Time-Decaying Representations of Streaming Time Series

Themistoklis Palpanas[1], Michail Vlachos[1], Eamonn Keogh[1], Dimitrios Gunopulos[1], and Wagner Truppel[1]

University of California, Riverside
Riverside CA 92521 USA
{themis,mvlachos,eamonn,dg,wagner}@cs.ucr.edu

Abstract. During the last years we have witnessed a wealth of research on approximate representations for time series. The vast majority of the proposed approaches represent each value with approximately equal fidelity, which may not be always desirable. For example, mobile devices and real time sensors have brought home the need for representations that can approximate the data with fidelity proportional to its age. We call such time-decaying representations *amnesic*.
In this work, we introduce a novel representation of time series that can represent arbitrary, user-specified time-decaying functions. We propose online algorithms for our representation, and discuss their properties. The algorithms we describe are designed to work on both the entire stream of data, or on a sliding window of the data stream. Finally, we perform an extensive empirical evaluation on 40 datasets, and show that our approach can efficiently maintain a high quality amnesic approximation.

1 Introduction

Time series are one of the most frequently encountered forms of data. Many applications in diverse domains, produce voluminous amounts of time series [24, 22]. The sheer number and size of the time series we need to manipulate in many of the real-world applications dictates the need for a more compact representation of time series than the raw data itself. A plethora of representations have been proposed for time series approximation [14].

The problem of approximating time series becomes more interesting and challenging in the context of streaming time series, where data values are continuously generated. In this situation, we cannot apply approximation techniques that require knowledge of the entire series. Furthermore, all current time series representations treat every point of the time series equally, which means that 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. In these cases, while we are willing to accept some margin of error in the approximation, we would like the most recent data to have 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, such as the Environmental Observation and Forecasting System [22], the robots developed by NASA for an urban setting [10], classification algorithms for identifying denial of service attacks [12], inductive learning [17], and others.

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 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. In Figure 1 we show how the approximation of a time series changes as a function of time. In this example, we use an unrestricted window to approximate the Space Shuttle STS-57 dataset, using piecewise linear approximation with ten linear segments. The time progresses form right to left (i.e., the most recent point is the left-most point). We observe that the approximation of the most recent points always remains accurate, while it gracefully degrades at each time step for the older points.

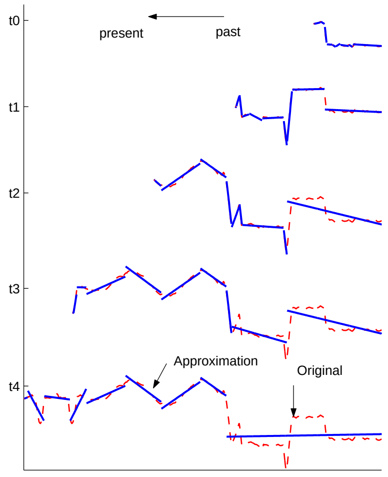


Fig. 1. Example of online amnesic approximation.

In this paper, we describe a framework for online amnesic approximation of streaming time series. While some recent work [5, 2] has proposed tools and techniques for computing special cases of amnesic approximations of time series, as we discuss in Section 6, these solutions are specific and rather restrictive in the variety of applications they can accommodate. In particular, the types of amnesic functions they can use are dictated by the *representation* of the time series. 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 an optimization problem, where we wish to minimize the reconstruction error given the available amount of memory for the approximation. 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. 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 datasets. 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 Section 4, and Section 5 discusses the experimental evaluation. Section 6 reviews related work, and Section 7 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*. In the streaming environment we may be interested in approximating the entire time series seen so far, *unrestricted window*, or a fixed number of the last values of the time series, *sliding window*.

Several techniques have been proposed in the literature for the approximation of time series, including *Discrete Fourier Transform (DFT)* [21, 7], *Discrete Cosine Transform (DCT)*, *Piecewise Aggregate Approximation (PAA)* [23], *Discrete Wavelet Transform (DWT)* [20, 4], *Adaptive Piecewise Constant Approximation (APCA)* [3, 18], *Piecewise Linear Approximation (PLA)* [15], *Piecewise Quadratic Approximation (PQA)*, and others.

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. 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 [24], it is not clear that it can be adapted to perform amnesic approximation. The same is true for *DCT* as well. We decided to use *PLA*, because is is already widely used and accepted [11, 16]. Moreover, it has been documented that *PLA* performs at least as good as all the other approaches mentioned above [14, 3], and also offers a superior quality of visualization of the time series.

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 [15]. Assume 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. In addition, suppose we use a single line segment to approximate all the points 1 to N, call it $s_{1,2}$. The above three lines are depicted in the top graph of Figure 2. 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 the bottom graph of Figure 2. The residual error of this new line is $E(\overline{s_{1,2}})$. 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. It turns out that we can also calculate $E(s_{1,2})$ without the need to refer to the original N points.

