Relevance Feedback Retrieval of Time Series Data

Eamonn J. Keogh
Dept of Info and Computer Science
University of California, Irvine
California 92697 USA
(949) 824-7210
eamonn@ics.uci.edu

Michael J. Pazzani
Dept of Info and Computer Science
University of California, Irvine
California 92697 USA
(949) 824-7405
pazzani@ics.uci.edu

ABSTRACT

There has been much recent interest in retrieval of time series data. Earlier work has used a fixed similarity metric (e.g., Euclidean distance) to determine the similarity between a user-specified query and items in the database. Here, we describe a novel approach to retrieval of time series data by using relevance feedback from the user to adjust the similarity metric. This is important because the Euclidean distance metric does not capture many notions of similarity between time series. In particular, Euclidean distance is sensitive to various "distortions" such as offset translation, amplitude scaling, etc. Depending on the domain and the user, one may wish a query to be sensitive or insensitive to these distortions to varying degrees. This paper addresses this problem by introducing a profile that encodes the user's subjective notion of similarity in a domain. These profiles can be learned continuously from interaction with the user. We further show how the user profile may be embedded in a system that uses relevance feedback to modify the query in a manner analogous to the familiar text retrieval algorithms.

Keywords

Time series, multimedia data, relevance feedback, modeling user subjectivity.

1. INTRODUCTION

Time series account for a significant portion of the data stored in business, medical, engineering, and social science databases. There are innumerable statistical tests one can perform on time series, such as determining autocorrelation coefficients, measuring linear trends, etc. Much of the utility of collecting this data, however, comes from the ability of humans to visualize the shape of the (suitably plotted) data. For example:

- Cardiologists view ECGs to diagnose arrhythmias [8].
- Chartists examine stock market data, searching for certain shapes that are thought to be indicative of a stock's future performance [5].
- Astronomers examine star light curves (the changes in frequency over time) to classify stars [13].

Unfortunately, the sheer volume of data collected means that only a small fraction can ever be viewed. For example, the MACHO dataset, a collection of time series consisting of star light intensities, is over 500 gigabytes [17]. Given the size of such datasets, most of the research on information retrieval for time series has focused on speeding up the retrieval time for query by content. However, two important issues in time series retrieval have not yet been explored.

- Relevance Feedback: In time series domains, as in text domains, users may not initially know how to form a query to retrieve precisely what they are looking for. In text domains, relevance feedback is used to solve this problem. In this paper, we show how relevance feedback may be applied to retrieval of time series data to learn which sections of the time series are most significant in a manner analogous to modifying the weight of terms in text retrieval. For example, a cardiologist might want to retrieve the electrocardiograms of previous patients that are similar in some way to that of the current patient. By providing feedback on the quality of the initial items retrieved, the cardiologist can indicate which similarities are significant for this task.
- Subjectivity of similarity: Most work on retrieval of time series has used Euclidean distance (or some approximation or extension thereof) as the distance measure. However, there is little evidence that Euclidean distance maps onto human intuition of similarity. Indeed the reverse appears to be true. Consider Figure 1, which shows clusters of four small data sets where similarity is determined by Euclidean distance. In each case, Euclidean distance indicates that the two lower items are the most similar. However, it is obvious that the two upper time series are most similar. The "correct" distance measure depends upon the user and problem, and it should be continuously learned as the user interacts with the system [15].

In this paper we address both relevance feedback and subjective measures of similarity. In Section 2 we introduce a new high-level representation of times series, and in Section 3 we show how this representation allows relevance feedback retrieval of time series data. In Section 4 we further show a method for dealing with subjectivity by modeling the user's subjective judgment of similarity.

