All Common Subsequences

Hui Wang
School of Computing and Mathematics
University of Ulster, Northern Ireland, UK
h.wang@ulster.ac.uk

Abstract

Time series data abounds in real world problems. Measuring the similarity of time series is a key to solving these problems. One state of the art measure is the *longest common subsequence*. This measure advocates using the length of the longest common subsequence as an indication of similarity between sequences, but ignores information contained in the second, third, ..., longest subsequences.

In order to capture the common information in sequences maximally we propose a novel measure of sequence similarity – the number of *all common subsequences*. We show that this measure satisfies the common properties of similarity functions. Calculating this measure is not trivial as a brute force approach is exponential in time. We present a novel dynamic programming algorithm to calculate this number in polynomial time. We also suggest a different way of extending a class of such measures to multidimensional, real-valued time series, in the spirit of *probabilistic metric spaces*.

We conducted an experimental study on the new similarity measure and the extension method for classification. It was found that both the new similarity and the extension method are consistently competitive.

1 Introduction

A time series is a sequence [1] of symbols or real numbers, representing the measurements of a variable at, usually, equal time intervals [Gunopulos and Das, 2000]. When more than one variable are involved we get a *multidimensional time series*. Examples of time series include: stock price movements, volume of sales over time, daily temperature readings, ECG signals, and gene expressions. In tasks involving time series we need to consider the issues of indexing and searching [Agrawal *et al.*, 1995; Goldin *et al.*, 2004] and similarity measuring [Gunopulos and Das, 2000] – the latter being the focus of this paper. One state of the art measure is the *longest common subsequence* (LCS) [Hirschberg, 1977; Bergroth *et al.*, 2000].

A subsequence is obtained from a sequence by removing 0 or more symbols, and a common subsequence of a set of sequences is a subsequence of every sequence in the set. LCS advocates the use of the longest common subsequence as an indication of the similarity relationship between sequences. However the *common information* shared between sequences is more than that contained in the longest common subsequence. The second longest, third longest and, in general, all common subsequences contain some common information to a certain degree. The use of all possible common information between sequences is intuitively appealing and may provide a fuller picture about the similarity relationship between the sequences. How to measure all such common information between two sequences?

The above question can be answered by addressing a new problem, *all common subsequences* (ACS) – counting the number of all common subsequences. We expect that two sequences are more similar to each other if they have more common subsequences. ACS is conceptually novel as far as we know. To motivate ACS, we consider three sequences $\{\alpha, \beta, \gamma\} = \{cbabca, bcabac, abcade\}$. The set of all common subsequences of α and β is (ignoring the empty sequence): $\{a$, aa, ab, aba, abc, ac, b, ba, baa, bab, $baba$, $babc$, bac, bb, bba, bbc, bc, bca, c, ca, caa, cab, $caba$, $cabc$, cac, cb, cba, $cbac$, cbc, $cc\}$. The set of all common subsequences of α and γ is: $\{a$, aa, ab, aba, abc, $abca$, ac, aca, b, ba, bc, bca, c, $ca\}$. Clearly $lcs(\alpha, \beta) = lcs(\alpha, \gamma) = 4$. However $acs(\alpha, \beta) = 31$, $acs(\alpha, \gamma) = 15$ [2], suggesting there is a relatively higher degree of commonality in α and β than in α and γ. However this difference can not be revealed by LCS.

Solving the ACS problem is not trivial. A brute-force approach has an exponential time complexity as we need to verify all subsequences. We take a dynamic programming approach, and develop a novel algorithm to calculate ACS between two sequences in polynomial time.

In order to evaluate ACS as a similarity measure we need to extend ACS, as well as LCS, to real-valued, multidimensional time series as both LCS and ACS are defined in terms

[1] Throughout this paper we use the terms "time series" and "sequence" interchangeably.

[2] We use a convention here: when talking about a concept like LCS or ACS, we use capital letters. When talking about the same concept as a measure function, we use lower case letters.

of symbolic sequences. We suggest a novel method for this purpose, which requires determination of equality/inequality. This is done in the spirit of *probabilistic metric spaces*, where we consider the probability that two data points are the same. This extension method is different from the threshold-based extension method [Vlachos *et al.*, 2003], where two data points in a multidimensional time series are deemed equal if they are close enough (judged by a threshold) in every dimension. Under this extension the ACS similarity is evaluated in an experimental study with some real-valued, multidimensional time series. The study shows that ACS is favorably comparable to LCS in general, and offers some property that is absent from LCS. Additionally the extension method is sound and competitive consistently.

