An Online Algorithm for Segmenting Time Series

Eamonn Keogh Selina Chu David Hart Michael Pazzani
Department of Information and Computer Science
University of California, Irvine, California 92697 USA
{eamonn, selina, dhart, pazzani}@ics.uci.edu

Abstract

In recent years, there has been an explosion of interest in mining time series databases. As with most computer science problems, representation of the data is the key to efficient and effective solutions. One of the most commonly used representations is piecewise linear approximation. This representation has been used by various researchers to support clustering, classification, indexing and association rule mining of time series data. A variety of algorithms have been proposed to obtain this representation, with several algorithms having been independently rediscovered several times. In this paper, we undertake the first extensive review and empirical comparison of all proposed techniques. We show that all these algorithms have fatal flaws from a data mining perspective. We introduce a novel algorithm that we empirically show to be superior to all others in the literature.

1. Introduction

In recent years, there has been an explosion of interest in mining time series databases. As with most computer science problems, representation of the data is the key to efficient and effective solutions. Several high level representations of time series have been proposed, including Fourier Transforms [1,13], Wavelets [4], Symbolic Mappings [2, 5, 24] and Piecewise Linear Representation (PLR). In this work, we confine our attention to PLR, perhaps the most frequently used representation [8, 10, 12, 14, 15, 16, 17, 18, 20, 21, 22, 25, 27, 28, 30, 31].

Intuitively Piecewise Linear Representation refers to the approximation of a time series T, of length *n*, with K straight lines. Figure 1 contains two examples. Because K is typically much smaller that *n*, this representation makes the storage, transmission and computation of the data more efficient. Specifically, in the context of data mining, the piecewise linear representation has been used to:

- Support fast exact similarly search [13].
- Support novel distance measures for time series, including "fuzzy queries" [27, 28], weighted queries [15], multiresolution queries [31, 18], dynamic time warping [22] and relevance feedback [14].
- Support concurrent mining of text and time series [17].
- Support novel clustering and classification algorithms [15].
- Support change point detection [29, 8].

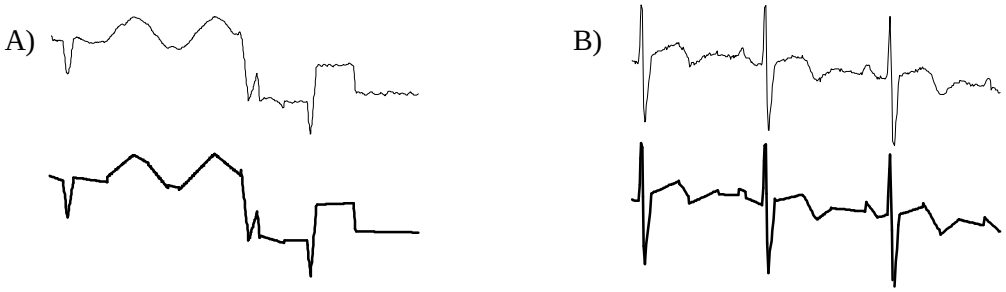

Figure 1: Two time series and their piecewise linear representation. A) Space Shuttle Telemetry. B) Electrocardiogram (ECG).

Surprisingly, in spite of the ubiquity of this representation, with the exception of [27], there has been little attempt to understand and compare the algorithms that produce it. Indeed, there does not even appear to be a consensus on what to call such an algorithm. For clarity, we will refer to these types of algorithm, which input a time series and return a piecewise linear representation, as *segmentation algorithms*.

The segmentation problem can be framed in several ways.

- Given a time series T, produce the best representation using only K segments.
- Given a time series T, produce the best representation such that the maximum error for any segment does not exceed some user-specified threshold, `max_error`.
- Given a time series T, produce the best representation such that the combined error of all segments is less than some user-specified threshold, `total_max_error`.

As we shall see in later sections, not all algorithms can support all these specifications.

Segmentation algorithms can also be classified as batch or online. This is an important distinction because many data mining problems are inherently dynamic [30, 12].

Data mining researchers, who needed to produce a piecewise linear approximation, have typically either independently rediscovered an algorithm or used an approach suggested in related literature. For example, from the fields of cartography or computer graphics [6, 9, 26].

In this paper, we review the three major segmentation approaches in the literature and provide an extensive empirical evaluation on a very heterogeneous collection of datasets from finance, medicine, manufacturing and science. The major result of these experiments is that that only online algorithm in the literature produces very poor approximations of the data, and that the only algorithm that consistently produces high quality results and scales linearly in the size of the data is a batch algorithm. These results motivated us to introduce a new algorithm that scales linearly in the size of the data set, is online, and produces high quality approximations.

The rest of the paper is organized as follows. In Section 2, we provide an extensive review of the algorithms in the literature. We explain the basic approaches, and the various modifications and extensions by data miners. In Section 3, we provide a detailed empirical comparison of all the algorithms. We will show that the most popular algorithms used by data miners can in fact produce very poor approximations of the data. The results will be used to motivate the need for a new algorithm that we will introduce and validate in Section 4. Section 5 offers conclusions and directions for future work.

