A Better Alternative to Piecewise Linear Time Series Segmentation[*]

Daniel Lemire[†]

Abstract

Time series are difficult to monitor, summarize and predict. Segmentation organizes time series into few intervals having uniform characteristics (flatness, linearity, modality, monotonicity and so on). For scalability, we require fast linear time algorithms. The popular piecewise linear model can determine where the data goes up or down and at what rate. Unfortunately, when the data does not follow a linear model, the computation of the local slope creates overfitting. We propose an adaptive time series model where the polynomial degree of each interval vary (constant, linear and so on). Given a number of regressors, the cost of each interval is its polynomial degree: constant intervals cost 1 regressor, linear intervals cost 2 regressors, and so on. Our goal is to minimize the Euclidean (l_2) error for a given model complexity. Experimentally, we investigate the model where intervals can be either constant or linear. Over synthetic random walks, historical stock market prices, and electrocardiograms, the adaptive model provides a more accurate segmentation than the piecewise linear model without increasing the cross-validation error or the running time, while providing a richer vocabulary to applications. Implementation issues, such as numerical stability and real-world performance, are discussed.

1 Introduction

Time series are ubiquitous in finance, engineering, and science. They are an application area of growing importance in database research [2]. Inexpensive sensors can record data points at 5 kHz or more, generating one million samples every three minutes. The primary purpose of time series segmentation is dimensionality reduction. It is used to find frequent patterns [9] or classify time series [12]. Segmentation points divide the time axis into intervals behaving approximately according to a simple model. Recent work on segmentation used quasi-constant or quasi-linear intervals [10], quasi-unimodal intervals [8], step, ramp or impulse [6], or quasi-monotonic intervals [4, 5]. A good time series segmentation algorithm must

- be fast (ideally run in linear time with low overhead);
- provide the necessary vocabulary such as flat, increasing at rate x, decreasing at rate x, $\ldots$;
- be accurate (good model fit and cross-validation).

Typically, in a time series segmentation, a single model is applied to all intervals. For example, all intervals are assumed to behave in a quasi-constant or quasi-linear manner. However, mixing different models and, in particular, constant and linear intervals can have two immediate benefits. Firstly, some applications need a qualitative description of each interval [24] indicated by change of model: is the temperature rising, dropping or is it stable? In an ECG, we need to identify the flat interval between each cardiac pulses. Secondly, as we will show, it can reduce the fit error without increasing the cross-validation error. Intuitively, a piecewise model tells when the data is increasing, and at what rate, and *vice versa*. While most time series have clearly identifiable linear trends some of the time, this is not true over all time intervals. Therefore, the piecewise linear model locally overfits the data by computing meaningless slopes.

Global overfitting has been addressed by limiting the number of regressors [23], but this carries the implicit assumption that time series are somewhat stationary [19]. Some frameworks [24] qualify the intervals where the slope is not significant as being "constant" while others look for constant intervals within upward or downward intervals [4].

Piecewise linear segmentation is ubiquitous and was one of the first applications of dynamic programming [3]. We argue that many applications can benefit from replacing it with a mixed model (piecewise linear and constant). When identifying constant intervals *a posteriori* from a piecewise linear model, we risk misidentifying some patterns including "stair cases" or "steps". A contribution of this paper is experimental evidence that we reduce fit without sacrificing the cross-validation error or running time for a given model complexity by using adaptive algorithms where some intervals have a constant model whereas others have a linear model. The new heuristic we propose is neither more difficult to implement nor more expensive computationally. Our experiments include white noise and random walks as well as ECGs and stock market prices. We also compare against the dynamic programming (optimal) solution which we show can be computed in time $O(n^2k)$.

Performance-wise, common heuristics (piecewise linear or constant) have been reported to require quadratic

[*]Supported by NSERC grant 261437. The full version of this paper is available as a technical report [15].

[†]Université du Québec à Montréal (UQAM)

time [10]. We want fast linear time algorithms. When the number of desired segments is small, top-down heuristics might be the only sensible option. We show that if we allow the one-time linear time computation of a buffer, adaptive (and non-adaptive) top-down heuristics run in linear time ($O(n)$).

