A Fast and Robust Method for Pattern Matching in Time Series Databases

Eamonn Keogh

Department of Information and Computer Science
University of California, Irvine
California 92697-3425
eamonn@swim2birds.ics.uci.edu

Abstract

The problem of finding patterns of interest in time series databases (query by content) is an important one, with applications in virtually every field of science. A variety of approaches have been suggested. These approaches are robust to noise, offset translation, and amplitude scaling to varying degrees. However, they are all extremely sensitive to scaling in the time axis (longitudinal scaling). We present a method for similarity search that is robust to scaling in the time axis, in addition to noise, offset translation, and amplitude scaling. The method has been tested on medical, financial, space telemetry and artificial data. Furthermore the method is exceptionally fast, with the predicted 2 to 4 orders of magnitude speedup actually observed. The method uses a piecewise linear representation of the original data. We also introduce a new algorithm which both decides the optimal number of linear segments to use, and produces the actual linear representation.

Introduction

Time series account for a large amount of data stored in databases. A common task with a time series database is to look for an occurrence of a particular pattern within a longer sequence. Such queries have obvious applications in many fields, such as:

- Identifying patterns associated with growth in stock prices.
- Detecting anomalies in an online monitoring system.
- Identifying non-obvious relationships between two time series. For example, a time series recording the sales of ice cream might show peaks in the same places as a time series recording the daily temperature. Such a relationship might be easily detected by simple correlation, but less obvious causal relationships that involve a time lag might be more difficult to discover.
- Hypothesis testing. Here an expert draws a model of a pattern he would expect to find in a database, given his understanding of the domain. Searching for the pattern can help to confirm or deny the original hypothesis, and inspire new ones.

Similarity searching is not a trivial problem. The two primary difficulties are time complexity and defining a similarity measure. The obvious brute force solution to the problem, 'sliding' the shorter query sequence Q against the longer reference sequence R, calculating the error term at each point, is called 'sequential scanning'. Sequential scanning has a time complexity[1] of $O((m-n+1)*n)$ where m is the number of datapoints in R, and n is the number of datapoints in Q. For some of the simple experiments performed for this paper m = 22,627 and n = 1426. Here, sequential scanning requires approximately 30 million operations. For real-world applications we would like to be able to search over much longer time series, ideally in real time.

Sequential scanning typically uses squared error as its similarity measure. This is adequate if an exact match to Q appears in R. However, we do not generally expect to find exact matches to our queries.

We need to allow some degree of fuzziness in our query. While many methods have been proposed which allow some degree of distortion in the y-axis between a query sequence and a potentially matching sub-section of a reference time series, none allows distortion, (i.e. stretching or compressing) in the time axis. The contribution of this paper will primarily be to introduce such a method. In the rest of the paper we shall refer to similarity measures which do *not* allow this elasticity in the time axis, as *lockstep* measures.

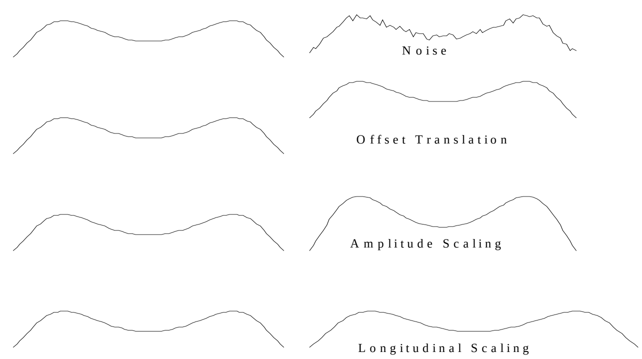


Figure 1: Some of the problems encountered when defining a similarity measure for a time series

Representation

As with most problems in computer science, representation is of key importance. We require a representation that will allow us to define a distance measure that is robust to longitudinal scaling. After examining other possibilities, including Fourier Transforms (Faloutsos et al 1995) and Relational Trees (Shaw and DeFigueiredo 1990, Ehrich and Foith 1976), we decided upon a piecewise liner representation. This representation has many desirable properties, including:

- High rates of data compression. For example, consider Figure 8. The original time series contains 22,627 points. The segmented version of it contains only 38 segments, yet captures most of the essential information.
- Relative insensitivity to noise.
- Intuitiveness and ease of visualization.