2. Background and Related Work

In this section, we describe the three major approaches to time series segmentation in detail. Almost all the algorithms have 2 and 3 dimensional analogues, which ironically seem to be better understood. A discussion of the higher dimensional cases is beyond the scope of this paper. We refer the interested reader to [9], which contains an excellent survey.

Although appearing under different names and with slightly different implementation details, most time series segmentation algorithms can be grouped into one of the following three categories.

- **Sliding Windows**: A segment is grown until it exceeds some error bound. The process repeats with the next data point not included in the newly approximated segment.
- **Top-Down**: The time series is recursively partitioned until some stopping criteria is met.
- **Bottom-Up**: Starting from the finest possible approximation, segments are merged until some stopping criteria is met.

Table 1 contains the notation used in this paper.

T	A time series in the form $t_1,t_2,\ldots,t_n$
T[*a*:*b*]	The subsection of T from *a* to *b*, $t_a,t_{a+1},\ldots,t_b$
Seg_TS	A piecewise linear approximation of a time series of length *n* with K segments. Individual segments can be addressed with Seg_TS(i).
create_segment(T)	A function which takes in a time series and returns a linear segment approximation of it.
calculate_error(T)	A function which takes in a time series and returns the approximation error of the linear segment approximation of it.

Table 1: The notation used in this paper

Given that we are going to approximate a time series with straight lines, there are at least two ways we can find the approximating line.

- **Linear Interpolation**: Here the approximating line for the subsequence T[*a*:*b*] is simply the line connecting t_a and t_b. This can be obtained in constant time.
- **Linear Regression:** Here the approximating line for the subsequence T[*a*:*b*] is taken to be the best fitting line in the least squares sense [27]. This can be obtained in time linear in the length of segment.

The two techniques are illustrated in Figure 2. Linear interpolation tends to closely align the endpoint of consecutive segments, giving the piecewise approximation a "smooth" look. In contrast, piecewise linear regression can produce a very disjointed look on some datasets. The aesthetic superiority of linear interpolation, together with its low computational complexity has made it the technique of choice in computer graphic applications [9]. However, the quality of the approximating line, in terms of Euclidean distance, is generally inferior to the regression approach.

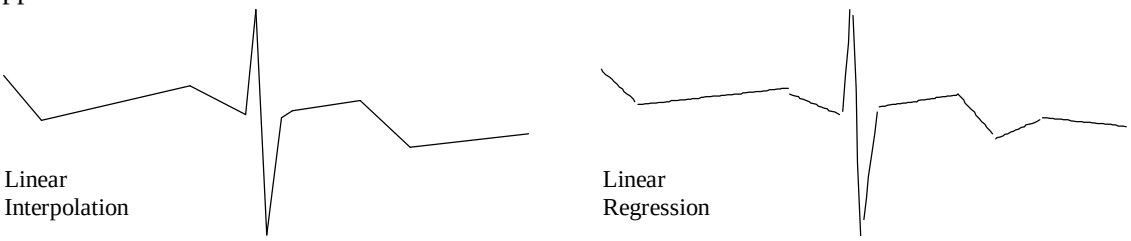

Figure 2: Two 10-segment approximations of electrocardiogram data. The approximation created using linear interpolation has a smooth aesthetically appealing appearance because all the endpoints of the segments are aligned. Linear regression, in contrast, produces a slightly disjointed appearance but a tighter approximation in terms of residual error.

In this paper, we deliberately keep our descriptions of algorithms at a high level, so that either technique can be imagined as the approximation technique. In particular, the pseudocode function `create_segment(T)`can be imagined as using interpolation, regression or any other technique.

All segmentation algorithms also need some method to evaluate the quality of fit for a potential segment. A measure commonly used in conjunction with linear regression is the sum of squares, or the residual error. This is calculated by taking all the vertical differences between the best-fit line and the actual data points, squaring them and then summing them together. Another commonly used measure of goodness of fit is the distance between the best fit line and the data point furthest away in the vertical direction (i.e. the L_∞ norm between the line and the data). As before, we have kept our descriptions of the algorithms general enough to encompass any error

measure. In particular, the pseudocode function `calculate_error(T)`can be imagined as using any sum of squares, furthest point, or any other measure.

2.1 The Sliding Window Algorithm.

The Sliding Window algorithm works by anchoring the left point of a potential segment at the first data point of a time series, then attempting to approximate the data to the right with increasing longer segments. At some point *i*, the error for the potential segment is greater than the user-specified threshold, so the subsequence from the anchor to *i*-1 is transformed into a segment. The anchor is moved to location *i*, and the process repeats until the entire time series has been transformed into a piecewise linear approximation. The pseudocode for the algorithm is shown in Table 2.

```
Algorithm Seg_TS = Sliding_Window(T , max_error)
anchor = 1;
while not finished segmenting time series
i = 2;
 while  calculate_error(T[anchor: anchor + i ]) < max_error
   i = i + 1;
 end;
 Seg_TS = concat(Seg_TS, create_segment(T[anchor: anchor + (i-1)]);
 anchor = anchor + i;
end;
```

Table 2: The generic Sliding Window algorithm

The Sliding Window algorithm is attractive because of its great simplicity, intuitiveness and particularly the fact that it is an online algorithm. Several variations and optimizations of the basic algorithm have been proposed. Koski et al. noted that on ECG data it is possible to speed up the algorithm by incrementing the variable *i* by "leaps of length *k*" instead of 1. For $k = 15$ (at 400Hz), the algorithm is 15 times faster with little effect on the output [12].