Data mining requires high scalability over very large data sets. Implementation issues, including numerical stability, must be considered. In this paper, we present algorithms that can process millions of data points in minutes, not hours.

2 Related Work

The original dynamic programming solution proposed by Bellman [3] ran in time $O(n^3 k)$, and while it is known that a $O(n^2 k)$-time implementation is possible for piecewise constant segmentation [22], we will show in this paper that the same reduced complexity applies for piecewise linear and mixed models segmentations as well. Except for Pednault who mixed linear and quadratic segments [20], we know of no other attempt to segment time series using polynomials of variable degrees in the data mining and knowledge discovery literature.

3 Complexity Model

Our complexity model is purposely simple. The model complexity of a segmentation is the sum of the number of regressors over each interval: a constant interval has a cost of 1, a linear interval a cost of 2 and so on. In other words, a linear interval is as complex as two constant intervals. Conceptually, regressors are real numbers whereas all other parameters describing the model only require a small number of bits.

In our implementation, each regressor counted uses 64 bits (“double” data type in modern C or Java). There are two types of hidden parameters which we discard: the width or location of the intervals and the number of regressors per interval. The number of regressors per interval is only a few bits and is not significant in all cases. The width of the intervals in number of data points can be represented using $\kappa \lceil \log m \rceil$ bits where m is the maximum length of a interval and κ is the number of intervals: in the experimental cases we considered, $\lceil \log m \rceil \leq 8$ which is small compared to 64, the number of bits used to store each regressor counted. We should also consider that slopes typically need to be stored using more accuracy (bits) than constant values. This last consideration is not merely theoretical since a 32 bits implementation of our algorithms is possible for the piecewise constant model whereas, in practice, we require 64 bits for the piecewise linear model (see proposition 5.1 and discussion that follows). Experimentally, the piecewise linear model can significantly outperform (by $\approx 50\%$) the piecewise constant model in accuracy and vice versa. For the rest of this paper, we take the fairness of our complexity model as an axiom.

4 Time Series, Segmentation Error and Leave-One-Out

Time series are sequences of data points $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ where the x values, the “time” values, are sorted: $x_i > x_{i-1}$. In this paper, both the x and y values are real numbers. We define a segmentation as a sorted set of segmentation indexes $z_0, \ldots, z_\kappa$ such that $z_0 = 0$ and $z_\kappa = n$. The segmentation points divide the time series into intervals $S_1, \ldots, S_\kappa$ defined by the segmentation indexes as $S_j = \{(x_i, y_i) | z_{j-1} \leq i < z_j\}$. Additionally, each interval $S_1, \ldots, S_\kappa$ has a model (constant, linear, upward monotonic, and so on).

In this paper, the segmentation error is computed from $\sum_{j=1}^{\kappa} Q(S_j)$ where the function Q is the square of the l_2 regression error. Formally, $Q(S_j) = \min_p \sum_{r=z_{j-1}}^{z_j - 1} (p(x_r) - y_r)^2$ where the minimum is over the polynomials p of a given degree. For example, if the interval S_j is said to be constant, then $Q(S_j) = \sum_{z_j \leq l \leq z_{j+1}} (y_l - \bar{y})^2$ where $\bar{y}$ is the average, $\bar{y} = \sum_{z_{j-1} \leq l < z_j} \frac{y_l}{z_{j+1} - z_j}$. Similarly, if the interval has a linear model, then $p(x)$ is chosen to be the linear polynomial $p(x) = ax + b$ where a and b are found by regression. The segmentation error can be generalized to other norms, such as the maximum-error (l_∞) norm [4, 16] by replacing the $\sum$ operators by $\max$ operators.

Beside the data fit error, another interesting form of error is obtained by cross-validation: divide your data points into two sets (training and test), and measure how well your model, as fitted over the training set, predicts the test set. We predict a missing data point (x_i, y_i) by first determining the interval $[z_{j-1}, z_j)$ corresponding to the data point ($x_{z_{j_1}} < x_i < x_{z_j}$) and then we compute $p(x_i)$ where p is the regression polynomial over S_j. The error is $|p(x_i) - y_i|$. We opt for the leave-one-out cross-validation where the test set is a single data point and the training set is the remainder. We repeat the cross-validation over all possible missing data points, except for the first and last data point in the time series, and compute the mean square error. If computing the segmentation takes linear time, then computing the leave-one-out error in this manner takes quadratic time, which is prohibitive for long time series.