[1] Note that the worst case is when $n \approx \sqrt{m}$

The high rate of data compression is of particular importance. As we will see, the computational speedup we achieve will be proportional to the square of compression rate. In addition, it will allow us to search huge time series in main memory without having to swap in and out small sub-sections.

There are many well-known segmentation algorithms in the literature, many of which were pioneered by Pavlidis (1974). It should be emphasized that the distance measures, and search techniques presented in the rest of this paper will work with any segmentation algorithm. For completeness, however, we present a variant of the 'bottom up' algorithm which not only produces a segmentation, but chooses K, the number of segments used to represent the time series.

The decision of how many linear segments to use to approximate a given time series is a very important one. If we choose too small a K, we will lose important features and may fail in our goal of similarity search. On the other hand, if we choose too large a K, we capture all the necessary detail in our time series, but retain redundant information. This is of particular importance because the speedup achieved by the search algorithm presented later in this paper is proportional to the *square* of the compression rate. Approaches in the literature for choosing the correct K range from 'eyeballing' (i.e., manual inspection and choice) to requiring that each segment have its error norm ε less than some threshold (Pavlidis 1973). The problem with the latter approach is that given two time series, A and B, where B is simply A plus noise, it will produce two segmentations, similar in shape, but with the segmentation representing B containing far too many segments. Ideally in this situation we would like the algorithm to produce identical segmentations.

Our segmentation algorithm works in the following manner. We begin by approximating the given time series T (which contains n points), with j linear segments, where $j = \lfloor \frac{n}{3} \rfloor$. At this stage, each segment contains 3 points (we allow the last segment to contain 4 or 5 points if there exists a remainder from $\lfloor \frac{n}{3} \rfloor$). Each segment is the best-fit line through its collection of points, as determined using the classic regression definition $y - \bar{y} = \frac{S_{xy}}{S_x^2}(x - \bar{x})$. Naturally we don't expect each segment to fit the data perfectly. They will contain some amount of residual error for each point they approximate. This residual error, which is the vertical distance from the best-fit line to the data points, we will call $d_1, d_2, \ldots, d_j$. We define the normalized error e_i, for each segment as: $e_i = \frac{\sum_{m=1}^{j} d_m^2}{j}$

Informally, the measure e_i can be considered a measure of how well the i^{th} segment is approximating its share of the data, normalizing for its length. We note that, in general, the e_i's will display considerable variance. We formally capture this variability by defining the Balance of Error for k segments as: $B_k = std(e_1, e_2, \ldots, e_k)$

The segmentation algorithm begins by merging two neighboring segments to produce a new approximation of the time series with j-1 segments. This process is repeated until we have a single line approximation of the original time series (i.e. j = 1). At this point we have an entire family of linear approximations, one for each value of j from 1 to $\lfloor \frac{n}{3} \rfloor$. Note that we have left two crucial details unspecified :

- What is the criterion for selecting which pair of segments to merge in each iteration?
- Which of the approximations should we actually use?

The criterion for selecting which pair of segments to merge is 'Merge the two segments, which give the minimum value of B_k in the next iteration'. The criterion for selecting which approximation to use is 'Select the model for which B_k was minimized'. The intuition behind this method is as follows. It is known that under certain assumptions, the best approximation for a given function using piecewise polynomials requires that the error norm be equal among all segments (Lawson 1964, Pavlidis 1973). Almost none of these assumptions are true for an arbitrary time series, but as we shall see, balance of error is a useful heuristic. In the first iteration of our algorithm, the balance of error is distributed almost randomly. In each subsequent iteration, the algorithm attempts to redress this. This is possible while the number of segments remaining is greater than the true K. When the number of segments remaining equals K, all segments will have almost equal error norms. In the next iteration two of those segments must be merged, resulting in a larger segment which has a very large error norm.

To test this algorithm we needed data sets for which the true value of K is not subjective. We built artificial time series containing 1000 points, but with a highly linear structure, as in Figure 2.a. We then added some Guassian noise to the model, as in Figure 2.b, and attempted to reconstruct the original model as in Figure 2c. Figure 2.d shows B_k plotted against k. B_k is clearly minimized at k = 20, which is the correct number of segments in this example.