Depending on the error measure used, there may be other optimizations possible. Vullings et al. noted that since the residual error is monotonically non-decreasing with the addition of more data points, one does not have to test every value of *i* from 2 to the final chosen value [30]. They suggest initially setting *i* to *s*, where *s* is the mean length of the previous segments. If the guess was pessimistic (the measured error is still less than `max_error`) then the algorithm continues to increment *i* as in the classic algorithm. Otherwise they begin to decrement *i* until the measured error is less than `max_error`. This optimization can greatly speed up the algorithm if the mean length of segments is large in relation to the standard deviation of their length. The monotonically non-decreasing property of residual error also allows binary search for the length of the segment. Surprisingly, no one we are aware of has suggested this.

The Sliding Window algorithm can give pathologically poor results under some circumstances. Most researchers have not reported this [25, 31], perhaps because they tested the algorithm on stock market data, and its relative performance is best on noisy data. Shatkay (1995), in contrast, does notice the problem and gives elegant examples and explanations [27]. They consider three variants of the basic algorithm, each designed to be robust to a certain case, but they underline the difficulty of producing a single variant of the algorithm which is robust to arbitrary data sources.

Park et al. (2001) suggested modifying the algorithm to create "*monotonically changing*" segments [21]. That is, all segments consist of data points of the form of $t_1 \leq t_2 \leq \ldots \leq t_n$ or $t_1 \geq t_2 \geq \ldots \geq t_n$. This modification worked well on the smooth synthetic dataset it was demonstrated on.

But on real world datasets with any amount of noise, the approximation is greatly overfragmented.

Variations on the Sliding Window algorithm are particularly popular with the medical community (where it is known as FAN or SAPA), since patient monitoring is inherently an online task [11, 12, 19, 30].

2.2 The Top-Down Algorithm.

The Top-Down algorithm works by considering every possible partitioning of the times series and splitting it at the best location. Both subsections are then tested to see if their approximation error is below some user-specified threshold. If not, the algorithm recursively continues to split the subsequences until all the segments have approximation errors below the threshold. The pseudocode for the algorithm is shown in Table 3.

```
Algorithm Seg_TS = Top_Down(T , max_error)
best_so_far = inf;
for i = 2 to length(T) - 2  // Find best place to split the time series.
improvement_in_approximation = improvement_splitting_here(T,i);
  if improvement_in_approximation < best_so_far
    breakpoint = i;
    best_so_far = improvement_in_approximation;
  end;
end;
                              // Recursively split the left segment if necessary.
if calculate_error(T[1:breakpoint]) > max_error
  Seg_TS = Top_Down(T[1: breakpoint]);
end;
                              // Recursively split the right segment if necessary.
if calculate_error( T[breakpoint + 1:length(T)] ) > max_error
  Seg_TS = Top_Down(T[breakpoint + 1: length(T)]);
end;
```

Table 3: The generic Top-Down algorithm

Variations on the Top-Down algorithm (including the 2-dimensional case) were independently introduced in several fields in the early 1970's. In cartography, it is known as the Douglas-Peucker algorithm [6]; in image processing, it is known as Ramers algorithm [26]. Most researchers in the machine learning/data mining community are introduced to the algorithm in the classic textbook by Duda and Harts, which calls it "Iterative End-Points Fits"[7].

In the data mining community, the algorithm has been used by [18] to support a framework for mining sequence databases at multiple abstraction levels. Shatkay and Zdonik use it (after considering alternatives such as Sliding Windows) to support approximate queries in time series databases [28].

Park et al. introduced a modification where they first perform a scan over the entire dataset marking every peak and valley [22]. These extreme points is used to create an initial segmentation, and the Top-Down algorithm is applied to each of the segments (in case the error on an individual segment was still too high). They then use the segmentation to support a special case of dynamic time warping. This modification worked well on the smooth synthetic dataset it

was demonstrated on. But on real world data sets with any amount of noise, the approximation is greatly overfragmented.

Lavrenko et al. uses the Top-Down algorithm to support the concurrent mining of text and time series [17]. They attempt to discover the influence of news stories on financial markets. Their algorithm contains some interesting modifications including a novel stopping criteria based on the t-test.

Finally Smyth and Ge use the algorithm to produce a representation which can support a Hidden Markov Model approach to both change point detection and pattern matching [8].