The rest of the paper is organized as follows. The *longest common subsequence* concept is briefly reviewed in next section as it inspired the work presented here. Our *all common subsequence* concept is presented in Section 3, along with a dynamic programming algorithm to calculate ACS. A method is presented in Section 4, which can be used to extend ACS as well as LCS to multidimensional, real-valued time series. Evaluation is discussed in Section 5 and related work is discussed in Section 6, followed by some concluding remarks.

2 Longest common subsequence

The *longest common subsequence*, as a computer science problem, is concerned with the search for an efficient method of finding the longest common subsequence of two or more sequences. A milestone in the study of LCS is the development of dynamic programming algorithms [Hirschberg, 1977; Bergroth *et al.*, 2000], which calculates LCS in polynomial time.

The dynamic programming algorithm for calculating LCS was originally designed to work for one dimensional sequences of discrete values. It was extended in [Vlachos *et al.*, 2003] to work for multidimensional, real-valued sequences as follows:

Let A and B be two d-dimensional sequences with length n and m respectively: $A = (a_1, \cdots, a_n)$ and $B = (b_1, \cdots, b_m)$, where $a_i = (a_{i,1}, \cdots, a_{i,d})$, and $b_j = (b_{j,1}, \cdots, b_{j,d})$. Let $Head(A) = (a_1, \cdots, a_{n-1})$ and $Head(B) = (b_1, \cdots, b_{m-1})$. Then the LCS similarity between A and B is [Vlachos *et al.*, 2003]:

(1)

$$lcs_{\delta,\epsilon}(A,B) = \begin{cases} 0, & \text{if } A \text{ or } B \text{ is empty} \\ 1 + lcs_{\delta,\epsilon}(Head(A), Head(B)), & \\ \quad \text{if } |a_{n,i} - b_{m,i}| < \epsilon \text{ for } 1 \le i \le d & \\ \quad \text{and } |n-m| \le \delta & \\ max(lcs_{\delta,\epsilon}(Head(A), B), & \\ \quad lcs_{\delta,\epsilon}(A, Head(B))), \text{otherwise} & \end{cases}$$

where δ is an integer that controls matching in time, and $0 < \epsilon < 1$ is a real number that controls matching in space. The LCS algorithm, thus restricted, has a computational complexity of $O(\delta(n+m))$ if we only allow a matching window δ in time [Das *et al.*, 1997]. The advantage of LCS is that it supports elastic and imprecise matches.

3 All common subsequence

Here we consider sequences of discrete symbols. In the next section we will consider sequences of multidimensional, real-valued sequences.

Let $\mathcal{A}$ be a finite set of symbols. A sequence α is an ordered set $\{s_i\}_0^n = \{s_0, s_1, \cdots, s_n\}$, where $s_i \in \mathcal{A}$. For simplicity we also write the above sequence as $s_0 s_1 \cdots s_n$. The *empty sequence* is an empty set and is denoted by ϵ. The data space V is then the set of all sequences.

Consider two symbols a, b in a sequence α. Looking from left to right, if a comes before b we write $a \le_\alpha b$. Consider two sequences α and β. If α can be obtained by deleting some symbols from β, we say α is a *subsequence* of β (or, equivalently, β is a supersequence of α) and we write $\alpha \preceq \beta$. It is obvious that the $\preceq$ relation is reflexive and transitive.

A common subsequence of two sequences is a subsequence of both sequences. For two identical sequences, they have the maximal number of common subsequences relative to their length. For two different sequences, they have the minimal number of common subsequences. Therefore the number of common subsequences is an indication of similarity. Whether or not it is a good similarity indicator is unclear, but will be evaluated in a later section. We call this *all common subsequences* (ACS) similarity. For two sequences α and β, we write $acs(\alpha, \beta)$ for the ACS similarity of the two sequences.

Lemma 1. *The ACS similarity has the following properties:*

- $acs(\alpha, \beta) \ge 0$.
- $acs(\alpha, \beta) \le acs(\alpha, \alpha)$ *and* $acs(\alpha, \beta) \le acs(\beta, \beta)$.
- *ACS is symmetric, i.e.,* $acs(\alpha, \beta) = acs(\beta, \alpha)$*;*
- $acs(\alpha, \beta) = acs(\alpha^r, \beta^r)$*, where* α^r *is the reverse sequence of* α.

