13, we depict CRE as a function of K and N for the unrestricted-window model. Using 50 segments, our algorithm performs within 3-11 percent of the offline algorithm for streams of length 1,000-3,000 points (Fig. 13a). Although for increasing N, we observe a very slow buildup of the relative error. In the experiment in Fig. 13b, the number of segments we use is 1 percent, 3 percent, and 5 percent of N. In this case, where the ratio N/K remains fixed, CRE remains relatively stable as we increase N. In both cases, our algorithm performs better as the number of segments increases.

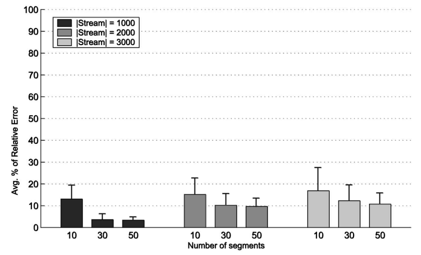
The graphs shown in Fig. 13 also depict the 95 percent confidence intervals for the error values we report. These intervals indicate that there is a small but noticeable variation in the performance of the algorithm across the diverse collection of the 40 data sets we used in our study. Indeed, a close inspection of the experimental results reveals that for the relatively smooth data sets, like the ones shown in Fig. 12a, *GrAp-R* performs extremely close (and, in some cases, identically) to *BottomUp* resulting in similar (or the same) approximation quality. Fig. 12a depicts the four data sets for which the performance of *GrAp-R* and *BottomUp* is most similar to each other. In other words, these are the examples where *GrAp-R* performs the best. For the more unstructured data sets, like the ones shown in Fig. 12b, the difference in the performance between *GrAp-R* and *BottomUp* is more pronounced. In Fig. 12b, we depict the four data sets for which the performance of *GrAp-R* and *BottomUp* is furthest away from each other. Note that these data sets are much more challenging to approximate, since they exhibit many sudden variations and unpredictable patterns. In these cases, *BottomUp* has the opportunity to make better global decisions that affect the overall approximation quality.

Fig. 14 shows the speedup that our algorithm achieves, which translates to one or two orders of magnitude faster execution than the offline algorithm (for the experiments we ran). We observe that the speedup increases significantly for decreasing K. This is because the amount of work that *GrAp-R* does remains almost constant (depends on $\log K$), while *BottomUp* requires lots of extra effort for smaller values of K. As expected, the speedup gets larger when we increase N.

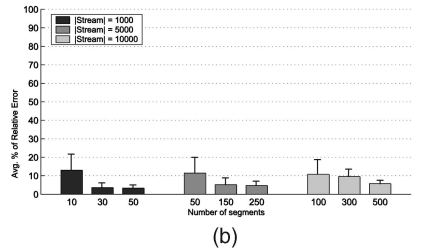
We also run the same experiments for the sliding-window model. Fig. 15a illustrates the results for the speedup, which in this case is mainly a function of the window size (K does not seem to affect the speedup in this case, because of the particular choices of K and the window size). The *GrAp-R* algorithm is 10-30 times faster than *BottomUp*. The results for the error are similar to those for the unrestricted-window model and are omitted for brevity.

The trends for the error and time remain the same as we increase K and N. All the above results show that the online algorithm achieves considerable benefits in terms of speed while losing little in approximation accuracy, when compared to the offline algorithm.

With the next experiment, we address a question that was raised in light of Lemma 2. In the sliding-window model, we temporarily allow the last segment of the approximation model to grow beyond the end of the window until it completely falls out of the boundaries of the window and we discard it. Fig. 15b depicts the average

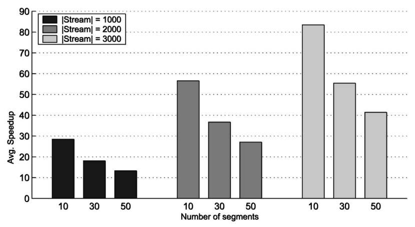


(a)

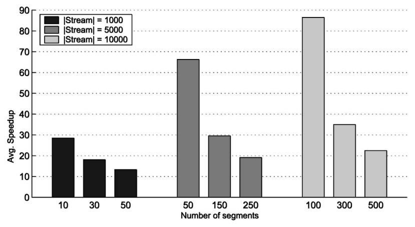


(b)

Fig. 13. Comparison of the approximation error between *GrAp-R* and *BottomUp* (unrestricted window). (a) Fixed K. (b) Fixed N/K.