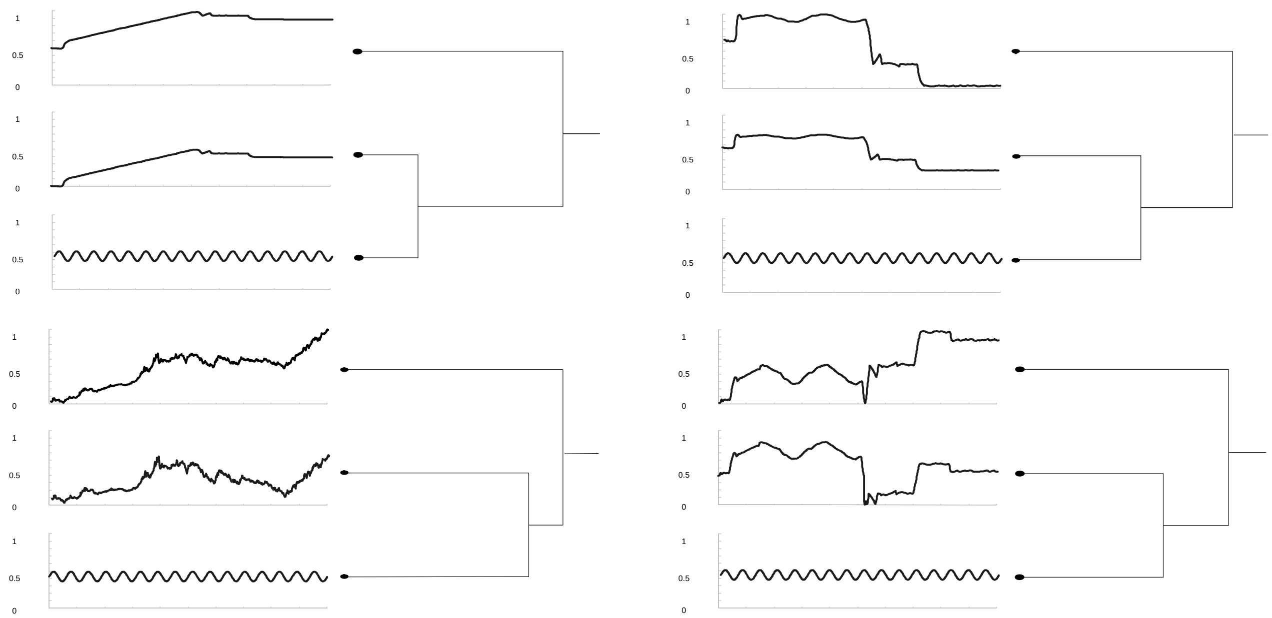

Figure 1: The Euclidean distance measure can produce unintuitive clustering of time series data. Clockwise from the top left, unintuitive clustering caused by offset translation, amplitude scaling, discontinuities and linear trends. Although the top two time series appear most similar in all four cases, Euclidean distance indicates that the bottom two are most similar

2. REPRESENTING TIME SERIES

Using the original ‘raw’ data for query by content in time series databases is computationally expensive (especially for an interactive system) and fails to abstract key features of the data. What is needed is a higher-level representation. Several such representations have been proposed, and we refer the interested reader to [10] for a detailed discussion of their rival merits. Piece-wise linear segmentation, which attempts to model the data as sequences of straight lines, (as in Figure 2) has many advantages as a representation. Pavlidis and Horowitz [14] point out that it provides a useful form of data compression and noise filtering. Shatkay and Zdonik [7] describe a method for fuzzy queries on linear (and higher order polynomial) segments. Keogh and Smyth [11] further demonstrate a framework for probabilistic pattern matching using linear segments and present an algorithm for segmenting time series.

In previous work on pattern matching using piece-wise linear segments when comparing two time series to see if they are similar, all segments are considered to have equal importance. However, one may wish to assign different weights to different parts of the time series. As an example, consider the problem of pattern matching with electrocardiograms. A cardiologist attempting to diagnose a recent myocardial infarction will pay close attention to the S-T wave (as shown in Figure 2) and downplay the importance of the rest of the electrocardiogram. If we wish an algorithm to reproduce the cardiologist’s ability, we need a representation that allows us to weight different parts of a time series differently. In this paper we adopt piece-wise linear segments to represent the shape of a time series, and augment the queries with a weight vector that contains the relative importance of each individual linear segment. The weight vector associated with the query may be updated by relevance feedback.

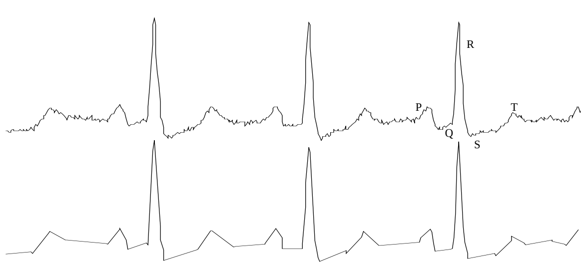

Figure 2: An example of a time series and its piece-wise linear representation

2.1 Notation

For clarity we will refer to ‘raw’, unprocessed temporal data as time series and to a piece-wise representation of a time series as a sequence. We will use the following notation throughout this paper. A time series, sampled at *k* points, is represented as an uppercase letter such as A. The segmented version of A, containing *K* linear segments, is denoted as a bold uppercase letter such as **A**, where **A** is a 5-tuple of vectors of length *K*.

$$\mathbf{A} \equiv \{AXL, AXR, AYL, AYR, AW\}$$

The i^{th} segment of sequence **A** is represented by the line between (AXL_i AYL_i) and (AXR_i AYR_i), and AW_i, which represents the segment's weight. Figure 3 illustrates this notation. After a time series is segmented to obtain a sequence, we initialize all the weights to one. Thereafter, if any of the weights are changed, the weights are renormalized such that the sum of the products of each weight with the length of its corresponding segment, equals the length of the entire sequence, so that the following is always true:

$$\sum_{i=1}^{K} W_i * (AXR_i - AXL_i) = AXR_k - AXL_1$$

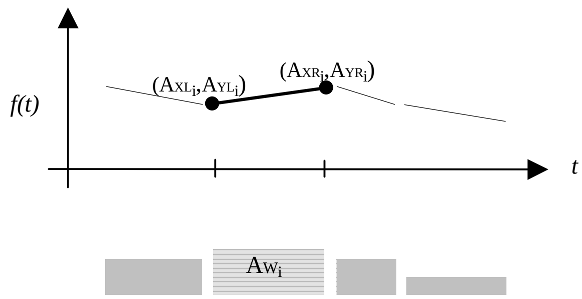

Figure 3: We represent times series by a sequence of straight segments, together with a sequence of weights (shown as the histogram) which contain the relative importance of each segment

This renormalization is important because it gives the following property: the total weight associated with a sequence of a given length is constant, regardless of how many segments are used to represent it.

2.2 Comparing Time Series

To retrieve time series, we need a distance (or conversely similarity) measure to compare the query with segments stored in the database. We start by adopting a distance measure, *DS*, that approximates the Euclidean distance measure on the 'raw' data to facilitate comparison to earlier work [2],[3],[6]. However, we also allow for a weight to be associated with each segment in the query. In Section 4 we further show to generalize this distance measure to allow for user subjectivity to distortions caused by offset translation, amplitude scaling, discontinuities and linear trends.