If the underlying model is truly linear and we have enough data, we can always recover the true model regardless of how much (symmetrically

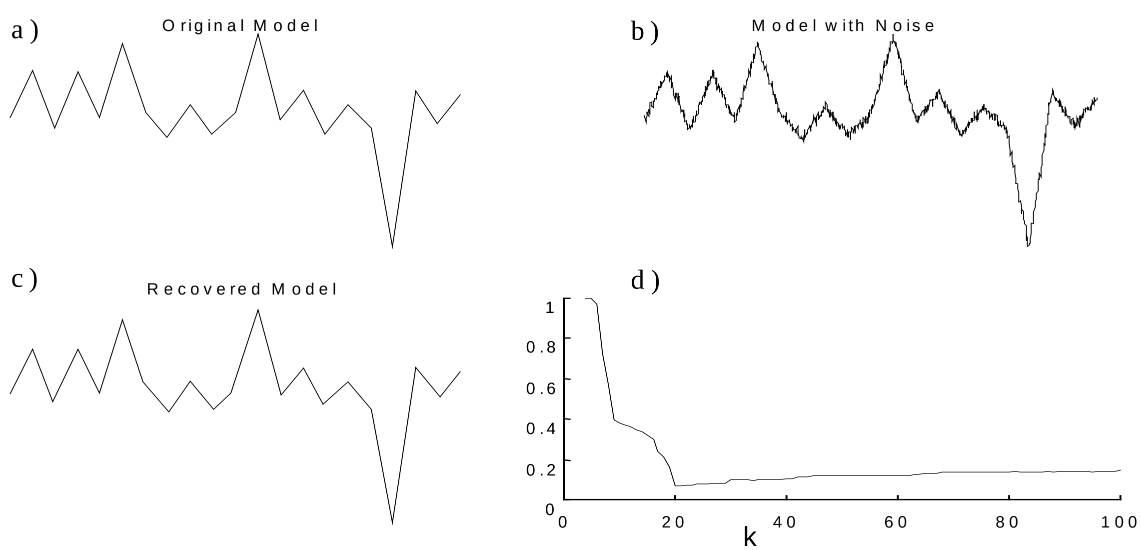


Figure 2: a) An artificial time series with a highly linear structure, containing 1000 points. **b)** A noisy version of **a**. **c)** The model recovered by our algorithm. **d)** The Balance of Error, B_k, plotted against k.

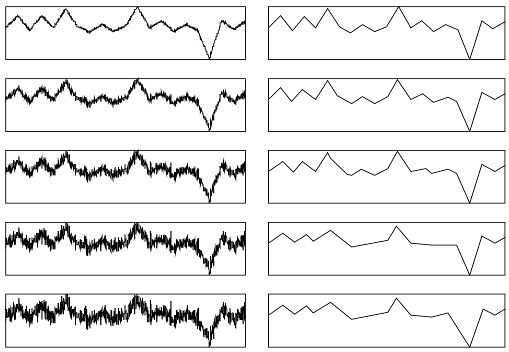


Figure 3: Different amounts of noise added to the same time series shown in Figure 2, and the recovered models.

distributed) noise we have. In general, neither of these assumptions holds, so we need to consider what our algorithm will do in these cases. Figure 3 shows the same time series shown in Figure 2.a with various amounts of noise and the models recovered. Given enough noise the algorithm does fail to recover the original model, but appears to degrade gracefully, capturing the overall shape of the underlying time series.

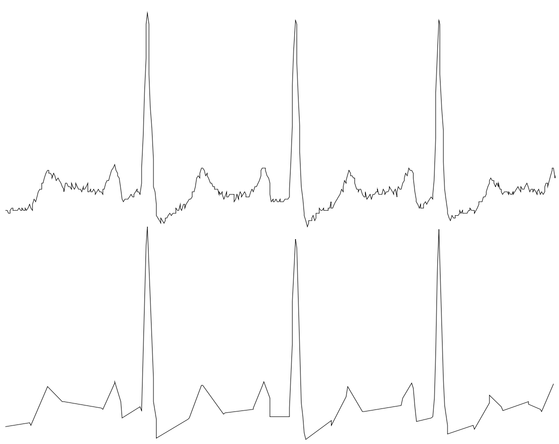


Figure 4: An electrocardiogram containing 512 points, and its 37-segment representation as produced by our algorithm.