2.3 The Bottom-Up Algorithm.

The Bottom-Up algorithm is the natural complement to the Top-Down algorithm. The algorithm begins by creating the finest possible approximation of the time series, so that $n/2$ segments are used to approximate the n-length time series. Next, the cost of merging each pair of adjacent segments is calculated, and the algorithm begins to iteratively merge the lowest cost pair until a stopping criteria is met. When the pair of adjacent segments i and $i+1$ are merged, the algorithm needs to perform some bookkeeping. First, the cost of merging the new segment with its right neighbor must be calculated. In addition, the cost of merging the $i-1$ segments with its new larger neighbor must be recalculated. The pseudocode for the algorithm is shown in Table 4.

```
Algorithm Seg_TS = Bottom_Up(T , max_error)
for i = 1 : 2 : length(T)         // Create initial fine approximation.
 Seg_TS = concat(Seg_TS, create_segment(T[i: i + 1 ]));
end;
for i = 1 : length(Seg_TS) – 1      // Find cost of merging each pair o
 segments.
 merge_cost(i) = calculate_error([merge(Seg_TS(i), Seg_TS(i+1))]);
end;

while min(merge_cost) < max_error             // While not finished.
 index = min(merge_cost);                     // Find "cheapest" pair to merge
 Seg_TS(index) = merge(Seg_TS(index), Seg_TS(index+1))); // Merge them.
 delete(Seg_TS(index+1));                                // Update records.
 merge_cost(index) = calculate_error(merge(Seg_TS(index), Seg_TS(index+1)));
 merge_cost(index-1) = calculate_error(merge(Seg_TS(index-1), Seg_TS(index)));
end;
```

Table 4: The generic Bottom-Up algorithm

Two and three-dimensional analogues of this algorithm are common in the field of computer graphics where they are called *decimation* methods [9]. In data mining, the algorithm has been used extensively by two of the current authors to support a variety of time series data mining tasks [14, 15, 16]. In medicine, the algorithm was used by Hunter and McIntosh to provide the high level representation for their medical pattern matching system [10].

2.4 Feature Comparison of the Major Algorithms

We have deliberately deferred the discussion of the running times of the algorithms until now, when the readers' intuition for the various approaches are more developed. The running time for each approach is data dependent. For that reason, we discuss both a worst-case time that gives an upper bound and a best-case time that gives a lower bound for each approach.

We use the standard notation of $\Omega(f(n))$ for a lower bound, $O(f(n))$ for an upper bound, and $q(f(n))$ for a function that is both a lower and upper bound.

Definitions and Assumptions The number of data points is n, the number of segments we plan to create is K, and thus the average segment length is L = n/K. The actual length of segments created by an algorithm varies and we will refer to the lengths as L_i.

All algorithms, except top-down, perform considerably worse if we allow any of the L_i to become very large (say n/4), so we assume that the algorithms limit each L_i to some multiple cL of the average length. It is trivial to code the algorithms to enforce this, so the time analysis that follows is exact when the algorithm includes this limit.

All algorithms, except top-down, perform considerably worse if we allow any of the L_i to become very large (say n/4), so we assume that the algorithms limit the maximum length L to some multiple of the average length. It is trivial to code the algorithms to enforce this, so the time analysis that follows is exact when the algorithm includes this limit. Empirical results show, however, that the segments generated (with no limit on length) are tightly clustered around the average length, so this limit has little effect in practice.

We assume that for each set S of points, we compute a best segment and compute the error in $q(n)$ time. This reflects the way these algorithms are coded in practice, which is to use a packaged algorithm or function to do linear regression. We note, however, that we believe one can produce asymptotically faster algorithms if one custom codes linear regression (or other best fit algorithms) to reuse computed values so that the computation is done in less than O(n) time in subsequent steps. We leave that as a topic for future work. In what follows, all computations of best segment and error are assumed to be $q(n)$.

Top-Down: The best time for Top-Down occurs if each split occurs at the midpoint of the data. The first iteration computes, for each split point i, the best line for points [1,i] and for points [i+1,n]. This takes $\theta(n)$ for each split point, or $q(n^2)$ total for all split points. The next iteration finds split points for [1, n/2] and for [n/2+1,n]. This gives a recurrence $T(n) = 2T(n/2) + q(n^2)$ where we have T(2) = c, and this solves to $T(n) = \Omega(n^2)$. This is a lower bound because we assumed the data has the best possible split points.

The worst time occurs if the computed split point is always at one side (leaving just 2 points on one side), rather than the middle. The recurrence is $T(n) = T(n-2) + q(n^2)$ We must stop after K iterations, giving a time of $O(n^2K)$.

Sliding Windows: For this algorithm, we compute best segments for larger and larger windows, going from 2 up to at most cL (by the assumption we discussed above). The maximum time to compute a single segment is $\sum_{i=2}^{cL} q(i) = \theta(L^2)$. The number of segments can be as few as n/cL = K/c or as many as K. The time is thus $q(L^2 K)$ or $q(Ln)$. This is both a best case and worst case bound.

Bottom Up: The first iteration computes the segment through each pair of points and the costs of merging adjacent segments. This is easily seen to take O(n) time. In the following iterations, we look up the minimum error pair i and i +1 to merge; merge the pair into a new segment S_{new}; delete from a heap (keeping track of costs is best done with a heap) the costs of merging segments i-1 and i and merging segments i+1 and i+2; compute the costs of merging S_{new} with S_{i-1} and with S_{i-2}; and insert these costs into our heap of costs. The time to look up the best cost is $q(1)$ and the time to add and delete costs from the heap is O(log$n$). (The time to construct the heap is O(n).)