The first two properties are common to all similarity measures [Osborne and Bridge, 1997], but the last two are additional. The proof of the lemma follows directly from the definition of ACS. The following lemma shows how many subsequences there are in one sequence, which can be proved by mathematical induction.

Lemma 2. *For a sequence of length* m*, the number of its subsequences is* $\sum_{i=0}^{m} \binom{i}{m} = 2^m$*, which includes the empty sequence.*

The relationship between sequence and subsequence is shown in the following lemmas. Let $N(\alpha|\gamma)$ be the set of subsequences of α that are supersequences of γ, and $N(\alpha)$ be the set of all subsequences of α.

Lemma 3. *Consider* $\gamma_1, \gamma_2 \preceq \alpha$*. Then*

$$\gamma_1 \preceq \gamma_2 \Longleftrightarrow N(\alpha|\gamma_2) \subseteq N(\alpha|\gamma_1)$$

Proof. $\Rightarrow$: Suppose $\gamma_1 \preceq \gamma_2$. Consider $x \in N(\alpha|\gamma_2)$. Then $\gamma_2 \preceq x$. Since the relation $\preceq$ is clearly transitive we have $\gamma_1 \preceq x$. Therefore $x \in N(\alpha|\gamma_1)$. We then have $N(\alpha|\gamma_2) \subseteq N(\alpha|\gamma_1)$.
$\Leftarrow$: Suppose $N(\alpha|\gamma_2) \subseteq N(\alpha|\gamma_1)$. By definition we have $\gamma_2 \preceq x$ for all $x \in N(\alpha|\gamma_2)$, and $\gamma_1 \preceq y$ for all $y \in N(\alpha|\gamma_1)$. To show that $\gamma_1 \preceq \gamma_2$, we assume it is not true. Then $\gamma_2 \notin N(\alpha|\gamma_1)$, but $\gamma_2 \in N(\alpha|\gamma_2)$. Therefore it is not possible $N(\alpha|\gamma_2) \subseteq N(\alpha|\gamma_1)$, contradicting the assumption. □

Lemma 4. *$\alpha \preceq \beta$ if and only if $N(\alpha) \subseteq N(\beta)$.*

Proof. $\Rightarrow$: Suppose $\alpha \preceq \beta$. Consider $x \in N(\alpha)$. Then x is a subsequence of α, i.e., $x \preceq \alpha$. By the transitivity of $\preceq$ we have $x \preceq \beta$, so x is a subsequence of β. Therefore we have $x \in N(\beta)$. This proves $N(\alpha) \subseteq N(\beta)$.

$\Leftarrow$: Suppose $N(\alpha) \subseteq N(\beta)$. To show that $\alpha \preceq \beta$ we assume it is not true. Then $\alpha \notin N(\beta)$. However $\alpha \in N(\alpha)$. Therefore it is not possible that $N(\alpha) \subseteq N(\beta)$, contradicting the assumption. This proves $\alpha \preceq \beta$. □

3.1 Calculating the number of all common subsequences

Now we discuss how to calculate the number of all common subsequences, $acs(A, B)$, of any two sequences A and B. Since the number of subsequences of a sequence is exponential in its length, a straightforward brute force approach is clearly not feasible. We turn our attention to dynamic programming.

"*Dynamic programming can be thought of as being the reverse of recursion. Recursion is a top-down mechanism – we take a problem, split it up, and solve the smaller problems that are created. Dynamic programming is a bottom-up mechanism – we solve all possible small problems and then combine them to obtain solutions for bigger problems. The reason that this may be better is that, using recursion, it is possible that we may solve a small subproblem many times. Using dynamic programming, we solve it once.*" [Hirschberg, 2005]

Theorem 1. *Consider two sequences, $A = (A_1, \cdots, A_m)$ and $B = (B_1, \cdots, B_n)$. Let $N(i, j)$ be the number of common subsequences of $(A_1, \cdots, A_i)$ and $(B_1, \cdots, B_j)$, i.e., the prefixes of sequences A and B of lengths i and j. Then*

$$N(i,j) = N(i-1, j-1) \times 2, \text{ if } A_i = B_j$$
$$N(i,j) = N(i-1, j) + N(i, j-1) - N(i-1, j-1), \quad \text{if } A_i \neq B_j$$

Consequently $acs(A, B) = N(m, n)$.