(a)



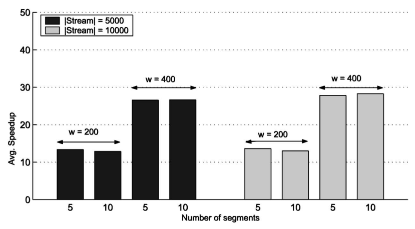
(b)

Fig. 14. Speedup of *GrAp-R* against *BottomUp* (unrestricted window). (a) Fixed K. (b) Fixed N/K.

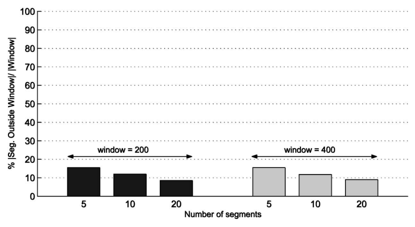
number of points outside the sliding window that are represented by the last segment, as a percentage of the window size. In all the cases we tested, this number ranges between 10-15 percent and therefore is not a restricting factor for our representation.

In the last set of experiments, we evaluate the performance of *GrAp-A*, which is the algorithm we propose for the absolute amnesic functions. We run the experiments with the unrestricted-window model and for three different stream sizes. In the case of *GrAp-A*, we are interested in minimizing the number of segments K used in the amnesic approximation. Therefore, when we compare this algorithm to *BottomUp*, we measure the cumulative relative increase in the required number of segments (*CRIS*):

$$CRIS = 100 \cdot \frac{\sum_{j=1}^{N}(K_{GrAp-A}(T[1..j]) - K_{BottomUp}(T[1..j]))}{\sum_{j=1}^{N} K_{BottomUp}(T[1..j])}.$$



(a)



(b)

Fig. 15. (a) Speedup of *GrAp-R* against *BottomUp* and (b) the number of excess points represented by *GrAp-R*, both for the sliding-window model.

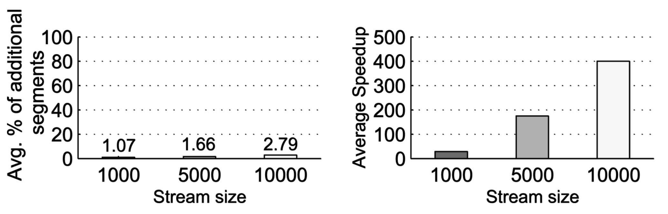


Fig. 16. Average increase in (a) the number of required segments and (b) speedup for *GrAp-A* against *BottomUp* (unrestricted window).

The results (refer to Fig. 16) show that *GrAp-A* is able to find a representation with a minimal number of additional segments when compared to the offline algorithm, that is, 1-3 percent more segments for streams of length 1,000-10,000. There is only a slight increase in *CRIS* as we move to longer streams. As in the case of relative amnesic functions, the speedup is considerable, with our algorithm running more than two orders of magnitude faster than *BottomUp*.

6.2 Comparison to Optimal

In this section, we investigate how our techniques compare to the optimal algorithm, *Opt*, implemented with dynamic programming. Unfortunately, due to the high time complexity of the optimal algorithm, this experiment is only possible for relatively small data sets.

We use the same set of 40 data sets and perform the experiment as follows: From each data set, we randomly extract a subsequence of length 512 and segment it into 16, 32, and 64 segments, using *BottomUp* and *Opt*. In the case of *GrAp-R*, we treat the data subsequences as streams, have the algorithm operate on those, and record the performance of the algorithm during the last iteration. We measure the relative increase in error for the *GrAp-R* and *BottomUp* algorithms, defined as $(E_{GrAp-R} - E_{Opt})/E_{Opt}$ and $(E_{BottomUp} - E_{Opt})/E_{Opt}$, respectively. A zero value for the relative error means that the algorithm under consideration has found the optimal solution. For each data set and each number of segments, we average the results over 10 randomly extracted subsequences and then average the relative error over all 40 data sets. The results are shown in Table 2. In the same table, we also report how much slower *Opt* executes when compared to *GrAp-R* and *BottomUp*. The results suggest that we lose little by using *GrAp-R* as opposed to *BottomUp*, since both algorithms manage to find solutions close to the optimal. Note that this excellent performance comes from tremendous savings in terms of computational cost. The optimal algorithm is several orders of magnitude slower than *GrAp-R* and is clearly inapplicable for an online environment.