In the best case, the merged segments always have about equal length, and the final segments have length L. The time to merge a set of length 2 segments, which will end up being one length L segment, into half as many segments is $q(L)$ (for the time to compute the best segment for every pair of merged segments), not counting heap operations. Each iteration takes the same time repeating q (log L) times gives a segment of size L.

The number of times we produce length L segments is K, so the total time is $\Omega(K\ L \log L) = \Omega(n \log n/K)$. The heap operations may take as much as $O(n \log n)$. For a lower bound we have proven just $\Omega(n \log n/K)$.

In the worst case, the merges always involve a short and long segment, and the final segments are mostly of length cL. The time to compute the cost of merging a length 2 segment with a length *i* segment is $q(i)$, and the time to reach a length cL segment is $\sum_{i=2}^{cL} q(i) = q(L^2)$. There are at most n/cL such segments to compute, so the time is $n/cL * q(L^2) = O(Ln)$. (Time for heap operations is inconsequential.)

This complexity study is summarized in Table 5.

Algorithm	User can specify[1]	Online	Complexity	Used by[2]
Top-Down	E, ME, K	No	$O(n^2K)$	6, 7, 8, 18, 22, 17
Bottom-Up	E, ME, K	No	$O(Ln)$	10, 14, 15 , 16
Sliding Window	E	Yes	$O(Ln)$	11, 12, 19, 25, 30, 31, 27

Table 5: A feature summary for the 3 major algorithms.[1]KEY: E → Maximum error for a given segment, ME → Maximum error for a given segment for entire time series, K → Number of segments. [2]Possibly with modifications and/or extensions

In addition to the time complexity, there are other features a practitioner might consider when choosing an algorithm. First, there is the question of whether the algorithm is online or batch. Secondly, there is the question of how the user can specify the quality of desired approximation. With trivial modifications the Bottom-Up algorithm allows the user to specify the desired value of K, the maximum error per segment, or total error of the approximation. A (non-recursive) implementation of Top-Down can also be made to support all three options. However Sliding Window only allows the maximum error per segment to be specified.

3. Empirical Comparison of the Major Segmentation Algorithms

In this section, we will provide an extensive empirical comparison of the three major algorithms. It is possible to create artificial datasets that allow one of the algorithms to achieve zero error (by any measure), but forces the other two approaches to produce arbitrarily poor approximations. In contrast, testing on purely random data forces the all algorithms to produce essentially the same results. To overcome the potential for biased results, we tested the algorithms on a very diverse collection of datasets. These datasets where chosen to represent the extremes along the

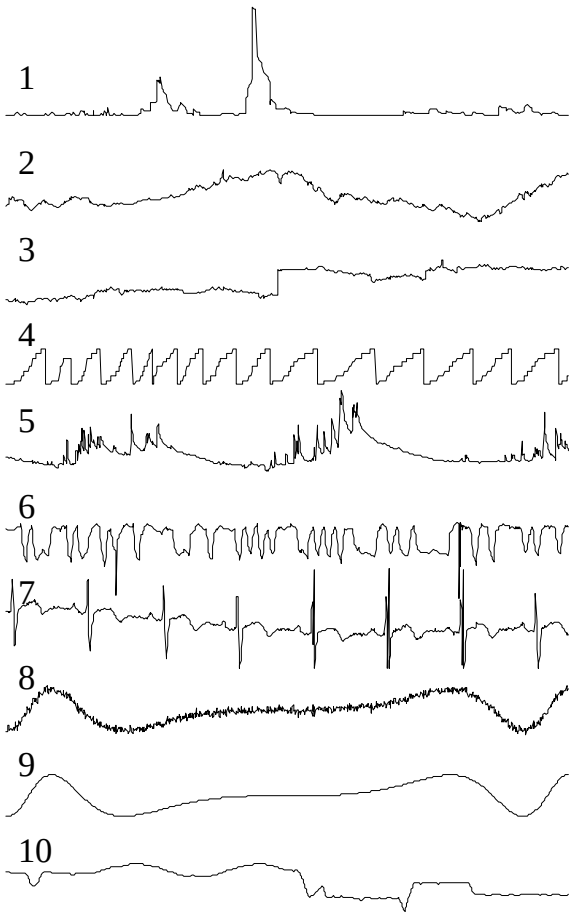

Figure 3: The ten datasets used in the experiments. 1) Radio Waves. 2) Exchange Rates. 3) Tickwise II. 4) Tickwise I. 5) Water

Level. 6) Manufacturing. 7) ECG. 8) Noisy Sine Cubed. 9) Sine Cube. 10) Space Shuttle

following dimensions, stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric, etc. In addition, the data sets represent the diverse areas in which data miners apply their algorithms, including finance, medicine, manufacturing and science. Figure 3 illustrates the 10 datasets used in the experiments.