Figure 4 shows the results of applying the algorithm to data which is clearly non-linear. Again, the segmented model appears to capture the essential shape of the underlying time series. To avoid confusion throughout the rest of the paper, we will refer to a piecewise-segmented linear representation of a time series as a 'sequence'.

Distance Measures

We begin this section by defining a simple distance measure, which does not possess robustness to longitudinal scaling. Later we will generalize this measure to incorporate longitudinal scaling and compare the two metrics.

A difficulty arises when attempting to compare two sequences represented by piecewise linear segments. In general, the endpoints of the segments do not line up. We solve this problem by projecting every endpoint on each sequence onto the other sequence. We measure the variance of the length of the projected lines, and use this as our distance measure. To get some intuition as to why this makes sense as a distance measure, consider the case where we are comparing two identical sequences. All the projected lines will have the same length, so their variance, and therefore their distance, will be zero. If there is a single feature present in one sequence and not the other, then at least one of the projected lines will be longer (or shorter) than the rest, and the variance and distance will suitably increase. For this measure to work, the two sequences must not cross. This is easily arranged by adding some large constant to all the y-values of one of the sequences.

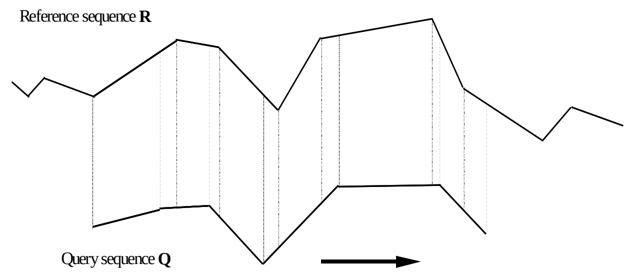


Figure 5: Subsequence matching with the segmented representation is accomplished by 'sliding' **Q**, along **R** and projecting the endpoints of each segment onto its corresponding place in the other sequence. The distance between two sequences is the variance of the projected lines.

Note that this measure handles:

- **Noise**. By segmenting the original time series we have 'canceled out' the effects of symmetrically distributed noise. We have also minimized the effects of lone spikes by averaging their effects over 'good' data.
- **Offset translation**. The variance of the projected lines is unaffected by the mean length of those lines, so shifting one of the sequences in the y-axis has no effect on the distance measure.
- **Amplitude Scaling**. There are occasions when we might want amplitude scaling to effect our distance measure, and other occasions when we want to be insensitive to it. In the latter case, we have merely two normalize the two sequences[2] before projecting the lines, and measuring their variance.

This distance measure has the following desirable properties.

$d(x,y) \geq 0 \;\; \forall \; x, y$

$d(x,y) = 0$ iff $x = y$

$d(x,y) = d(y,x)$

Having shown how to compare two sequences of the same length, we need now to consider how to do subsequence matching. This is accomplished in the obvious way. We take the shorter sequence, the Query Q, and 'slide' it along the longer Reference sequence R. We anchor the left edge of Q with the left side of every segment in R, then compute the distance between Q, and the appropriate subsection of R. We find the best match by keeping a *best-so-far* variable. This technique, which we call 'Segmented Sequential Scanning' has a time complexity of $O(MN)$, where M is the number of segments used to represent the time series R, and N is the number of segments used to represent the query Q. If c is the *average* number of datapoints represented by each individual segment in both Q and R, then we can rewrite the time complexity as $O((m/c)(n/c))$. This allows easy comparison to the time complexity for classic sequential scanning, and we can see that the speedup achieved is proportional to c^2. The actual value of c is domain dependent. In the experiments used to illustrate this paper it ranged from 6 to 578, giving a theoretical speedup of 36 to 334,084.

Having defined a *lockstep* distance measure that uses an underlying segmented representation, we will now show a generalized version that allows stretching and shrinking in the time axis. The format for the query now includes two numbers to represent the maximum stretch and the

2 We need to be careful when normalizing, not to be unduly influenced by a single feature, so rather than use the min and the max, we use the upper and lower quartiles as reference points.