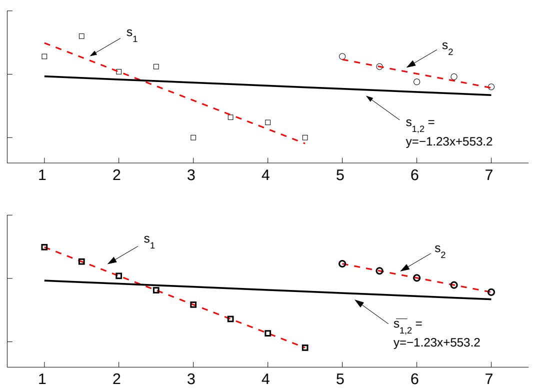


Fig. 2. Combining two regression lines.

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. 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 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, $E(s_{1,2}) = E(s_1) + E(s_2) + E(\overline{s_{1,2}})$.*

Another interesting property of *PLA* is that for the computation of the error $E(\overline{s_{1,2}})$ 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 $E(\overline{s_{1,2}})$ in constant time, according to the following lemma.

Lemma 1. [Computing the Error Between Two Segments.] *The error, $E(\overline{s_{1,2}})$, 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*[1] *in time $O(1)$, regardless of the length of the line segments.*

3 Problem Formulation

3.1 Amnesic Functions

The role of an amnesic function $A(x)$, is to specify 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 corresponds to position $x = 0$ of the amnesic function. Note that $A(x)$ is only defined for $x \geq 0$, since $t_i \leq t_N$. A key property that an amnesic function has to satisfy is the *monotonicity* property.

Definition 1. *An amnesic function, $A(x)$, is called monotonic iff $A(x) \leq A(x+1)$, for every value of x.*

The monotonicity property poses a natural restriction. It ensures that if at time t we can tolerate some error $E^t(T[i])$ in the approximation of point $T[i]$, 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 Figure 3).

Piecewise Constant: The *piecewise constant function* is the simplest form that an amnesic function can assume (with the exception of the constant function, which is a trivial case, and we do not discuss it here). It has the following general form.

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

where $c_1, \ldots, c_L$ are constants, such that $c_1 < \ldots < c_L$. 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: $A(x) = \alpha x + \beta$, $\alpha, \beta > 0$.

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

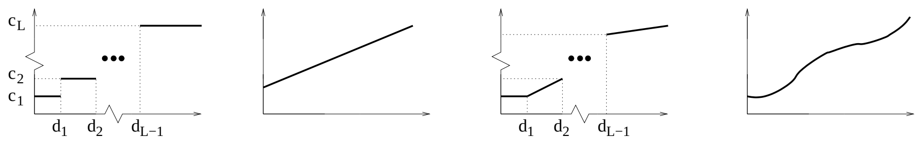
$$A(x) = \begin{cases} \alpha_1 x + \beta_1 & , 0 \leq x < d_1; \\ \ldots & \\ \alpha_L x + \beta_L & , d_{L-1} \leq x, \end{cases}$$

where $\alpha_j \in [0, \pi/2), 1 \leq j \leq L, \beta_1 > 0$, and $\beta_2 = \alpha_1 d_1 + \beta_1, \ldots, \beta_L = \alpha_{L-1} d_{L-1} + \beta_{L-1}$.

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.

[1] The formula requires the introduction of additional notation, and we omit it due to lack of space.



(a) piecewise constant (b) linear (c) piecewise linear (d) continuous

Fig. 3. The different classes of amnesic functions.

Relative: Relative amnesic functions determine the relative approximation error we can tolerate. When we use a relative amnesic function, we weight the error of some data point by the inverse of the amnesic function corresponding to that point, so that $E(x) = E(x)/RA(x)$.

Absolute: Absolute amnesic functions specify the *maximum* allowable error for the approximation. The error $E(x)$ should satisfy the inequality $E(x) \leq AA(x)$.

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 K and $RA(x)$, find an approximation Q using K segments that at each time step minimizes the error $E(T[1..N]) = \sum_{j=1}^{K} (E(s_j)/RA(t_N - t_{s_j}))$.

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

Note that in the *URA* and *UAA* problems the optimization objective is different. In the *URA* problem we seek to minimize the approximation error , while in the *UAA* problem we want to minimize the space. Following the definition of the problems for the unrestricted window, we can define the corresponding problems for the sliding window model.

In the next section, we simply outline the solutions we propose to the above problems. We omit a detailed explanation of our algorithms in the interest of space. A more extended discussion can be found elsewhere [19].