Proof. Let $S(i-1, j-1)$ be the set of all common subsequences between $(A_1, \cdots, A_{i-1})$ and $(B_1, \cdots, B_{j-1})$. So $N(i-1, j-1) = |S(i-1, j-1)|$.

If $A_i = B_j$, then $S(i,j) = S(i-1, j-1) \cup S(i-1, j-1)A_i$. Here $S(i-1, j-1)A_i = \{xA_i : x \in S(i-1, j-1)\}$. Therefore $N(i, j) = N(i-1, j-1) \times 2$.

If $A_i \neq B_j$, then some new common subsequences may be added to $S(i, j-1)$ or $S(i-1, j)$ on top of $S(i-1, j-1)$. Therefore $S(i,j) = S(i, j-1) \cup S(i-1, j) - S(i-1, j-1)$. Consequently we have $N(i,j) = N(i, j-1) + N(i-1, j) - N(i-1, j-1)$. □

Algorithm 3.1: ***calACS*** – calculate all common subsequence

```
Input: Two sequences $A$ and $B$.
Output: The number of all common subsequences
        $acs(A, B)$.
1  begin
2      for $i = 0$ to $|A|$ do $N(i, 0) = 1$
3      for $j = 0$ to $|B|$ do $N(0, j) = 1$
4      for $i = 1$ to $|A|$ do
5          for $j = 1$ to $|B|$ do
6              if $A_i = B_j$ then
                   $N(i,j) = N(i-1, j-1) \times 2$
7              else $N(i, j) =$
                   $N(i-1, j) + N(i, j-1) - N(i-1, j-1)$
8          end
9      end
10     $acs(A, B) = N(|A|, |B|)$
11 end
```

Note that the δ parameter as discussed in Section 2 can also be used together with the above ***calACS*** algorithm.

The $acs(A, B)$ is unbounded. We can normalize it by the number of all subsequences, which can be calculated as in Lemma 2.

4 Extension of LCS and ACS

The ACS is presented in the previous section on the assumption that a sequence is an ordered set of symbols or discrete values. When a sequence is real-valued or multidimensional, the dynamic programming algorithm for calculating ACS must be extended.

This is also the case for LCS, but a method is presented in [Vlachos *et al.*, 2003] to extend LCS. This extension method can be directly used for ACS but, as discussed in Section 2, the threshold ϵ is hard to choose and is usually different for different data. This is not desirable [Keogh *et al.*, 2004].

Here we present a different solution for the problem of extending LCS and ACS, possibly some other similarity/distance measures as well, where we need to decide if two points are equal (equality checking). Our solution is in the spirit of Probabilistic Metric Spaces [Schweizer and Sklar, 1983], where the line between 'equal' and 'not equal' is probabilistic. Consider two sequences $A = (A_1, \cdots, A_m)$ and $B = (B_1, \cdots, B_n)$. For any pair A_i and B_j we need to decide, at some point in the LCS and ACS algorithms, if they are equal. The answer to this question may not be deterministic, especially when real numbers are involved. In general the question can be answered by measuring the probability, $Prob(A_i = B_j)$, that they are equal [3].

A general template for both LCS and ACS is the following. The distance/similarity between a sequence $(A_1, \cdots, A_i)$ and another sequence $(B_1, \cdots, B_j)$ is

$$DS(i,j) = \begin{cases} DS_1 & \text{if } A_i = B_j \\ DS_0 & \text{otherwise} \end{cases} \tag{2}$$

[3]Probabilistic Metric Spaces are concerned with a more general question: what is the probability that the distance between A_i and B_j is less than a given threshold δ?

In the case of LCS, $DS_1 = 1 + DS(i-1, j-1)$, and $DS_0 = max(DS(i-1, j), DS(i, j-1))$. In the case of ACS, $DS_1 = DS(i-1, j-1) \times 2$, and $DS_0 = DS(i, j-1) + DS(i-1, j) - DS(i-1, j-1)$.

We propose to use the following template instead:

(3)
$$DS(i, j) = DS_1 \times Prob(A_i = B_j) + D_0 \times Prob(A_i \neq B_j)$$

where $Prob(A_i \neq B_j) = 1 - Prob(A_i = B_j)$. This approach avoids using the threshold ϵ.