It is convenient for notational purposes to assume that the endpoints of the two sequences being compared are aligned in the X-axis. If this is not the case, the sequences are aligned to split some segments at the point that corresponds to an endpoint in the other sequence. Figure 4 illustrates this process. We call each pair of segments that are aligned in the X-axis a slice, and we note that each slice has a weight. First, we show how to measure the weighted Euclidean distance between the pair of segments in a slice. The overall distance between two sequences is simply the square root of the summation over all such slices.

The intuition behind the *DS* approximation of Euclidean distance is to imagine that the two segments in a slice have been replaced with raw data. Figure 4.b illustrates this idea, with phantom lines drawn between each corresponding pair of points. We need to sum the squares of the lengths of all such lines. Rather than actually perform a summation, we use an equivalent closed form equation. Let L be the absolute difference in Y-axis values between the left endpoints of the two segments. Let R be the absolute difference in Y-axis values between the right endpoints of the two segments. Let x be the length of the slice. Let c be $\min(L,R)$ and let $\Delta = (\max(L,R) - c)\,/x$. The sum of squares between the two segments in the i^{th} slice is:

$$ss_i = (x+1)\left(c^2 + c\Delta x + \tfrac{1}{6}\Delta^2 x(2x+1)\right)$$

The weighted distance between two sequences of length i is:

$$DS(A,B) = \sqrt{\sum_{i \in slice} ss_i \times sliceW_i}$$

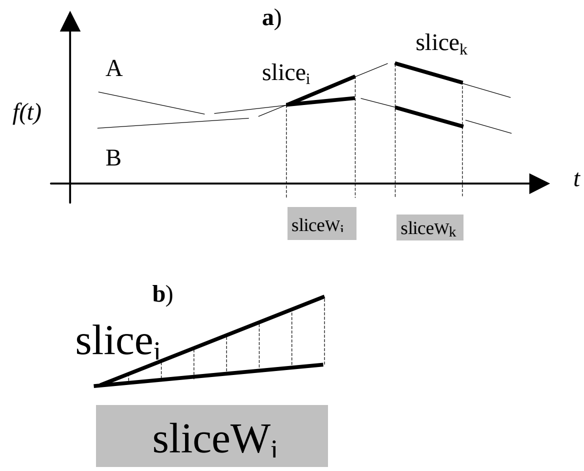

Figure 4: a) Two examples of slices. **b)** A "zoom-in" of a $slice_i$ with lines shown-between the two segments using the same granularity as the original time series

2.3 Merging Time Series

In this section we define an operation on sequences which we call 'merge'. The merge operator combines two sequences and is analogous to weighted vector addition used in relevance feedback of text. The basic idea is that the merge operator takes two sequences as input and returns a single sequence whose shape is a compromise between the two original sequences, and whose weight vector reflects how much corresponding segments in each sequence agree. Repeated application of the merge operator allows multiple sequences to be combined.

When merging two sequences, one may wish for one of the two input sequences to contribute more to the final sequence than the other does. To accommodate this, we associate a term called 'influence' with each of the input sequences. The influence term associated with a sequence **S** is a scalar, denoted as **S**I, and may be informally considered a 'mixing weight'.

The algorithm shown in Table 1 takes the two sequences **A** and **B** with influence terms **A**I and **B**I respectively, and outputs a new sequence **C**. Figure 5 shows two sequences that have been merged with various values for the influence terms.

```
if (AI * BI < 0) then sign = -1
else sign =  1
end
mag  = min(|AI|,|BI|) / max(|AI|,|BI|)
scale = max(max(AYL),(AYR))- min(min(AYL),(AYR))
for i = 1 to K
    CXL_i = AXL_i
    CXR_i = AXR_i
    CYL_i = ((AYL_i * AI)+(BYL_i * BI))/(AI+BI)
    CYR_i = ((AYR_i * AI)+(BYR_i * BI))/(AI+BI)
    run   = AXR_i - AXl_i
    rise  = |(AYL_i - BYL_i)-(AYR_i - BYR_i)|
    diff  = (rise / run) * scale
    CW_i = (AW_i*BW_i)*(1+(sign * mag)/(1 + diff))
end
```

Table 1: The merge algorithm.

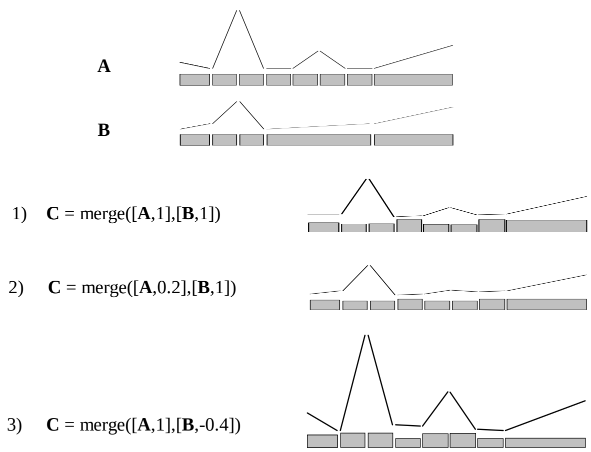

Figure 5: Examples of the merge operator with various influence terms

1) With two equal influence terms, the shape of the resultant sequence **C** is “halfway” between **A** and **B**.
2) With **B**’s influence term much larger than **A**’s, the shape of the resulting sequence **C** is much closer to **B** than **A**.
3) With a negative influence term for **B** the shape of the resulting sequence **C** looks like **A** where the differences between **A** and **B** have been exaggerated.