7 Related Work

There exists an extensive literature in the area of time series approximation [22]. Some of the representations that have been proposed include the Fourier transform [13], [30], many different wavelets [28], [9], piecewise polynomials [38], [8], singular value decomposition [8], and symbolic approximations [2]. Many of the above approximation techniques have been adapted to work in an online fashion. For example, piecewise constant approximation can be created online with little loss of accuracy [25], as well as *DFT* [40]. Most of other time series representations have been or could trivially be calculated in an incremental

TABLE 2
Comparison among *GrAp-R*, *BottomUp*, and Optimal

K	$(E_{GrAp-R} - E_{Opt})/E_{Opt}$	$Time_{Opt}/Time_{GrAp-R}$
16	0.102	1857
32	0.083	1886
64	0.064	1912
K	$(E_{BottomUp} - E_{Opt})/E_{Opt}$	$Time_{Opt}/Time_{BottomUp}$
16	0.058	112
32	0.051	137
64	0.042	173

fashion [20]. There has also appeared work on data stream summarization, using wavelets [15] and histograms [16]. Cohen and Strauss [11] present a framework for maintaining time-decaying stream aggregates such as sum and average.

Even though each year seems to produce new representations for time series [7], [26], the interest in using PLA has not waned. If anything, the opposite is true. Recent years have seen an explosion of interest in using PLA to support a wide variety of data mining and indexing tasks. For example, in the previous year alone, PLA has been used to support a finite state automaton to simulate respiratory motion [36], to do forecasting of the stock market [35], to support anomaly detection in space telemetry [31], and to produce text-based weather summaries [32]. This diverse list merely hints at the broad applicability of PLA to real-world problems.

Chen et al. [10] describe a framework for multidimensional regression analysis of time series with a tilt time frame. Nevertheless, they do not explicitly tailor their representations to match different amnesic functions. Bulut and Singh proposed using wavelets to represent "data streams which are biased towards the more recent values" [6] and successfully implemented their method. Although the bias to more recent values can be seen as a special case of an amnesic function, the particular function is dictated by the hierarchical nature of the wavelet transform. A subsequent study [39] generalizes on these ideas, by decoupling the approximation of the time series from a particular dimension-reduction algorithm, but requires the user to specify how the available memory will be used for the approximation. Our work removes all the restrictions inherent in the above approaches. The framework we propose takes into account the form of the amnesic function as an integral part of the problem and provides an effective and efficient solution for a much more general class of amnesic functions.

There has also been relevant work in machine learning and, more specifically, in the neural network community, where the main goal is to model time-varying patterns in time series [3], [12]. What is different in our approach is that we propose a summarization technique using an arbitrary user-defined amnesic function that is compatible with several existing distance measures and can be directly used by a multitude of indexing and data mining algorithms.

8 Conclusions

We have introduced the first method to allow the online approximation of streaming time series, which allows arbitrary user-defined reduction of quality with time. This kind of approximation is of increasing importance in many diverse application domains such as mobile and real-time devices. We justified our choice of representation with extensive comparisons to competing techniques and described how we can adapt to allow arbitrary amnesic functions for streaming data. We empirically evaluated our algorithms with extensive experiments on 40 different data sets. The results show that our algorithms offer significant performance improvements over the direct computational approach while maintaining the quality of the approximation close to optimal. Possible directions for future work include supporting indexed similarity search and other queries on our representation.

Acknowledgments

This work was partially supported by the FP7 EU Large-scale Integrating Project **OKKAM—Enabling a Web of Entities** (contract no.~ICT-215032). For more details, visit http://www.okkam.org.

References

[1] *The UCR Time Series Data Mining Archive,* Computer Science and Eng. Dept., Univ. of California, Riverside, http://www.cs.ucr.edu/~eamonn/TSDMA/, 2002.

[2] H. André-Jönsson and D. Badal, "Using Signature Files for Querying Time-Series Data," *Principles of Data Mining and Knowledge Discovery,* pp. 211-220, June 1997.

[3] A. Barreto, A. Araujo, and S. Kremer, "A Taxonomy for Spatiotemporal Connectionist Networks Revisited: The Unsupervised Case," *Neural Computation,* vol. 15, pp. 1255-1320, 2003.

[4] J. Basch, "Kinetic Data Structures," PhD dissertation, Dept. Computer Science, Stanford Univ., 1999.