4 Amnesic Approximation

4.1 Relative Amnesic Functions

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 Figure 4.

```
1 let $H$ be a min-priority queue, and $EQ = \emptyset$ 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 min 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**($EQ$, $t_N$, $s_m$, $s_{m+1}$, $s_{m,m+1}$);
```

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

The *GrAp-R* algorithm also makes use of a time-event queue EQ to keep track of the way that the dependencies among the segments used for the approximation change.

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. The queue EQ flags the times at which the segments cross a discontinuity in the amnesic function. 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.

In the case of linear amnesic functions, 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. Therefore, we insert those times in EQ, and update the heap accordingly.

Assume we have a piecewise linear amnesic function, which is comprised of L sections. Then, we treat each section separately, as in the case of linear amnesic functions.

When the amnesic function is continuous, we identify two cases. First, when the amnesic function has a closed form formula, we 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 piecewise linear approximation using L sections. Then, we proceed as with the piecewise linear amnesic functions.

4.2 Absolute Amnesic Functions

When we use absolute amnesic functions, there is no restriction in the number of segments that we may use for the approximation, and there is no need to maintain a heap structure on the adjacent pairs of segments. We call the new algorithm *GrAp-A*, and it works as follows. During each time step, the algorithm assigns the new data point of the time series to a new segment, s_i, by itself. Then, it tries to merge s_i with its adjacent segment. If the segment that results from the merge has error less than what is specified by the absolute amnesic function, then the merge is realized. In either case, the next step involves the update of the time-event queue EQ. In EQ we have to schedule 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, we process any merges that are scheduled to happen at the current time, and update the queue.

5 Experimental Evaluation

We implemented our algorithms, and the traditional *BottomUp* algorithm for *PLA* [13], to compare against our techniques. In order to evaluate our algorithms, we used an extensive set of real-world datasets. These are 40 datasets coming from diverse fields, including finance, medicine, biometrics, chemistry, astronomy, robotics, networking and industry, and covering the complete spectrum of stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric, etc.. The results reported are averages over all 40 datasets.

In the following experiments, we quantify the relative 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 hom many times faster *GrAp-R* is when compared to *BottomUp*.

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

In the experiments of Figure 5, the number of segments we use is $1\%, 3\%$, and 5% of N. We observe that CRE remains relatively stable as we increase N, with our algorithm performing within $3\% - 13\%$ of the offline algorithm. At the same time, our algorithm runs one or two orders of magnitude faster than the offline algorithm. 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.

In the next set of experiments, we evaluate the performance of *GrAp-A*. During these experiments, we use an absolute amnesic function, and we are interested in minimizing the number of segments, K, used in the amnesic approximation. We compare our algorithm to *BottomUp*, and we measure the speedup and the cumulative relative increase in the required number of segments, *CRIS*.

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

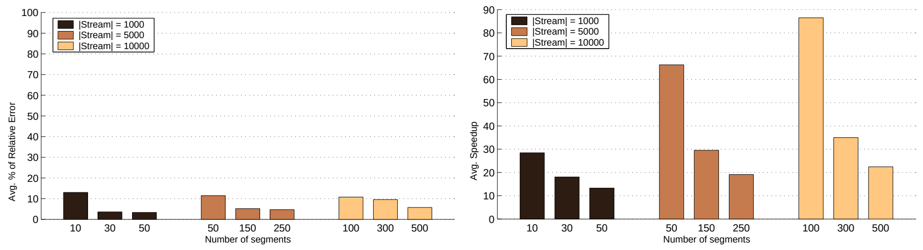


Fig. 5. Error (left) and speedup (right) for *GrAp-R* and *BottomUp*.

These experiments (refer to Figure 6) 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 $1\% - 3\%$ more segments for streams of length $1000 - 10000$. 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*.

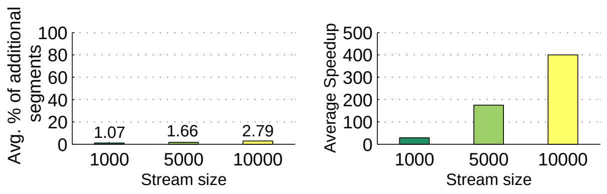


Fig. 6. Error (left) and speedup (right) for *GrAp-A* and *BottomUp*.

6 Related Work

There exists an extensive literature in the area of time series approximation [14, 7, 21, 20, 4, 23, 3, 3, 1]. 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 [18], as well as *DFT* [24]. Most of the time series representations have been, or could trivially be, calculated in an incremental fashion [13]. There is also work on data stream summarization, using wavelets [8] and histograms [9]. Cohen and Strauss [6] present a framework for maintaining time-decaying stream aggregates, such as sum and average. Chen et al. [5] describe a framework for multi-dimensional regression analysis of time series with a tilt time frame. Yet, 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” [2]. Although the bias to more recent values can be seen as a special case of an amnesic function, the particular function is completely dictated by the hierarchical nature of the wavelet transform. Our work removes all the restrictions inherent in the above approaches.