3. QUERY REFINEMENT VIA RELEVANCE FEEDBACK

Relevance feedback is the reformulation of a search query in response to feedback provided by the user for the results of previous versions of the query. It has an extensive history in the text domain, dating back to Rocchio’s classic paper [18]. Recently, it has seen application to multimedia domains, notably the MARS project, an image retrieval system [19]. However, no one has attempted explore time series databases in a relevance feedback framework, in spite of the fact that relevance feedback has been shown to significantly improve the querying process in text databases [16]. In this section we present a simple query refinement algorithm which utilizes the merge operation defined in Section 2.

3.1 Formulating a New Query

The query refinement algorithm works in the following manner. An initial query sequence **Q** is used to rank all sequences in the database. This query may be hand drawn by the user or it may be a sequence or subsequence from the database. The best *n* sequences are retrieved and shown to the user. The user assigns ratings to the retrieved sequences on a scale from –3 to +3. After the user has evaluated the top *n* sequences, the query update rule in table 2 is used to produce a new query $\mathbf{Q}_{new}$, and the search process begins again.

```
Obtain the initial query Q
while user not finished do
  Find the n sequences that minimize DS(Q,S_i)
  Display the sequences to the user
  Assign the user’s rating for the i^th sequence to SI_i
  Q_new  = merge([Q_old,Q_old I],[S_1,S_1 I],[S_2,S_2 I],…,[S_n,S_n I])
  Q_new I = Q_old I + S_1 I + S_2 I + … + S_n I
end
```

Table 2: An outline of the query refinement algorithm.

Figure 6 shows a sample screen shot of the system.

3.2 Evaluating Query Refinement

To evaluate the effectiveness of the query refinement algorithm to converge on the intended query, we generated synthetic data of two similar time series and provided positive feedback for one type. We constructed 500 “Type A”, and 500 “Type B” time series, which are defined as follows:

- Type A: $\mathrm{Sin}(x^3)$ normalized to be between zero and one, plus Gaussian noise with $\sigma = .1$ $-2 \le x \le 2$
- Type B: $\mathrm{Tan}(\mathrm{Sin}(x^3))$ normalized to be between zero and one, plus Gaussian noise with $\sigma = .1$ $-2 \le x \le 2$

The time series, which were sampled at 800 points, were segmented. Figure 7 shows an example of each type. Note that they are superficially very similar, although Type B has a somewhat sharper peak and valley. We built an initial query by averaging all 1,000 time series and segmenting the result.

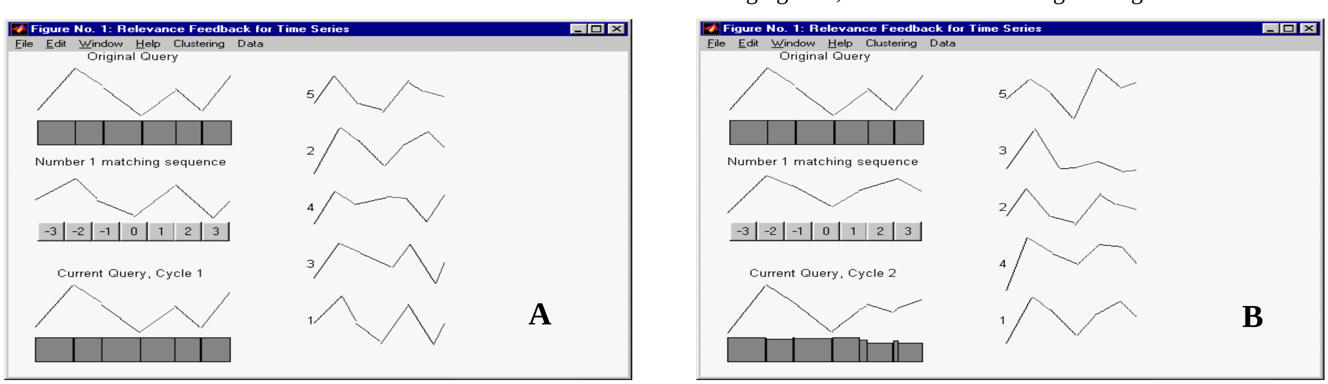

Figure 6: Screen shots of the relevance feedback graphic user interface. The four areas of each screen are as follows:
Top left : The original query, as drawn by the user.
Right : The best 5 matches to the query.
Center left : The "ranking window", where the user gives feedback on 5 best matches in the form of a number from –3 to 3.
Bottom left : The current query, automatically constructed from the original query and user feedback on retrieved items.
In this simple example a query was executed to retrieve the 5 best matches as shown in A. The user rated the number 2 matching sequence as a '3', and all others as '0'. Note that the original query and the number 2 matching sequence differ greatly on the right side. This results in the current query in B having low weights on the right side and the "softening" of the rightmost peak. This in turn changes the 5 best items retrieved in B