The question now is how to determine $Prob(A_i = B_j)$. In the case of symbols or discrete values, we can simply and reasonably take $Prob(A_i = B_j) = 1$ if $A_i = B_j$, and $Prob(A_i = B_j) = 0$ otherwise. In the case of real numbers, either 1-dimensional or multidimensional, we can estimate $Prob(A_i = B_j)$ by the distance between A_i and B_j normalized to the range of $[0, 1]$. Note that the distance can be the Euclidean distance or other distance. This is a simplistic approach, but a more involved approach is beyond the scope of this paper.

5 Experimental evaluation

We conducted an experimental study to evaluate the ACS similarity measure along with the extension method. In this study we demonstrate the utility of ACS for classification in an experiment on some public datasets. We consider the following multidimensional, real-valued time series: ECG2c, ECG3c and Auslan.

The ECG datasets are from the BIDMC Congestive Heart Failure Database. The original database contains two ECG signals, but Keogh et al [Keogh *et al.*, 2004] separated the signals and create two datasets (ECG2c – *ecg_clustering.txt* and ECG3c – *ecg_clustering_3class.txt*) of one-dimensional time series. Each instance of 3,200 contiguous data points (about 20 heartbeats) of each signal is randomly extracted from each long ECG signals of each patient (class). Twenty instances are extracted from each class, resulting in eighty total instances for each dataset.

The Australian Sign Language (Auslan) data is available from [Keogh and Folias, 2002]. We consider a clean, preprocessed version of Auslan. There are 10 classes: 'come', 'girl', 'man', 'maybe', 'mine', 'name', 'read', 'right', 'science', 'thank'. There are 20 examples in each class, and all classes come from one signer. All time series are of the same length (by interpolation).

Auslan is an interesting and challenging dataset as "it is multivariate, and without exploiting the multivariate property it is hard to do much better than random guessing." [Keogh and Folias, 2002] We use the following similarity/distance measures on the datasets: Euclidean (euc), Longest Common Subsequence (lcs) and All Common Subsequences (acs). Because all of the datasets are real-valued and some are multidimensional, these distance/similarity measures need to be extended. We consider the following combinations:

- euc: the standard Euclidean distance, used in our extended similarity measures (lcs_w_euc, acs_w_euc).
- ncm: the NCM similarity (see [Wang and Dubitzky, 2005]), used in our extended similarity measures (lcs_w_ncm, acs_w_ncm).
- lcs_vhgk: the extended LCS presented in [Vlachos *et al.*, 2003] (see Section 2).
- acs_vhgk: ACS extended in a way similar to lcs_vhgk.
- lcs_w_euc: LCS extended as in Eq.3, where the probability is estimated (simplistically) as the normalized Euclidean distance.
- acs_w_euc: ACS extended as in Eq.3, where the probability is estimated as the normalized Euclidean distance.
- lcs_w_ncm: where the probability is the normalized NCM.
- acs_w_ncm: where the probability is the normalized NCM.

We used the leave-one-out evaluation strategy in the k-nearest neighbors (kNN) classification framework, meaning that we classify every time series based on all other time series and then we measure the classification accuracy. There are two parameters in the experiment: δ and ϵ, which are required by acs and lcs. For simplicity we set $\delta = 3$ for all measures, without loss of generality. For ϵ we test-experimented on samples of the data with various values and set it to a value that gives the best overall result, that is, $\epsilon = 0.13$. Additionally we set parameter k (required for kNN) to $1, 4, 7, 10, 13, 16, 19$.

Experimental results are presented in Tables 1, 2, 3, and 4, from which we observe the following:

- when used as part of acs or lcs, ncm gives consistently better results than euc: $acs\_w\_ncm > acs\_w\_euc$ and $lcs\_w\_ncm > lcs\_w\_euc$.
- acs has an edge over lcs, especially for relatively large k: $acs\_w\_euc > lcs\_w\_euc$ and $acs\_w\_ncm > lcs\_w\_ncm$.
- The extension method presented in this paper (see Section 4) has an edge over that suggested in [Vlachos *et al.*, 2003], especially for small k values: $lcs\_w\_euc > lcs\_vhgk$.

We believe that the superior performance of ACS over LCS is due to the fact that ACS is able to capture more common information in a pair of sequences than LCS.

6 Related work

Paper [Lin *et al.*, 2005] presents a novel measure of dissimilarity between two time sequences – the degree of difference for pattern p, weighted by the confidence of the pattern, in two time series. Given two time series, s_1 and s_2, they are first of all represented as symbolic SAX sequences. A pattern is a subsequence in the SAX representation of a time series. Without weighting, the dissimilarity is

$$DSim(s_1, s_2) \propto \sum_p \frac{|f_1(p) - f_2(p)|}{max(f_1(p), f_2(p))}$$

where $f_i(p)$ is the frequency of pattern p in sequence s_i.