7 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 proposed efficient algorithms, and we empirically evaluated them with extensive experiments on 40 different datasets. The results show that our algorithms offer significant performance improvements over the direct computational approach, while maintaining a very good quality of approximation.

References

[1] H. André-Jönsson and D. Badal. Using Signature Files for Querying Time-Series Data. In *Principles of Data Mining and Knowledge Discovery*, pages 211–220, Trondheim, Norway, June 1997.

[2] Ahmet Bulut and Ambuj K. Singh. SWAT: Hierarchical Stream Summarization in Large Networks. In *International Conference on Data Engineering*, pages 303–314, Bangalore, India, March 2003.

[3] Kaushik Chakrabarti, Eamonn J. Keogh, Sharad Mehrotra, and Michael J. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM Transactions on Database Systems*, 27(2):188–228, 2002.

[4] K. Chan and W. Fu. Efficient Time Series Matching by Wavelets. In *International Conference on Data Engineering*, pages 126–133, Sydney, Australia, March 1999.

[5] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *VLDB International Conference*, pages 323–334, Hong Kong, China, August 2002.

[6] Edith Cohen and Martin Strauss. Maintaining Time-Decaying Stream Aggregates. In *ACM PODS International Conference*, pages 223–233, San Diego, CA, USA, jun 2003.

[7] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *ACM SIGMOD International Conference*, pages 419–429, Minneapolis, MI, USA, May 1994.

[8] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *VLDB International Conference*, pages 79–88, Rome, Italy, sep 2001.

[9] Sudipto Guha and Nick Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. In *International Conference on Data Engineering*, pages 567–576, San Jose, CA, USA, March 2002.

[10] R. Hogg, A. Rankin, M. McHenry, D. Helmick, C. Bergh, S. Roumeliotis, and L. Matthies. Sensors and Algorithms for Small Robot Leader/Follower Behavior. In *SPIE AeroSense Symposium*, Orlando, FL, USA, April 2001.

[11] Jim Hunter and Neil McIntosh. Knowledge-Based Event Detection in Complex Time Series Data. In *Artificial Intelligence in Medicine and Medical Decision Making*, pages 271–280, Aalborg, Denmark, June 1999.

[12] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A Framework for Classifying Denial of Service Attacks. In *ACM SIGCOMM Conference*, page To appear, Karlsruhe, Germany, August 2003.

[13] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An Online Algorithm for Segmenting Time Series. In *IEEE International Conference on Data Mining*, pages 289–296, San Jose, CA, USA, November 2001.

[14] Eamonn J. Keogh and Shruti Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *International Conference on Knowledge Discovery and Data Mining*, pages 102–111, Edmonton, Canada, July 2002.

[15] Eamonn J. Keogh and Michael J. Pazzani. An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback. In *International Conference on Knowledge Discovery and Data Mining*, pages 239–243, New York, NY, USA, August 1998.

[16] A. Koski, M. Juhola, and M. Meriste. Syntactic Recognition of ECG Signals By Attributed Finite Automata. *Pattern Recognition*, 28(12):1927–1940, 1995.

[17] Ivan Koychev. Gradual Forgetting for Adaptation to Concept Drift. In *ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, Berlin, Germany, August 2000.

[18] Iosif Lazaridis and Sharad Mehrotra. Capturing Sensor-Generated Time Series with Quality Guarantees. In *International Conference on Data Engineering*, pages 429–440, Bangalore, India, March 2003.

[19] Themistoklis Palpanas, Michail Vlachos, Eamonn Keogh, Dimitrios Gunopulos, and Wagner Truppel. Online Amnesic Approximation of Streaming Time Series. In *International Conference on Data Engineering*, Boston, MA, USA, March 2004.

[20] Ivan Popivanov and Renée J. Miller. Similarity Search Over Time Series Data Using Wavelets. In *International Conference on Data Engineering*, pages 802–813, San Jose, CA, USA, February 2002.

[21] Davood Rafiei. On Similarity-Based Queries for Time Series Data. In *International Conference on Data Engineering*, Sydney, Australia, March 1999.

[22] David Steere, Antonio Baptista, Dylan McNamee, Calton Pu, and Jonathan Walpole. Research Challenges in Environmental Observation and Forecasting Systems. In *Mobile Computing and Networking*, Boston, MA, USA, August 2000.

[23] B. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary LP-Norms. In *VLDB International Conference*, pages 385–394, Cairo, Egypt, September 2000.

[24] Yunyue Zhu and Dennis Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB International Conference*, pages 358–369, Hong Kong, China, August 2002.