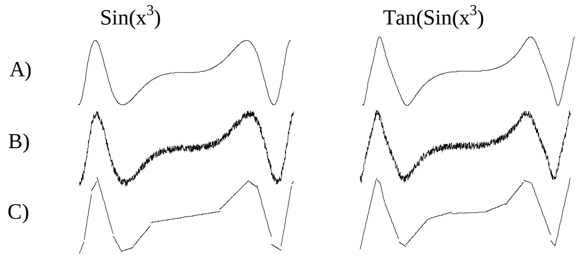

Figure 7: Synthetic data created for relevance feedback experiment. **A)** The original time series. **B)** The original time series with noise added. **C)** The segmented version of B

Twenty-five experimental runs were made. Each run consisted of the following steps. It was randomly decided whether Type A or Type B was to be the "target" shape for that particular experimental run. The initial query was made, and the best 15 sequences were shown to a user, who then rated them by assigning ratings that reflected how closely he thought they resembled the target shape. A refined query was built and the search/rate process was repeated twice more.

We evaluated the effectiveness of the approach by measuring the average precision of the top 15 sequences and the precision at the 25, and 50 percent recall points. Precision (*P*) is defined as the proportion of the returned sequences which are deemed relevant, and recall (*R*) is defined as the proportion of relevant items which are retrieved from the database. These results are shown in Table 3.

	Initial Query	Second Query	Third Query
P of top 15	.51	.91	.97
P at 25% *R*	.52	.91	.96
P at 50% *R*	.49	.89	.95

Table 3: Results of query refinement experiments

As one might expect, the initial query returns the sequences in essentially random order. The second query produces remarkable improvement, and the third query produces near perfect ranking. This experiment demonstrates the ability of the system to converge rapidly on an accurate refined query.

4. HANDLING SUBJECTIVITY OF SIMILARITY

The representation of time series presented above provides a flexible query language allowing arbitrary shapes and weights. However, it is still possible that a query using this representation could fail to retrieve an item that the user would have found relevant. Consider the problem of offset translation, illustrated in Figure 8. Two similar (or even identical) shapes can have an arbitrarily large dissimilarity because they are separated in the Y-axis. As a familiar real-world example, consider two stocks, whose values fluctuate around $100 and $20 respectively. It is possible that the stock movements are very similar but are separated by a constant amount. Agrawal et al [2] and Das et al [3] have noted this problem before. Their solution is to normalize the two sequences being compared such that they have the same mean. However this approach significantly reduces the discriminating power of the query language. For example suppose a user wishes to know whether this database contain a plateau at $40. Using the retrieval method proposed in Section 3 of the paper, the user could simply draw a straight line of the appropriate length at the $40 level, and use this as a query. Using the "normalizing" query systems of Agrawal et al or Das et al, however, will result in all plateaus being returned, regardless of their mean value (i.e plateaus at $1 or $120).

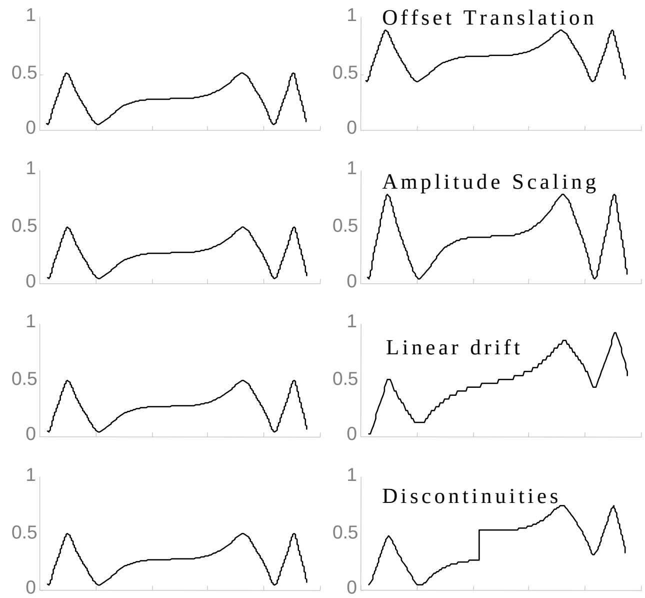

Figure 8: The four global distortions defined and discussed in Section 4.1

Whether a query should be sensitive to offset translation or not is clearly domain dependent. In stock market analysis, chartists typically wish to be insensitive to it. However in a database which contains patients temperatures, we definitely need to be able to differentiate between a patient whose temperature hovers around 98 degrees and a patient whose temperature hovers around 103 degrees. The stock market and medical database examples presented above represent the two extremes of sensitivity/insensitivity to offset translation. However it is generally the case that a user is willing to trade off some fidelity to shape for fidelity to offset. Naturally the amount trade-off will depend on the user and their information goals and may change during the process of interacting with the dataset. In this section, we show how to deal with this problem by introducing a user profile that encodes the user's subjective tolerance to offset translation and other global distortions of sequences.

4.1 Global Distortions

In this section we introduce four global distortions which can occur in time series databases and show how we measure them. Note that in every case the distortion in the database is relative to a particular query. The term distortion is not meant to imply that the data is somehow corrupt. It simply means that some simple transformation of candidate sequence would greatly reduce the Euclidean distance between the query and the candidate sequence.