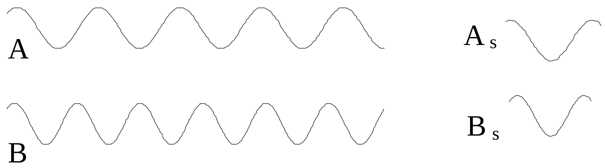


Figure 6: Although **A** and **B** will be considered very dissimilar by *lockstep* distance measures, short sub -sequences such $\mathbf{A_s}$ and $\mathbf{B_s}$ can always be found, which by themselves are arbitrarily similar.

maximum shrink allowed. We will represent this form of query by Q*. In this paper we assume for simplicity that we will allow the template to deform within this interval, without penalty. The more general form where we expect the Q* to be a certain length and penalize longer or shorter potential matches according to some probabilistic model is a simple generalization (Keogh and Smyth 1997, Burl and Perona 1996).

Our approach is based upon the simple observation that although a time series and a stretched version of itself are considered dissimilar to each other, short sub-sections of the time series, and the corresponding sub-sections from the stretched version will appear similar. Figure 6 demonstrates this phenomenon.

We will begin, therefore, by extracting two small sub-sections Q_l and Q_r from Q*, and searching for matches for these sub-queries using the *lockstep* algorithm described above (We will discuss the technique for selecting Q_l and Q_r below). We actually do two searches for each sub-query, one where it has been stretched by the maximum amount allowed, and one where it has been compressed by the maximum amount allowed. The result of this search is used to build a table as shown in Figure 7.

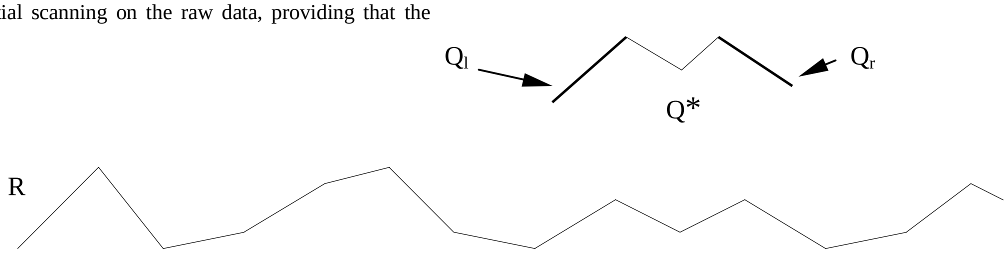
Row one shows the goodness of fit of Q_l when aligned with the i^{th} segment in R. Row two shows the goodness of fit of Q_r when aligned with the i^{th} segment in R. Note there are two numbers in each cell, corresponding to the stretched and compressed versions of the templates. Row three contains the x-values of the left side of the i^{th} segment in R. Note that no matter what length a potential overall match to our query is, the contribution of dissimilarity from Q_l and Q_r is bounded between the two values in the corresponding cell in the table.

We will now use this table to guide our search for Q*. First let us consider how to do a brute-force search of the table. We begin by assuming that Q_l is anchored in the first column in row 1 of the table. Given that the length of Q* is variable, the section representing Q_r might be in any one of several contiguous cells in row 2. For each possibility we compress or stretch Q* by an appropriate amount and compare this compressed or stretched query to the sequence R to obtain its goodness of fit. We obtain the appropriate scaling factor by dividing the difference of the values in row 3 of the two columns in question, by the standard length of Q* (Scaling Q* simply corresponds to multiplying its x-values by the scaling factor). Surprisingly, even doing this brute force search, the algorithm is still faster than sequential scanning on the raw data, providing that the average number of points represented by a segment is greater than about 5, and the maximum stretch is less than approximately 200%. Of course, we can avoid this brute force search using a variety of heuristics, including the following:

- Instead of searching left to right, it is better to search the table by anchoring Q_l at the most promising cells first, and keeping a *best-so-far* variable (The most promising cells are identified by looking at the lower of the two values). At every stage of the search we can eliminate all cells which have a minimum value greater than the current *best-so-far*, because they could not be part of an overall match which has a lower dissimilarity than our *best-so-far*. Typically this heuristic alone prunes 95% of the search space.
- Rather than using Q_l as the anchor point for the search, it might be better to use Q_r and search 'backwards'. This would be true if low values for Q_r are rarer than low values of Q_l. A simple test for skewness for each row will tell us the best row to search from.