Naturally, beyond the cross-validation and fit errors, a segmentation should provide the models required by the application. A rule-based system might require to know where the data does not significantly increase or decrease and if flat intervals have not been labelled, such queries are hard to support elegantly.

5 Polynomial Fitting in Constant Time

The naive fit error computation over a given interval takes linear time $O(n)$: solve for the polynomial p and then compute $\sum_i (y_i - p(x_i))^2$. This has lead other authors to conclude that top-down segmentation algorithm such as Douglas-Peucker's require quadratic time [10] while we will show they can run in linear time. To segment a time series into quasi-polynomial intervals in optimal time, we must compute fit errors in constant time ($O(1)$).

PROPOSITION 5.1. *Given a time series $\{(x_i, y_i)\}_{i=1,\ldots,n}$, if we allow the one-time $O(n)$ computation of a prefix buffer, finding the best polynomial fitting the data over the interval $[x_p, x_q]$ is $O(1)$. This is true whether we use the Euclidean distance (l_2) or higher order norms (l_r for $\infty > r > 2$).*

Proof. Computing the best polynomial fitting some data points over a specific range and computing the corresponding fit error in constant time is equivalent to computing range sums of the form $\sum_{i=p}^{q} x_i^i y_i^l$ in constant time for $0 \leq i, l \leq 2N$. To do so, simply compute once all prefix sums $P_q^{j,l} = \sum_{i=0}^{q} x_i^j y_i^l$ and then use their subtractions to compute range queries $\sum_{i=p}^{q} x_i^j y_i^l = P_q^{j,l} - P_{p-1}^{j,l}$.

Prefix sums speed up the computation of the range sums (making them constant time) at the expense of update time and storage: if one of the data point changes, we may have to recompute the entire prefix sum. More scalable algorithms are possible if the time series are dynamic [14]. Computing the needed prefix sums is only done once in linear time and requires $(N^2 + N + 1)n$ units of storage ($6n$ units when $N = 2$). For most practical purposes, we argue that we will soon have infinite storage so that trading storage for speed is a good choice. It is also possible to use less storage [17].

When using floating point values, the prefix sum approach causes a loss in numerical accuracy which becomes significant if x or y values grow large and $N > 2$. When $N = 1$ (constant polynomials), 32 bits floating point numbers are sufficient, but for $N \geq 2$, 64 bits is required. In this paper, we are not interested in higher order polynomials and choosing $N = 2$ is sufficient.

6 Optimal Adaptive Segmentation

An algorithm is optimal, if it can find a segmentation with minimal error given a model complexity k. Since we can compute best fit error in constant time for arbitrary polynomials, a dynamic programming algorithm computes the optimal adaptive segmentation in time $O(n^2Nk)$ where N is the upper bound on the polynomial degrees. Unfortunately, if $N \geq 2$, this result does not hold in practice with 32 bits floating point numbers.

We improve over the classical approach [3] because we allow the polynomial degree of each interval to vary. In the tradition of dynamic programming [13, pages 261–265], in a first stage, we compute the optimal cost matrix (R): $R_{r,p}$ is the minimal segmentation cost of the time interval $[x_0, x_p)$ using a model complexity of r. If $E(p, q, d)$ is the fit error of a polynomial of degree d over the time interval $[x_p, x_q)$, computable in time $O(1)$ by proposition 5.1, then

$$R_{r,q} = \min_{0 \leq p \leq q, 0 \leq d < N} R_{r-1-d,p} + E(p, q, d)$$

with the convention that $R_{r-1-d,p}$ is infinite when $r - 1 - d < 0$ except for $R_{-1,0} = 0$. Because computing $R_{r,q}$ only requires knowledge of the prior rows, $R_{r',\cdot}$ for $r' < r$, we can compute R row-by-row starting with the first row (see Algorithm 1). Once we have computed the $r \times n + 1$ matrix, we reconstruct the optimal solution with a simple $O(k)$ algorithm (see Algorithm 2) using matrices D and P storing respectively the best segmentation points and the best degrees.