Offset translation was mentioned in the previous section. The amount of offset translation between a query and another sequence is denoted by the scalar *O* and is calculated as follows:

$$OT(Q,S) \equiv \big(1+\big(mean(Q)-mean(S)\big)\big)/\,2$$

Where *mean*(**S**) is simply the mean Y value of the sequence **S**. Note that $0 \le O \le 1$. Values close to ½ indicate that little or no translation occurred. Values less than ½ indicate that the candidate sequence **S** was shifted down to achieve a better match and values greater than ½ indicate that the candidate sequence was shifted up to achieve a better fit.

Amplitude scaling, where two sequences are alike, but one has been 'stretched' or 'compressed' in the y-axis, can be dealt with similarly. It simply requires normalizing the sequences before applying the distance operator. Agrawal et al. [2] describe how to do this with raw time series. Normalizing with sequences is similar, but can be accomplished *k*/*K* times faster. The amount of amplitude scaling is denoted by the scalar *A* and is calculated as follows:

$$AS(Q,S) \equiv \begin{cases} if \;\; std(Q) \le std(S) & (std(Q)/std(S))/2 \\ else & (2 - std(S)/std(Q))/2 \end{cases}$$

Where *std*(**S**) is simply the standard deviation of the sequence **S**. Note that $0 \le A \le 1$. Values close to ½ indicate that little or no rescaling occurred. Values less than ½ indicate that the candidate sequence **S** was 'compressed' to achieve a better match and values greater than ½ indicate that the candidate sequence was 'stretched' to achieve a better fit.

Linear drift occurs naturally in many domains. As an example, consider two time series, which measure the absolute sales of ice cream in two cities with similar climates and populations. We would expect the two time series to be very similar, but if one city's population remains constant while the other experiences steady growth, we will see linear drift. The amount of linear detrending necessary to improve the fit between is denoted by the scalar *L* and is calculated as follows:

$$LD(Q,S) \equiv \left(1 + \left(\frac{slope(Q) - slope(S)}{maxpossible\ slope} \right) \right) \Big/ 2$$

Where *max possible slope* is the greatest possible slope difference, easily calculated since we know the Δx, the length of the query, and Δy is one. Note that $0 \le L \le 1$. Values close to ½ indicate that little or no detrending occurred. Values less than ½ indicate that the candidate sequence **S** had some trend removed to achieve a better match and values greater than ½ indicate that the candidate sequence had some trend added to achieve a better fit.

Finally, some datasets contain discontinuities. These are typically sensor calibration artifacts (caused by power spikes for example), but may have other causes. We introduce the following definition to facilitate discussion:

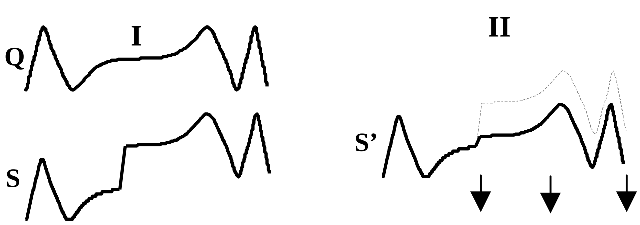

Figure 9: How the relative discontinuity term *R* is calculated. **I**: The distance *DS*(**Q**,**S**) is calculated. **II**: A single subsection of **S** is translated such that *DS*(**Q**,**S'**) is minimized. *RD*(**Q**,**S**) is defined as *DS*(**Q**,**S'**) /*DS*(**Q**,**S**)

Definition 1 (Relative Discontinuity):

A sequence **S** is said to be relatively discontinuous with respect to a query **Q**, if translating a single subsection of **S** in the y-axis results in a new sequence **S'**, with *DS*(**Q**,**S**) » *DS*(**Q**,**S'**). The amount of relative discontinuity is denoted by the scalar *R*, and is calculated as follows:

$$RD(Q,S) \equiv DS(Q,S') / DS(Q,S)$$

Note that $0 \le R \le 1$. A value for *R* close to one indicates the absence of a discontinuity. Smaller values indicate that a relative discontinuity was detected.

4.2 Representing the User Profile

Given that we can measure the distance between two time series and we can measure the various global distortions discussed above, we now have a framework for measuring the subjective distance between two time series **Q** and **S**. The idea is that we shift, rescale, detrend and remove relative discontinuities from **S** to produce a new sequence **S'**. We then measure the distance between **Q** and the transformed sequence **S'**. The subjective distance is defined as:

$$sub\_dis(Q,S) \equiv DS(Q,S') \times TransformationPenalty(S,S')$$

Where *TransformationPenalty*(**S**,**S'**) is the "cost" of converting **S** into **S'**. This cost depends on the user's profile, their subjective judgement of the desirability of the four distortions in a particular domain.

We need to represent the user's subjective preferences. We do this by learning the parameters of a "preference" distribution for each distortion. These distributions are defined only in the range of [0,1], which is also the range for all the global distortions. The height of the distribution represents the relative desirability of a particular amount of distortion. For example, if $F_{OT}(x)$ is the distribution function for the offset translation, and a sequence **S** had its offset translation relative to **Q** measured as *O* = (**Q**,**S**), then the penalty term for offset translation is $f_{OT}(O)$ / argmax($f_{OT}(x)$).