3.1 Experimental Methodology

For simplicity and brevity, we only include the linear regression versions of the algorithms in our study. Since linear regression minimizes the sum of squares error, it also minimizes the Euclidean distance (the Euclidean distance is just the square root of the sum of squares). Euclidean distance, or some measure derived from it, is by far the most common metric used in data mining of time series [1, 2, 4, 5, 13, 14, 15, 16, 25, 31]. The linear interpolation versions of the algorithms, by definition, will always have a greater sum of squares error.

We immediately encounter a problem when attempting to compare the algorithms. We cannot compare them for fixed values of K, since Sliding Windows does not allow one to specify the number of segments. Instead we give each of the algorithms a fixed `max_error` and measure the total error of the entire piecewise approximation.

The performance of the algorithms depends on the value of `max_error`. As `max_error` goes to zero all the algorithms have the same performance, since they would produce *n*/2 segments with no error. At the opposite end, as `max_error` becomes very large, the algorithms once again will all have the same performance, since they all simply approximate T with a single best-fit line. Instead, we must test the relative performance for some reasonable value of `max_error`, a value that achieves a good trade off between compression and fidelity. Because this "reasonable value" is subjective and dependent on the data mining application and the data itself, we did the following. We chose what we considered a "reasonable value" of `max_error` for each dataset, then we bracketed it with 6 values separated by powers of two. The lowest of these values tends to produce an over-fragmented approximation, and the highest tends to produce a very coarse approximation. So in general, the performance in the mid-range of the 6 values should be consider most important. Figure 4 illustrates this idea.

Since we are only interested in the relative performance of the algorithms, for each setting of `max_error` on each data set, we normalized the performance of the 3 algorithms

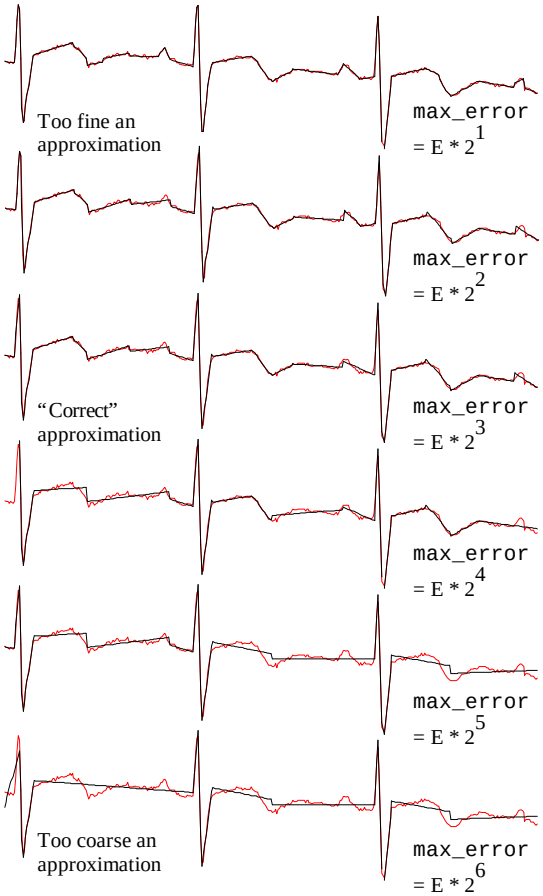

Figure 4: We are most interested in comparing the segmentation algorithms at the setting of the user-defined threshold `max_error` that produces an intuitively correct level of approximation. Since this setting is subjective, we chose a value for `E`, such that `max_error = E*2`^{i} (i = 1 to 6), brackets the range of reasonable approximations.

by dividing by the error of the worst performing approach.

3.2 Experimental Results

The experimental results are summarized in Figure 5. The most obvious result is the generally poor quality of the Sliding Windows algorithm. With a few exceptions, it is the worse performing algorithm, usually by a large amount.

Comparing the results for Sine cubed and Noisy Sine supports our conjecture that the noisier a dataset, the less difference one can expect between algorithms. This suggests that one should exercise caution in attempting to generalize the performance of an algorithm that has only been demonstrated on a single noisy dataset [25, 31].

Top-Down does occasionally beat Bottom-Up, but only by small amount. On the other hand Bottom-Up often significantly out performs Top-Down, especially on the ECG, Manufacturing and Water Level data sets.

Figure 5: A comparison of the three major times series segmentation algorithms, on ten diverse datasets, over a range in parameters. Each experimental result (ie. a triplet of histogram bars) is normalized by dividing by the performance of the worst algorithm on that experiment.

4. A New Approach

Given the noted shortcomings of the major segmentation algorithms, we investigated alternative techniques. The main problem with the Sliding Windows algorithm is its inability to look ahead, lacking the global view of its offline (batch) counterparts. The Bottom-Up and the Top-Down approaches produce better results, but are offline and require the scanning of the entire data set. This is impractical or may even be unfeasible in a data-mining context, where the data are in the order of terabytes or arrive in continuous streams. We therefore introduce a novel approach in which we capture the online nature of Sliding Windows and yet retain the superiority of Bottom-Up. We call our new algorithm SWAB (**S**liding **W**indow **a**nd **B**ottom-up).