Algorithm 1 First part of dynamic programming algorithm for optimal adaptive segmentation of time series into intervals having degree $0, \ldots, N - 1$.

```
1: INPUT: Time Series $(x_i, y_i)$ of length $n$
2: INPUT: Model Complexity $k$ and maximum degree $N$
   ($N = 2 \Rightarrow$ constant and linear)
3: INPUT: Function $E(p, q, d)$ computing fit error with
   poly. of degree $d$ in range $[x_p, x_q)$ (constant time)
4: $R, D, P \leftarrow k \times n + 1$ matrices (initialized at 0)
5: for $r \in \{0, \ldots, k - 1\}$ do
6:     $\{r$ scans the rows of the matrices$\}$
7:     for $q \in \{0, \ldots, n\}$ do
8:         $\{q$ scans the columns of the matrices$\}$
9:         Find a minimum of $R_{r-1-d,p} + E(p, q, d)$ and store
           its value in $R_{r,q}$, and the corresponding $d, p$ tuple
           in $D_{r,q}, P_{r,q}$ for $0 \leq d \leq \min(r + 1, N)$ and
           $0 \leq p \leq q + 1$ with the convention that $R$ is $\infty$
           on negative rows except for $R_{-1,0} = 0$.
10: RETURN cost matrix $R$, degree matrix $D$, segmentation points matrix $P$
```

Algorithm 2 Second part of dynamic programming algorithm for optimal adaptive segmentation.

```
1: INPUT: $k \times n + 1$ matrices $R$, $D$, $P$ from dynamic
   programming algo.
2: $x \leftarrow n$
3: $s \leftarrow$ empty list
4: while $r \geq 0$ do
5:     $p \leftarrow P_{r,x}$
6:     $d \leftarrow D_{r,x}$
7:     $r \leftarrow r - d + 1$
8:     append interval from $p$ to $x$ having degree $d$ to $s$
9:     $x \leftarrow p$
10: RETURN optimal segmentation $s$
```

7 Piecewise Linear or Constant Top-Down Heuristics

Computing optimal segmentations with dynamic programming is $\Omega(n^2)$ which is not practical when the size of the time series is large. Many efficient online approximate algorithms are based on greedy strategies. Unfortunately, for small model complexities, popular heuristics run in quadratic time ($O(n^2)$) [10]. Nevertheless, when the desired model complexity is small, a particularly effective heuristic is the top-down segmentation which proceeds as follows: starting with a simple segmentation, we further segment the worst interval, and so on, until we exhaust the budget. Keogh *et al.* [10] state that this algorithm has been independently discovered in the seventies and is known by several name: Douglas-Peucker algorithm, Ramers algorithm, or Iterative End-Points Fits. In theory, Algorithm 3 computes the top-down segmentation, using polynomial regression of any degree, in time $O(kn)$ where k is the model complexity, by using fit error computation in constant time. numbers. The piecewise constant ($d = 0$) and piecewise linear ($d = 1$) cases are referred to as the "top-down constant" and "top-down linear" heuristics respectively.

Algorithm 3 Top-Down Heuristic.

```
INPUT: Time Series $(x_i, y_i)$ of length $n$
INPUT: Polynomial degree $d$ ($d = 0$, $d = 1$, etc.) and model complexity $k$
INPUT: Function $E(p, q)$ computing fit error with poly. in range $[x_p, x_q)$
$S$ empty list
$S \leftarrow (0, n, E(0, n))$
$b \leftarrow k - d$
while $b - d \geq 0$ do
    find tuple $(i, j, \epsilon)$ in $S$ with maximum last entry
    find minimum of $E(i, l) + E(l, j)$ for $l = i + 1, \ldots, j$
    remove tuple $(i, j, \epsilon)$ from $S$
    insert tuples $(i, l, E(i, l))$ and $(l, j, E(l, j))$ in $S$
    $b \leftarrow b - d$
$S$ contains the segmentation
```

8 Adaptive Top-Down Segmentation

Our linear time adaptive segmentation heuristic is based on the observation that a linear interval can be replaced by two constant intervals without model complexity increase. After applying the top-down linear heuristic from the previous section (see Algorithm 3), we optimally subdivide each interval once with intervals having fewer regressors (such as constant) but the same total model complexity. The computational complexity is the same, $O((k + 1)n)$. In practice, we first apply the top-down linear heuristic and then we seek to split the linear intervals into two constant intervals.