In addition to the above, we can speed up the search by a careful choice of the segments chosen to represent Q_l and Q_r. By choosing wisely we can ensure that most of the values in the rows 1 and 2 are large, with just a few small values. This will ensure we find low values for *best-so-far* quickly, and can abandon the search early. For all the experiments shown in this paper, we simply use the leftmost segment to represent Q_l and the rightmost segment to represent Q_r. In some cases, however, we may have been better off choosing other segments, based on how often similar angles occur in R, the segments length (longer is better), and the distance between the two segments (further apart is better).

We call this algorithm S7 (Similarity Searching while Shrinking and Stretching using Segmented Sequential Scanning). The time complexity for S7 has two major contributions, the time taken to build the table, and the time taken to search the table. The time taken to build the table is less than or equal to O(*MN*), the time taken for a *lockstep* search. The time taken to search the table depends on the interval over which Q* may deform and on the structure data itself. In the pathological case where Q* is allowed to stretch over an interval as long as R and all the elements of the table are almost equal, we could be forced to examine the entire table. In general however, working with real data, we found the time needed to search the table is much less than the time need to build the table. The experimental results section below summarizes the empirical results of experiments on various data sets.



<table>
<tr><td>Q_l</td><td>.02 | .03</td><td>.92 | .91</td><td>.23 | . 27</td><td>.20 | .18</td><td>.14 | .15</td><td>.82 | .84</td><td>.72 | .70</td><td>.11 | .09</td><td>.30 | .02</td><td>.71 | .72</td><td>.77 | .75</td><td>.21 | .18</td><td>.83 | .84</td><td>.41 | .43</td></tr>
<tr><td>Q_r</td><td>.71 |. 72</td><td>.21 | .24</td><td>.45 | .47</td><td>.48 | .43</td><td>.53 | .56</td><td>.13 | .16</td><td>.32 | .35</td><td>.46 | .41</td><td>.27 | .24</td><td>.43 | .41</td><td>.12 | .15</td><td>.64 | .61</td><td>.71 | .73</td><td>.12 | .14</td></tr>
<tr><td>X Axis</td><td>0</td><td>21</td><td>43</td><td>57</td><td>81</td><td>99</td><td>119</td><td>145</td><td>161</td><td>177</td><td>196</td><td>220</td><td>241</td><td>267</td></tr>
</table>

Figure 7: The query Q* is decomposed into Q_l and Q_r (show in bold). Both are Q_l and Q_r matched against the reference sequence R, and the results are used to build a table.

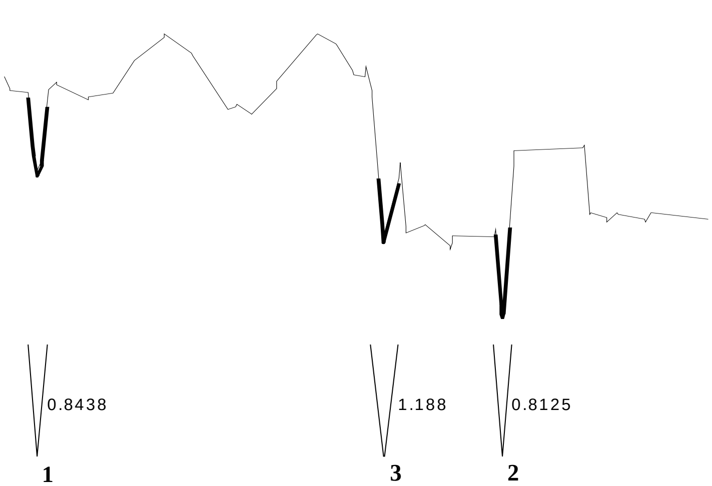


Figure 8: A simple Q* query on a Shuttle sensor. The three best matches are shown, with their scaling factors.