4.1 The SWAB segmentation algorithm

The SWAB algorithm keeps a buffer of size *w*. The buffer size should initially be chosen so that there is enough data to create about 5 or 6 segments. Bottom-Up is applied to the data in the buffer and the leftmost segment is reported. The data corresponding to the reported segment is removed from the buffer and more datapoints are read in. The number of datapoints read in depends on the structure of the incoming data. This process is performed by the `Best_Line` function, which is basically just classic Sliding Windows. These points are incorporated into the buffer and Bottom-Up is applied again. This process of applying Bottom-Up to the buffer, reporting the leftmost segment, and reading in the next "best fit" subsequence is repeated as long as data arrives (potentially forever).

The intuition behind the algorithm is this. The `Best_Line` function finds data corresponding to a single segment using the (relatively poor) Sliding Windows and gives it to the buffer. As the data moves through the buffer the (relatively good) Bottom-Up algorithm is given a chance to refine the segmentation, because it has a "semi-global" view of the data. By the time the data is ejected from the buffer, the segmentation breakpoints are usually the same as the ones the batch version of Bottom-Up would have chosen. The pseudocode for the algorithm is shown in Table 6.

Using the buffer allows us to gain a "semi-global" view of the data set for Bottom-Up. However, it important to impose upper and lower bounds on the size of the window. A buffer that

```
Algorithm Seg_TS = SWAB(max_error, seg_num)  // seg_num is integer 5 or 6

read in w number of data points // Enough to approximate seg_num of segments.
lower_bound = w / 2;
upper_bound = 2 * w;

while data at input
   T = Bottom_Up(w, max_error)       // Call the classic Bottom-Up algorithm
   Seg_TS = CONCAT(SEG_TS, T(1));
                                     // Sliding window to the right.
   w = TAKEOUT(w, w');               // Deletes w' points in T(1) from w.
   if data at input                  // Add w" points from BEST_LINE() to w.
      w = CONCAT(w, BEST_LINE(max_error));
      {check upper and lower bound, adjustment if necessary}
   else                             // flush approximated segments from buffer.
      Seg_TS = CONCAT(SEG_TS, (T – T(1)))
   end;
end;

Function S = BEST_LINE(max_error)       //returns S points to approximate
while error ≤ max_error                 // next potential segment.
   read in one additional data point, d, into S
   S = CONCAT(S, d);
   error = approx_segment(S);
end while;
return S;
```

Table 6: The SWAB (Sliding Window and Bottom-up) algorithm

is allowed to grow arbitrarily large will revert our algorithm to pure Bottom-Up, but a small buffer will deteriorate it to Sliding Windows, where excessive fragmentation may occur. In our algorithm, we used an upper (and lower) bound of twice (and half) of the initial buffer.

Our algorithm can be seen as operating on a continuum between the two extremes of Sliding Windows and Bottom-Up. The surprising result (demonstrated below) is that by allowing the buffer to contain just 5 or 6 times the data normally contained by is a single segment, the algorithm produces essentially the same results as Bottom-Up, yet is able to process a never-ending stream of data. Our new algorithm requires only a small, constant amount of memory, and the time complexity is a small constant factor worse than that of the standard Bottom-Up algorithm.

4.2 Experimental Validation

We repeated the experiments in Section 3, this time comparing the new algorithm with pure (batch) Bottom-Up and classic Sliding Windows. The result, summarized in Figure 6, is that the new algorithm produces results that are essentiality identical to Bottom-Up.

Figure 6: A comparison of the SWAB algorithm with pure (batch) Bottom-Up and classic Sliding Windows, on ten diverse datasets, over a range in parameters. Each experimental result (ie. a triplet

of histogram bars) is normalized by dividing by the performance of the worst algorithm on that experiment.

5. Conclusions and Directions for Future Work

We have carried out the first extensive review and empirical comparison of time series segmentation algorithms from a data mining perspective. We have shown the most popular approach, Sliding Windows, generally produces very poor results, and that while the second most popular approach, Top-Down, can produce reasonable results, it does not scale well. In contrast, the least well known, Bottom-Up, approach produces excellent results and scales linearly with the size of the dataset.

In addition, we have introduced SWAB, a new online algorithm, which scales linearly with the size of the dataset, requires only constant space and produces high quality approximations of the data.

We plan to extend this work in several directions.

- The performance of Bottom-Up is particularly surprising given that it explores a smaller space of representations. Because the initialization phase of the algorithm begins with all line segments having length two, all merged segments will also have even lengths. In contrast the two other algorithms allow segments to have odd or even lengths. It would be interesting to see if removing this limitation of Bottom-Up can improve its performance further.
- For simplicity and brevity, we have assumed that the inner loop of the SWAB algorithm simply invokes the Bottom-Up algorithm each time. This clearly results in some computation redundancy. We believe we may be able to reuse calculations from previous invocations of Bottom-Up, thus achieving speedup.

Reproducible Results Statement In the interests of competitive scientific inquiry, all datasets and code used in this work are available, together with a spreadsheet detailing the original unnormalized results, by emailing the first author.