Because the algorithm only splits an interval if the fit error can be reduced, it is guaranteed not to degrade the fit error. However, improving the fit error is not, in itself, desirable unless we can also ensure we do not increase the cross-validation error.

An alternative strategy is to proceed from the top-down constant heuristic and try to merge constant intervals into linear intervals. We chose not to report our experiments with this alternative since, over our data sets, it gives worse results and is slower than all other heuristics.

Algorithm 4 Adaptive Top-Down Heuristic.

```
INPUT: Time Series $(x_i, y_i)$ of length $n$
INPUT: Bound on Polynomial degree $N$ and model complexity $k$
INPUT: Function $E(p, q, d)$ computing fit error with poly. in range $[x_p, x_q)$
$S$ empty list
$d \leftarrow N - 1$
$S \leftarrow (0, n, d, E(0, n, d))$
$b \leftarrow k - d$
while $b - d \geq 0$ do
    find tuple $(i, j, d, \epsilon)$ in $S$ with maximum last entry
    find minimum of $E(i, l, d) + E(l, j, d)$ for $l = i + 1, \ldots, j$
    remove tuple $(i, j, \epsilon)$ from $S$
    insert tuples $(i, l, d, E(i, l, d))$ and $(l, j, d, E(l, j, d))$ in $S$
    $b \leftarrow b - d$
for tuple $(i, j, q, \epsilon)$ in $S$ do
    find minimum $m$ of $E(i, l, d') + E(l, j, q - d' - 1)$ for $l = i + 1, \ldots, j$ and $0 \leq d' \leq q - 1$
    if $m < \epsilon$ then
        remove tuple $(i, j, q, \epsilon)$ from $S$
        insert tuples $(i, l, d', E(i, l, d'))$ and $(l, j, q - d' - 1, E(l, j, q - d' - 1))$ in $S$
$S$ contains the segmentation
```

9 Implementation and Testing

Using a Linux platform, we implemented our algorithms in C++ using GNU GCC 3.4 and flag "-O2". Intervals are stored in an STL *list* object. Source code is available from the author. Experiments run on a PC with an AMD Athlon 64 (2 GHZ) CPU and enough internal memory so that no disk paging is observed.

Using ECG data and various number of data points, we benchmark the optimal algorithm, using dynamic programming, against the adaptive top-down heuristic: Fig. 1 demonstrates that the quadratic time nature of the dynamic programming solution is quite prevalent ($t \approx n^2/50000$ seconds) making it unusable in all but toy cases, despite a C++ implementation: nearly a full year would be required to opti-

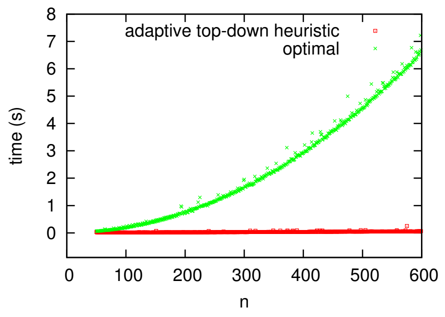

Figure 1: Adaptive Segmentation Timings: Time in seconds versus the number of data points using ECG data. We compare the optimal dynamic programming solution with the top-down heuristic ($k = 20$).

mally segment a time series with 1 million data points! Even if we record only one data point every second for an hour, we still generate 3,600 data points which would require about 4 minutes to segment! Computing the leave-one-out error of a quadratic time segmentation algorithm requires cubic time: to process the numerous time series we chose for this paper, days of processing are required.

We observed empirically that the timings are not sensitive to the data source. The difference in execution time of the various heuristics is negligible (under 15%): our implementation of the adaptive heuristic is not significantly more expensive than the top-down linear heuristic because its additional step, where constant intervals are created out of linear ones, can be efficiently written as a simple sequential scan over the time series.