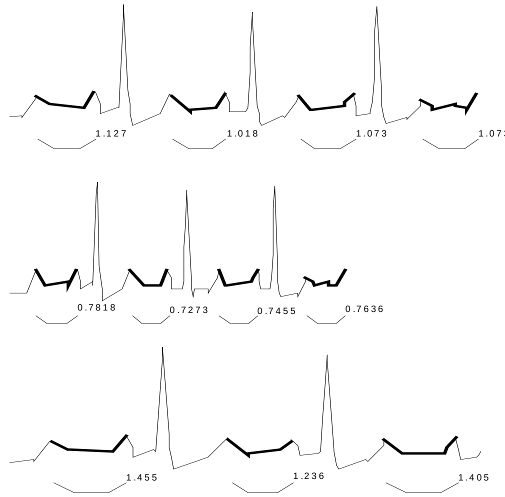


Figure 9: Our algorithm is able to track a particular part of an electrocardiogram, regardless of pulse rate.

Experimental Results

The above algorithm was tested on many data sets from medical, financial and engineering domains. Due to space limitations, we present only a small subset of the results. For each experiment we also ran classic sequential scanning as a strawman comparison. Figure 8 shows the result of a simple query on data from a Space Shuttle sensor (Shuttle STS-67 V71H3140B). The original time series has 22,627 points, which our segmentation algorithm transformed into 38 segments. The query Q* is a simple valley, which was allowed to stretch or compress by up to 25%. The three best matches are displayed, together with the amount of scaling necessary to achieve a fit. Surprisingly, one cannot duplicate this result with classic sequencing scanning. If you make the query template narrow enough to find the narrow valleys in the data (Figure 8, **1** and **2**), you are unable to find the wider valley (Figure 8, **3**) and if you make the query template wider, you can find the wider valley, but not the narrow ones.

Figure 9 shows an example of a query on an electrocardiogram. We built a simple three-segment query Q* which can find a certain part of the electrocardiogram. We built the initial template based upon visual inspection of an electrocardiogram recorded at a pulse rate of 70 beats per minute, and defined the limits of scaling based on knowledge of how variable a pulse rate can be. The algorithm easily finds the correct portions of the electrocardiogram, regardless of the patient's pulse rate.

Table 1 shows a comparison of our S7 algorithm vs classic sequential scanning. Note that the speedup achieved greatly depends on the average number of datapoints represented by a segment. The times reported for our algorithm do not include the time needed to segment the data; we assume that this is done off-line.

Related Work

Subsequence matching is a generalization of the much simpler problem of sequence matching. Here both sequences are assumed to be the same length. Agrawal et al. (1995) tackle this problem by dividing up the sequences into windows. Corresponding windows from two time series are compared, and the two time series are said to be similar if enough of these windows are similar. The individual windows are said to be similar if one lies within an envelope of a specified width around another. Each window has its data normalized to remove the effects of amplitude scaling and offset translation. A different approach favored by Berndt and Clifford (1994) uses dynamic time warping. Here two sequences are aligned, then one of them has its x-axis (time parameterization) locally stretched or compressed in order to minimize some distance function. Unfortunately, this technique is computationally very expensive, although it does lend itself to parallelization.

In general, pattern matching on 'raw data' is not feasible because of the sheer volume of data. In addition, raw data may contain spikes, drop-outs or other noise which could confuse the matching process. A variety of higher-level representations of times series have been proposed, most notably Discrete Fourier Transform. This approach involves performing a Discrete Fourier Transform on the original time series, discarding all but the K most informative coefficients, and then mapping these coefficients into K-dimensional space. The original work by Agrawal, Faloutsos and Swami (1993) only allowed the comparison of two time series of equal length, but was extended by Faloutsos, Ranganathan and Manolopoulos (1994) to include subsequence matching. Here they use a sliding window over the original time series to extract features. This process results in a trail in feature space. These trails are represented by their MBR's (Minimum Bounding Rectangles), and the MBR's are indexed by R*-

Data Set	Compression Rate Average number points per segment	Sequential scanning (Brute force approach) seconds	S7 seconds
Synthetic data	**32**	**471.2**	**4.8**
Shuttle	**578**	**4101.4**	**1.9**
Electrocardiogram	**61**	**1157.9**	**3.2**
Tree rings	**6**	**17.1**	**5.3**