[5] R. Bellman, "On the Approximation of Curves by Line Segments Using Dynamic Programming," *Comm. ACM,* vol. 4, no. 6, p. 284, 1961.

[6] A. Bulut and A.K. Singh, "SWAT: Hierarchical Stream Summarization in Large Networks," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03),* pp. 303-314, Mar. 2003.

[7] Y. Cai and R.T. Ng, "Indexing Spatio-Temporal Trajectories with Chebyshev Polynomials," *Proc. ACM SIGMOD '04,* pp. 599-610, June 2004.

[8] K. Chakrabarti, E.J. Keogh, S. Mehrotra, and M.J. Pazzani, "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases," *ACM Trans. Database Systems,* vol. 27, no. 2, pp. 188-228, 2002.

[9] K. Chan and W. Fu, "Efficient Time Series Matching by Wavelets," *Proc. 15th Int'l Conf. Data Eng. (ICDE '99),* pp. 126-133, Mar. 1999.

[10] Y. Chen, G. Dong, J. Han, B.W. Wah, and J. Wang, "Multi-Dimensional Regression Analysis of Time-Series Data Streams," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),* pp. 323-334, Aug. 2002.

[11] E. Cohen and M. Strauss, "Maintaining Time-Decaying Stream Aggregates," *Proc. 22nd ACM Symp. Principles of Database Systems (PODS '03),* pp. 223-233, June 2003.

[12] B. de Vries and J.C. Principe, "The Gamma Model—A New Neural Model for Temporal Processing," *Neural Networks,* vol. 5, pp. 565-576, 1992.

[13] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD '94,* pp. 419-429, May 1994.

[14] X. Ge and P. Smyth, "Segmental Semi-Markov Models for Endpoint Detection in Plasma Etching," *Proc. AEC/APC Symp.,* Sept. 2000.

[15] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* pp. 79-88, 2001.

[16] S. Guha and N. Koudas, "Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation," *Proc. 18th Int'l Conf. Data Eng. (ICDE '02),* pp. 567-576, Mar. 2002.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2001.

[18] R. Hogg, A. Rankin, M. McHenry, D. Helmick, C. Bergh, S. Roumeliotis, and L. Matthies, "Sensors and Algorithms for Small Robot Leader/Follower Behavior," *Proc. SPIE 15th AeroSense Symp.*, Apr. 2001.

[19] J. Hunter and N. McIntosh, "Knowledge-Based Event Detection in Complex Time Series Data," *Artificial Intelligence in Medicine and Medical Decision Making*, pp. 271-280, June 1999.

[20] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An Online Algorithm for Segmenting Time Series," *Proc. 17th Int'l Conf. Data Mining (ICDE '01)*, pp. 289-296, Nov. 2001.

[21] E. Keogh, S. Lonardi, and W. Chiu, "Finding Surprising Patterns in a Time Series Database in Linear Time and Space," *Proc. ACM SIGKDD '02*, pp. 550-556, July 2002.

[22] E.J. Keogh and S. Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Proc. ACM SIGKDD '02*, pp. 102-111, July 2002.

[23] E.J. Keogh and M.J. Pazzani, "An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback," *Proc. ACM SIGKDD '98*, pp. 239-243, Aug. 1998.

[24] A. Koski, M. Juhola, and M. Meriste, "Syntactic Recognition of ECG Signals by Attributed Finite Automata," *Pattern Recognition*, vol. 28, no. 12, pp. 1927-1940, 1995.

[25] I. Lazaridis and S. Mehrotra, "Capturing Sensor-Generated Time Series with Quality Guarantees," *Proc. 19th Int'l Conf. Data Eng. (ICDE '03)*, pp. 429-440, Mar. 2003.

[26] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms," *Proc. Eighth ACM SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD '03)*, June 2003.

[27] S. Park and W.W. Chu, "Discovering and Matching Elastic Rules from Sequence Databases," *Fundamenta Informaticae*, vol. 47, nos. 1-2, pp. 75-90, 2001.

[28] I. Popivanov and R.J. Miller, "Similarity Search over Time Series Data Using Wavelets," *Proc. Int'l Conf. Data Eng.*, pp. 802-813, Feb. 2002.

[29] W. Pugh, "Skiplists: A Probabilistic Alternative to Balanced Trees," *Comm. ACM*, vol. 33, no. 6, pp. 668-676, 1990.