10 Random Time Series and Segmentation

Intuitively, adaptive algorithms over purely random data are wasteful. To verify this intuition, we generated 10 sequences of Gaussian random noise ($n = 200$): each data point takes on a value according to a normal distribution ($\mu = 0, \sigma = 1$). As expected, the adaptive heuristic shows a slightly worse cross-validation error. However, this is compensated by a slightly better fit error (5%) when compared with the top-down linear heuristic. The dynamic programming solution is 10% more accurate for all three models (adaptive, linear, constant/flat), for twice the running time. The relative errors are not sensitive to the model complexity.

Many chaotic processes such as stock prices are sometimes described as random walk. Unlike white noise, the value of a given data point depends on its neighbors. We generated 10 sequences of Gaussian random walks ($n = 200$): starting at the value 0, each data point takes on the value $y_i = y_{i-1} + N(0, 1)$ where $N(0, 1)$ is a value from a normal distribution ($\mu = 0, \sigma = 1$). The adaptive algorithm improves the leave-one-out error (3%–6%) and the fit error ($\approx 13\%$) over the top-down linear heuristic. Again, the optimal algorithms improve over the heuristics by approximately 10% and the model complexity does not change the relative errors.

11 Stock Market Prices and Segmentation

Creating, searching and identifying stock market patterns is sometimes done using segmentation algorithms [24]. Keogh and Kasetty suggest [11] that stock market data is indistinguishable from random walks. If so, the good results from the previous section should carry over. However, the random walk model has been strongly rejected using variance estimators [18]. Moreover, Sornette [21] claims stock markets are akin to physical systems and can be predicted.

We segmented daily stock prices from dozens of companies [1]. Ignoring stock splits, we pick the first 200 trading days of each stock or index. The model complexity varies $k = 10, 20, 30$ so that the number of intervals can range from 5 to 30. We compute the segmentation error using 3 top-down heuristics: adaptive, linear and constant. As expected, the adaptive heuristic is more accurate than the top-down linear heuristic (the gains are between 4% and 11%). The leave-one-out cross-validation error is improved with the adaptive heuristic when the model complexity is small. Using all of the historical data available, we plot the 3 segmentations for Microsoft stock prices (see Fig. 2). The line is the regression polynomial over each interval and only 150 data points out of 5,029 are shown to avoid clutter.

12 ECGs and Segmentation

We use ECG samples from the MIT-BIH Arrhythmia Database [7]. The signals are recorded at a sampling rate of 360 samples per second with 11 bits resolution. Prior to segmentation, we choose time intervals spanning 300 samples (nearly 1 second) centered around the QRS complex. We select 5 such intervals by a moving window in the files of 6 different patients (“100.dat”, “101.dat”, “102.dat”, “103.dat”, “104.dat”, “105.dat”). The model complexity varies $k = 10, 20, 30$.

With the same model complexity, the adaptive top-down heuristic is better than the linear top-down heuristic (>5%), but more importantly, we reduce the leave-one-out cross-validation error as well for small model complexities.

13 Conclusion and Future Work

We argue that if one requires a multimodel segmentation including flat and linear intervals, it is better to segment accordingly instead of post-processing a piecewise linear segmentation. Mixing drastically different interval models (monotonic and linear) and offering richer, more flexible segmentation models remains an important open problem.

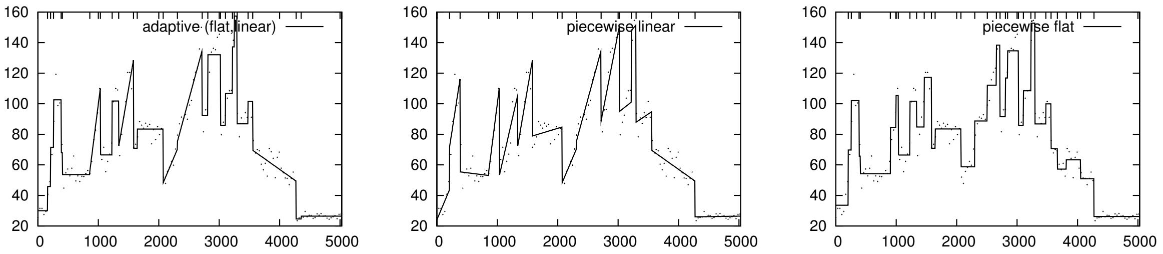

Figure 2: Microsoft historical daily stock prices (close) segmented by the adaptive top-down (top), linear top-down (middle), and constant top-down (bottom) heuristics. **For clarity, only 150 data points out of 5,029 are shown.**