References

[1] Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. *Proceedings of the 4th Conference on Foundations of Data Organization and Algorithms.*

[2] Agrawal, R., Lin, K. I., Sawhney, H. S., & Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in times-series databases. *Proceedings of 21th International Conference on Very Large Data Bases.* pp 490-50.

[3] Agrawal, R., Psaila, G., Wimmers, E. L., & Zait, M. (1995). Querying shapes of histories. *Proceedings of the 21st International Conference on Very Large Databases.*

[4] Chan, K. & Fu, W. (1999). Efficient time series matching by wavelets. *Proceedings of the 15th IEEE International Conference on Data Engineering*.

[5] Das, G., Lin, K. Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. *Proceedings of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 16-22.

[6] Douglas, D. H. & Peucker, T. K.(1973). Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Canadian Cartographer*, Vol. 10, No. 2, December. pp. 112-122.

[7] Duda, R. O. and Hart, P. E. 1973. Pattern Classification and Scene Analysis. Wiley, New York.

[8] Ge, X. & Smyth P. (2001). Segmental Semi-Markov Models for Endpoint Detection in Plasma Etching. To appear in *IEEE Transactions on Semiconductor Engineering*.

[9] Heckbert, P. S. & Garland, M. (1997). Survey of polygonal surface simplification algorithms, Multiresolution Surface Modeling Course. *Proceedings of the 24th International Conference on Computer Graphics and Interactive Techniques.*

[10] Hunter, J. & McIntosh, N. (1999). Knowledge-based event detection in complex time series data. *Artificial Intelligence in Medicine*. pp. 271-280. Springer.

[11] Ishijima, M., et al. (1983). Scan-Along Polygonal Approximation for Data Compression of Electrocardiograms. *IEEE Transactions on Biomedical Engineering.* BME-30(11):723-729.

[12] Koski, A., Juhola, M. & Meriste, M. (1995). Syntactic Recognition of ECG Signals By Attributed Finite Automata. *Pattern Recognition*, 28 (12), pp. 1927-1940.

[13] Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra (2000). Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems.*

[14] Keogh, E. & Pazzani, M. (1999). Relevance feedback retrieval of time series data. *Proceedings of the 22th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval.*

[15] Keogh, E., & Pazzani, M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. *Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 239-241, AAAI Press.

[16] Keogh, E., & Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. *Proceedings of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 24-20.

[17] Lavrenko, V., Schmill, M., Lawrie, D., Ogilvie, P., Jensen, D., & Allan, J. (2000). Mining of Concurent Text and Time Series. *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining.* pp. 37-44.

[18] Li, C,. Yu, P. & Castelli V.(1998). MALM: A framework for mining sequence database at multiple abstraction levels. *Proceedings of the 9th International Conference on Information and Knowledge Management.* pp 267-272.

[19] McKee, J.J., Evans, N.E., & Owens, F.J. (1994). Efficient implementation of the Fan/SAPA-2 algorithm using fixed point arithmetic. *Automedica*. Vol. 16, pp 109-117.

[20] Osaki, R., Shimada, M., & Uehara, K. (1999). Extraction of Primitive Motion for Human Motion Recognition. *The 2nd International Conference on Discovery Science*. pp.351-352.

[21] Park, S., Kim, S. W., & Chu, W. W. (2001). Segment-Based Approach for Subsequence Searches in Sequence Databases, To appear in *Proceedings of the 16th ACM Symposium on Applied Computing.*

[22] Park, S. & Lee, D., & Chu, W. W. (1999). Fast Retrieval of Similar Subsequences in Long Sequence Databases*", Proceedings of the 3rd IEEE Knowledge and Data Engineering Exchange Workshop*.

[23] Pavlidis, T. (1976). Waveform segmentation through functional approximation. *IEEE Transactions on Computers*.

[24] Perng, C., Wang, H., Zhang, S., & Parker, S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. *Proceedings of 16th International Conference on Data Engineering.*

[25] Qu, Y., Wang, C. & Wang, S. (1998). Supporting fast search in time series for movement patterns in multiples scales. *Proceedings of the 7th International Conference on Information and Knowledge Management*.

[26] Ramer, U. (1972). An iterative procedure for the polygonal approximation of planar curves. *Computer Graphics and Image Processing*. 1: pp. 244-256.

[27] Shatkay, H. (1995). Approximate Queries and Representations for Large Data Sequences. *Technical Report cs-95-03*, Department of Computer Science, Brown University.

[28] Shatkay, H., & Zdonik, S. (1996). Approximate queries and representations for large data sequences. *Proceedings of the 12th IEEE International Conference on Data Engineering*. pp 546-553.

[29] Sugiura, N. & Ogden, R. T. (1994). Testing Change-points with Linear Trend *Communications in Statistics B: Simulation and Computation*. 23: 287-322.

[30] Vullings, H.J.L.M., Verhaegen, M.H.G. & Verbruggen H.B. (1997). ECG Segmentation Using Time-Warping. *Proceedings of the 2nd International Symposium on Intelligent Data Analysis*.

[31] Wang, C. & Wang, S. (2000). Supporting content-based searches on time Series via approximation. *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*.