For simplicity we make the assumption that each of the global distortions are independent of each other. Given this assumption we can define the TransformationPenalty as:

$$TransformationPenalty(S,S') \equiv$$

$$\frac{f_{OT}(O)}{\mathrm{argmax}(f_{OT}(x))} \times \frac{f_{AS}(A)}{\mathrm{argmax}(f_{AS}(x))} \times \frac{f_{LT}(L)}{\mathrm{argmax}(f_{LT}(x))} \times \frac{f_{RD}(R)}{\mathrm{argmax}(f_{RD}(x))}$$

We chose the beta function to represent each of the preference distributions because it provides a rich class of distributions that can closely approximate many other functions, including the uniform, exponential, Gaussian, log-normal and linear distributions. Figure 10 shows a few of the different shapes that the beta distribution can take on. The shape of the beta distribution is determined by two parameters *r* and *s*. It is defined in the range of [0,1] which is also the range of all the preference distributions.

4.3 Learning the User Profile

We now show that it is possible to infer the user's subjectivity preferences from the user's estimates of the similarity between sequences by fitting a beta distribution to data [4]. However we have two addition questions must be addressed:

1) How should we initialize the distributions for the first search?

2) When we get feedback for an item, how do we know how much of it is attributable to the global distortions as opposed to the actual shape of the item?

The first of these problems we deal with in the following manner. The first time the user uses the system they are draw a query shape. Before a search of database occurs, the system produces multiple copies of the query and adds various amounts of the four global distortions to them. The user is asked to rate these and the ratings are used to learn the initial user profile.

The second of these problems is also easily dealt with. The solution we use is to show the user both the original sequence **S** and the transformed sequence **S'** and ask them to rate them independently. We can then easily decide the appropriate corrections to the actual shape of the item (using the query update rule in Table 2) and the user profile. The following illustrates how the parameters for the offset translation preference distribution are learned. The other preference distributions are learned in a similar fashion.

For each of the *n* sequences the search algorithm returns we do the following. We measure the amount of offset translation O_i. Next we present the user both the original sequence **S** and the transformed sequence **S'** and obtain ratings for each. The ratio of these two ratings are use as a weights to estimate the weighted mean and weighted variance of the distribution of all *n* offsets. We can then estimate the parameters of the beta distribution as:

$$\hat{r} = \hat{\mu}\left[\frac{\hat{\mu}(1-\hat{\mu})}{\hat{\sigma}^2} - 1\right] \qquad \hat{s} = (1-\hat{\mu})\left[\frac{\hat{\mu}(1-\hat{\mu})}{\hat{\sigma}^2} - 1\right]$$

Table 4 shows an outline of the new search algorithm, which we call Subjective-Distance Retrieval (SDR):

```
Initialize the user profile and obtain the initial query Q.
while user not finished do
 for each candidate sequence S in the database.
  Calculate O = OT(Q,S), remove offset
  Calculate A = AS(Q,S), remove amplitude scaling
  Calculate L = LT(Q,S), remove linear trend
  Calculate R = RD(Q,S),remove relative discontinuity
 sub_dis(Q,S) = DS(Q,S') × transformationPenalty(S,S')
 end
  Display the n best matches, and gather feedback.
  Update the queries shape and weight vector.
  Update the user profile.
end
```

Table 4: Outline of the Subjective-Distance Retrieval algorithm.

4.4 Using Subjective Distance Measures

To illustrate the utility of using user profiles, we did the following experiment. Using a database consisting of RR intervals obtained from Holter ECG tapes [20], we built a query and performed an initial search using Euclidean distance. A user examined and rated the first 10 sequences returned, and repeated the search in two different ways.

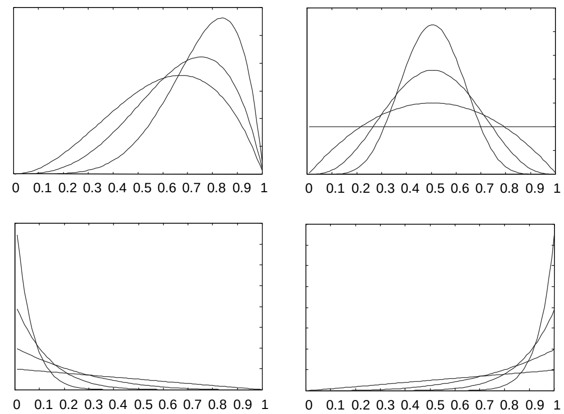

Figure 10: We use the beta distribution to model the user's preferences because it can approximate a wide variety of distributions over the range [0,1], which is also the range of all four global distortions

1) Using the query update rule discussed in section 3, *without* using the subjective distance measures.

2) Using the SDR algorithm, incorporating subjective distance measures.

The four top ranked sequences from Euclidean distance, relevance feedback query refinement without the subjective distance measure and relevance feedback query refinement with the subjective distance measure are show in Figure 11. The four sequences shown in Figure 11.A minimize the Euclidean distance to the initial query. The four sequences shown in 11.B minimize the weighted distance to the refined query, and the four sequences in Figure 11.C, when transformed by the user subjective distance measure, are most similar to the original query. Only the results retrieved with the subjective distance measure allow for similarity of shape while tolerating differences in offset, amplitude, etc

5. RELATED WORK

There has been no work on relevance feedback for time series. However, in the text domain there is an active and prolific research community. Salton and Buckley [16] provide an excellent overview and comparison of the various approaches. Picard [15] has long stressed the need to model subjectivity in information retrieval and has implemented image retrieval called FourEyes that attempts to learn the (subjective) weighting of features which are best suited to an image retrieval [12]. We have shown how both of these ideas may be applied to retrieval of time series data.

6. CONCLUSION

We introduced an enhanced representation of time series and demonstrated its utility for relevance feedback. We further demonstrated a method for dealing with user subjectivity by encoding a user profile.