As supporting evidence that mixed models are competitive, we consistently improved the accuracy by 5% and 13% respectively without increasing the cross-validation error over white noise and random walk data. Moreover, whether we consider stock market prices of ECG data, for small model complexity, the adaptive top-down heuristic is noticeably better than the commonly used top-down linear heuristic. The adaptive segmentation heuristic is not significantly harder to implement nor slower than the top-down linear heuristic.

Future work will investigate real-time processing for online applications such as high frequency trading [24] and live patient monitoring.

References

[1] Yahoo! Finance. last accessed June, 2006.

[2] S. Abiteboul, R. Agrawal, et al. The Lowell Database Research Self Assessment. Technical report, Microsoft, 2003.

[3] R. Bellman and R. Roth. Curve fitting by segmented straight lines. *J. Am. Stat. Assoc.*, 64:1079–1084, 1969.

[4] M. Brooks, Y. Yan, and D. Lemire. Scale-based monotonicity analysis in qualitative modelling with flat segments. In *IJCAI'05*, 2005.

[5] W. Fitzgerald, D. Lemire, and M. Brooks. Quasi-monotonic segmentation of state variable behavior for reactive control. In *AAAI'05*, 2005.

[6] D. G. Galati and M. A. Simaan. Automatic decomposition of time series into step, ramp, and impulse primitives. *Pattern Recognition*, 39(11):2166–2174, November 2006.

[7] A. L. Goldberger, L. A. N. Amaral, et al. PhysioBank, PhysioToolkit, and PhysioNet. *Circulation*, 101(23):215–220, 2000.

[8] N. Haiminen and A. Gionis. Unimodal segmentation of sequences. In *ICDM'04*, 2004.

[9] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *KDD'98*, 1998.

[10] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM'01*, pages 289–296, Washington, DC, USA, 2001. IEEE Computer Society.

[11] E. J. Keogh and S. F. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[12] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD'98*, pages 239–243, 1998.

[13] J. Kleinberg and É. Tardos. *Algorithm design*. Pearson/Addison-Wesley, 2006.

[14] D. Lemire. Wavelet-based relative prefix sum methods for range sum queries in data cubes. In *CASCON 2002*. IBM, October 2002.

[15] D. Lemire. A better alternative to piecewise linear time series segmentation. Technical Report cs.DB/0605103, arxiv.org, 2006.

[16] D. Lemire, M. Brooks, and Y. Yan. An optimal linear time algorithm for quasi-monotonic segmentation. In *ICDM'05*, 2005.

[17] D. Lemire and O. Kaser. Hierarchical bin buffering: Online local moments for dynamic external memory arrays. submitted in February 2004.

[18] A. W. Lo and A. C. MacKinlay. Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test. *The Review of Financial Studies*, 1(1):41–66, 1988.

[19] G. Monari and G. Dreyfus. Local Overfitting Control via Leverages. *Neural Comp.*, 14(6):1481–1506, 2002.

[20] E. Pednault. Minimum length encoding and inductive inference. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*. AAAI Press, 1991.

[21] D. Sornette. *Why Stock Markets Crash: Critical Events in Complex Financial Systems*. Princeton University Press, 2002.

[22] E. Terzi and P. Tsaparas. Efficient algorithms for sequence segmentation. In *SDM'06*, 2006.

[23] K. T. Vasko and H. T. T. Toivonen. Estimating the number of segments in time series data using permutation tests. In *ICDM'02*, 2002.

[24] H. Wu, B. Salzberg, and D. Zhang. Online event-driven subsequence matching over financial data streams. In *SIGMOD'04*, pages 23–34, New York, NY, USA, 2004. ACM Press.