Table 1

trees. The method is guaranteed to produce no false hits, but may produce false misses. There are limitations on the shortest possible query (it cannot be shorter than the width of the sliding window). Another problem with the method is that for queries greater than a certain length (the width of the sliding window), the query sequence must be decomposed and the results merged, greatly affecting the speed of the matching. Nevertheless, speedups of 3 to 100 times over sequential scanning are reported.
Others, including, Shatkay and Zdonik (1996) recognize that a piece-wise linear (and higher order) order representation, greatly reduces the required storage and search space for a time series, but fail to suggest a distance measure or search technique.

Summary

We have shown that a piecewise linear representation allows robust sequence matching which is much faster than current techniques. In addition we have presented an algorithm which is the first of which the authors are aware, to allow controlled longitudinal scaling. Some possible extensions and areas for future research include:

- Automating the process of selecting the best possible segments to use as Q_l and Q_r.
- Generalizing linear segments to higher order polynomials.
- Using the algorithm as a tool for learning relationships within, and between time series.
- Allowing non-uniform scaling, perhaps by using dynamic time warping (Brendt and Clifford 1994).

Acknowledgments

I would like to thank Dr. Michael Pazzani for his support and encouragement, Hagit Shatkay for the electro-cardiogram data, and Nira Brand for her valuable help and suggestions. All data mentioned in this paper are available in both original and segmented formats from http://www.ics.uci.edu/~eamonn/timeseries.html.

References

Agrawal, R., Faloutsos, C. and Swami, A. ‘Efficient Similarity Search in Sequence Databases’. In Proc. of the Fourth International Conference on Foundations of Data Organization and Algorithms, Chicago, October 1993.

Agrawal, R., Lin, K. I., Sawhney , H. S and Shim, K. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Times-Series Databases, In VLDB, September 1995.

Berndt, D. J., Clifford, J. ‘Using Dynamic Time Warping to Find Patterns in Time Series’. In KDD-94: AAAI Workshop on Knowledge Discovery in Databases. Pages 359-370, Seattle, Washington, July 1994.

Burl, M., Perona, P., ‘Recognition of Planer Class Objects’, Proceedings of the 1996 Computer Vision and Pattern Recognition Conference, IEEE Press, 1996.

Cheng, Y. C., Lu, S. Y. ‘Waveform Correlation using Tree Matching:’ IEEE Conf. PRIP, 1982.

Ehrich, R. W, and Foith, J. P., ‘Representation of Random Waveforms by Relational Trees’, IEEE Transactions On Computers, Vol. C-25, NO. 7, July 1976.

Faloutsos, C., Ranganathan, M., Manolopoulos, Y. ‘Fast Subsequence Matching in Time-Series Databases’, SIGMOD - Proceedings of Annual Conference, Minneapolis, May 1994.

Hagit, S., Zdonik, S. ‘Approximate Queries and Representations for Large Data sequences’, Proc. 12th IEEE International Conference on Data Engineering. Pages 546—553, New Orleans, Louisiana, February 1996.

Keogh, E., Smyth, P., ‘Probabilistic Approach to Fast Pattern Matching in Time Series Databases’, To appear KDD-97.

Lawson, C. L., ‘Characteristic Properties of the Segmented Rational Minimax Approximation Problem’, Numerical Math., Vol. 6, pp. 293-301. 1964.

Pavlidis, T., ‘Waveform Segmentation Through Functional Approximation’, IEEE Transactions on Computers, Vol, C-22, NO. 7 July 1976.

Pavlidis, T., Horowitz, S., ‘Segmentation of Plane Curves’, IEEE Transactions on Computers, Vol. C-23, NO 8, August 1974.

Shaw, S. W., DeFigueiredo, R. J. P. ‘Structural Processing of Waveforms as Trees’. IEEE Transactions on Acoustics, Speech, and Signal Processing. Vol. 38 No 2 February1990.

Wang, J. T., Zhang, K., Jeong, K and Shasha. ‘A System for Approximate Tree Matching’, IEEE Transactions on Knowledge and Data Engineering, 6, no 2 April 1994.