Possible areas for future research include incorporation of other global distortions, especially time axis distortions, and building an indexing system to speed up the interaction times.

7. REFERENCES

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

[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. In VLDB, September.

[3] Das, G., Lin, K., Mannila, H., Renganathan, G. & Smyth, P. (1998) Rule discovery from time series. KDD. pp. 16-22.

[4] Derman, C., Gleser, L. G. & Olkin, I (1973). A guide to probability, theory and application. Holt, Rinehart and Winston, Inc. New York. ISBN 0-03-0-78885-4.

[5] Edwards, R . D. (1997) Technical analysis of stock trends. MaGraw-Hill ISBN: 0814403735.

[6] Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. SIGMOD - Proceedings of Annual Conference, Minneapolis, May.

[7] Hagit, S., & Zdonik, S. (1996). Approximate queries and representations for large data sequences. Proc. 12th IEEE International Conference on Data Engineering. pp 546-553, New Orleans, Louisiana, February.

[8] Jenkins, D. & Gerred, S. (1997). ECGs by example. Churchill Livingstone ISBN 0 443 056978.

[9] Keogh, E. (1997). Fast similarity search in the presence of longitudinal scaling in time series databases. Proceedings of the 9th International Conference on Tools with Artificial Intelligence. pp 578-584. IEEE Press.

[10] Keogh, E. & Pazzani, M. (1998) Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining. pp 239-243, AAAI Press.

[11] 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, AAAI Press.

[12] Minka., T., P. & Picard., R., W. (1997) Pattern Recognition, Interactive Learning using a Society of Models, Vol. 30, pp. 565, also MIT Media Lab Perceptual Computing TR #349

[13] Ng, M.K., Huang, Z. (1997). Temporal data mining with a case study as astronomical data analysis. Lecture Notes in Computer Sciences. Springer. pp 2-18.

[14] Pavlidis, T., Horowitz, S., (1974). Segmentation of plane curves. IEEE Transactions on Computers, Vol. C-23, No 8, August.

[15] Picard R.W (1995). Computer learning of subjectivity. ACM Computing Surveys, Vol. 27, No. 4, pp. 621-623, Dec.

[16] Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. JASIS 41. pp. 288-297.

[17] Skinner, K. H. (1997). Temporal data mining techniques for classification of time series data. Bachelors Thesis. Dept of Comp Sci, Australian National University.

[18] Rocchio, J. J., Jr.(1971).Relevance feedback in information retrieval: The Smart System - Experiments in Automatic Document Processing, ed. Salton, G., Prentice-Hall Inc., pp. 337-354.

[19] Rui,Y., Huang, T .S., Mehrotra, S. & Ortega, M. (1997). Automatic matching tool selection using relevance feedback in MARS. Proceedings of 2nd Int. Conf. On Visual Information Systems.

[20] Zebrowski, J,J. (1997). http://www.mpipks-resden.mpg.de/~ntpta/Data/Zebrowski-I

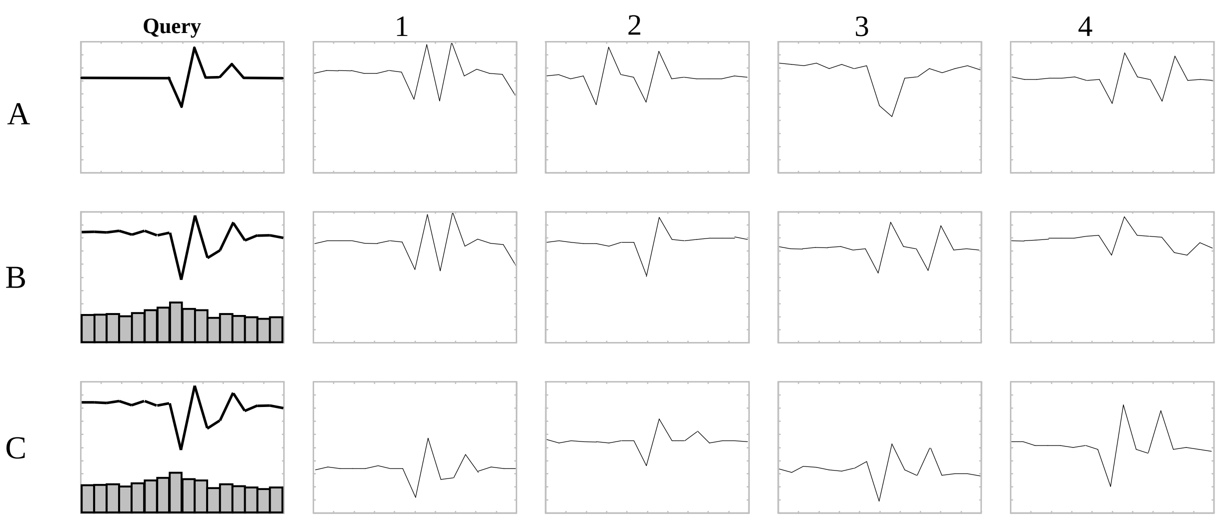

Figure 11: To illustrate the utility using subjective distance measures we did the following. We built a query and preformed an initial search using Euclidean distance (A). We examined and rated the first 10 sequences returned, and repeated the search in two different ways. Using the query update rule discussed in section 3, *without* using the subjective distance measures (B). Using the SDR algorithm, incorporating subjective distance measures (C)