[30] D. Rafiei, "On Similarity-Based Queries for Time Series Data," *Proc. 15th Int'l Conf. Data Eng. (ICDE '99)*, Mar. 1999.

[31] S. Salvador, P. Chan, and J. Brodie, "Learning States and Rules for Time Series Anomaly Detection," *Proc. 17th Int'l FLAIRS Conf.*, pp. 300-305, May 2004.

[32] S. Somayajulu, E. Reiter, and I. Davy, "SumTime-Mousam: Configurable Marine Weather Forecast Generator," *Expert Update*, vol. 6, no. 3, pp. 4-10, 2004.

[33] D. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole, "Research Challenges in Environmental Observation and Forecasting Systems," *Proc. ACM MobiCom '00*, Aug. 2000.

[34] H.J.L.M. Vullings, M.H.G. Verhaegen, and H.B. Verbruggen, "ECG Segmentation Using Time-Warping," *Proc. Second Int'l Symp. Intelligent Data Analysis (IDA '97)*, pp. 275-285, Aug. 1997.

[35] H. Wu, B. Salzberg, and D. Zhang, "Online Event-Driven Subsequence Matching over Financial Data Streams," *Proc. ACM SIGMOD '04*, pp. 23-34, June 2004.

[36] H. Wu, G.C. Sharp, B. Salzberg, D. Kaeli, H. Shirato, and S.B. Jiang, "A Finite State Model for Respiratory Motion Analysis in Image Guided Radiation Therapy," *Physics in Medicine and Biology*, vol. 49, no. 23, pp. 5357-5372, 2004.

[37] Y.-L. Wu, D. Agrawal, and A. El Abbadi, "A Comparison of DFT and DWT Based Similarity Search in Time-Series Databases," *Proc. Ninth ACM Int'l Conf. Information and Knowledge Management (CIKM '00)*, pp. 488-495, Nov. 2000.

[38] B. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary LP-Norms," *Proc. 26th Int'l Conf. Very Large Data Bases (VLDB '00)*, pp. 385-394, Sept. 2000.

[39] Y. Zhao and S. Zhang, "Generalized Dimension-Reduction Framework for Recent-Biased Time Series Analysis," *IEEE Trans. Knowledge and Data Eng.*, vol. 18, no. 2, pp. 231-244, Feb. 2006.

[40] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02)*, pp. 358-369, Aug. 2002.

Themis Palpanas received the BS degree from the National Technical University of Athens, Greece, and the MSc and PhD degrees from the University of Toronto, Canada. He is a faculty member of computer science at the University of Trento, Italy. Before joining the University of Trento, he worked at the IBM T.J. Watson Research Center and the University of California, Riverside, and visited Microsoft Research and the IBM Almaden Research Center. His interests include data management, data analysis, and streaming algorithms. He is the author of five US patents, and is serving on the program committees of several top database and data mining conferences.

Michail Vlachos received the BSc degree (with highest honors) in informatics from the Aristotle University of Thessaloniki, Greece, and the PhD degree from the University of California, Riverside. He is a research staff member at the IBM T.J. Watson Research Center, New York. His research interests expand on the areas of data mining, databases, time series, data visualization, clustering, and classification of multimedia data. He has coauthored more than 40 articles on the above areas. He has been a recipient of the Fulbright Foundation scholarship for graduate studies.

Eamonn Keogh is an associate professor of computer science at the University of California, Riverside. His research interests include data mining, machine learning, and information retrieval. Several of his papers have won best paper awards, including papers at ACM SIGKDD, ICDM, and ACM SIGMOD. He is the recipient of a five-year US National Science Foundation Career Award for efficient discovery of previously unknown patterns and relationships in massive time series databases.

Dimitrios Gunopulos received the PhD degree from Princeton University. He has held regular and visiting positions at the Max-Planck-Institute for Informatics, the University of Helsinki, the IBM Almaden Research Center, the Department of Computer Science and Engineering at the University of California, Riverside, and the Department of Informatics and Telecommunications at the University of Athens. His research is in the areas of data mining and knowledge discovery in databases, databases, sensor networks, peer-to-peer systems, and algorithms. He has coauthored more than 100 journal and conference papers that have been widely cited. He has served as a program committee chair for several leading databases and data mining conferences, and he s currently an associate editor for the *IEEE Transactions on Knowledge and Data Engineering*, the *IEEE Transactions on Parallel and Distributed Systems*, and the *ACM Transactions on Knowledge Discovery from Data*. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.