Paper [Yang *et al.*, 2003] introduces a new measure of similarity/distance between symbolic sequences. The measure is based on counting the occurrences of m-bit (i.e., consisting of m symbols) words (i.e., subsequences) in a symbolic sequence. Without weighting by the probability of each word in a sequence, the distance between two sequences s_1 and s_2 is defined as follows:

$$D_m(s_1, s_2) \propto \sum_{k=1}^{2^m} |R_1(w_k) - R_2(w_k)|$$

where $R_i(w_k)$ is the rank of subsequence w_k [4] in sequence s_i.

$DSim$ and D_m are both related to ACS in the sense they consider all common subsequences in s_1 and s_2. However there are significant differences:

- Conceptually both $DSim$ and D_m measure the average (weighted) difference of the frequency of a subsequence p in s_1 and the frequency of p in s_2, while ACS counts the number of common subsequences of two sequences.
- They all need to look through a large, exponential search space for subsequences, but ACS has a dynamic programming algorithm to calculate the similarity in polynomial time.
- ACS can be applied to sequences of real numbers without the need of transformation, while D_m can not. To use $DSim$ for sequences of real numbers, the symbol SAX representation (a type of transformation) is needed.

7 Conclusion

This paper presents a conceptually novel similarity measure for time series data – *all common subsequences* or ACS. It is intended to maximally capture the common information shared by two time series. A novel dynamic programming algorithm is presented to calculate the number of all common subsequences.

To use ACS and other equality-checking based sequence similarity measures (e.g., LCS) to handle multidimensional, read-valued time series, we need to extend these measures. This paper presents an extension method, which is inspired by the theory of *probabilistic metric spaces*. We seek to use the probability that two points are equal in calculating the similarity when a time series is real-valued or multidimensional.

We conducted an experimental study to evaluate the new similarity measure and the extension method. Our experimental results show that ACS is favorably comparable to LCS, and it displays a distinctive feature – stable performance as more data are considered for decision (i.e., as k gets larger), making ACS suitable for those tasks whose aim is to find a set of ranked sequences (e.g., information retrieval). The results also show that the extension method presented in this paper works well, and appears better than that discussed in [Vlachos *et al.*, 2003].

[4]The frequency of occurrence is calculated for every words, and is used as the basis for ranking all the words.

We feel that the way in which we estimate the probability of two data points being equal is a bit crude. In our future work we intend to develop more involved methods in order to make it more effective and efficient.

References

[Agrawal *et al.*, 1995] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. VLDB95*, 1995.

[Bergroth *et al.*, 2000] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *SPIRE'00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval*, page 39. IEEE Computer Society, 2000.

[Das *et al.*, 1997] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proc. of PKDD97: Principles of Data Mining and Knowledge Discovery in Databases*, pages 88–100, 1997.

[Goldin *et al.*, 2004] Dina Goldin, Todd Millstein, and Ayferi Kutlu. Bounded similarity querying for time-series data, 2004.

[Gunopulos and Das, 2000] Dimitrios Gunopulos and Gautam Das. Time series similarity measures. 2000. Tutorial notes of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.

[Hirschberg, 1977] Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of ACM*, 24(4):664–675, 1977.

[Hirschberg, 2005] Dan Hirschberg. Dynamic programming, 2005. `http://www.ics.uci.edu/~dan/class/161/notes/6/Dynamic.html`.

[Keogh and Folias, 2002] E. Keogh and T. Folias. The UCR time series data mining archive, 2002. `http://www.cs.ucr.edu/~eamonn/TSDMA/index.html`.

[Keogh *et al.*, 2004] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proc. SIGKDD04*, 2004.

[Lin *et al.*, 2005] Jessica Lin, Eamonn Keogh, and Stefano Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Information Visualization*, 4(2):61–82, 2005.

[Osborne and Bridge, 1997] Hugh Osborne and Derek Bridge. Models of similarity for case-based reasoning. In *Procs. of the Interdisciplinary Workshop on Similarity and Categorisation*, pages 173–179, 1997.

[Schweizer and Sklar, 1983] B. Schweizer and A. Sklar. *Probabilistic metric spaces*. North-Holland, 1983.

[Vlachos *et al.*, 2003] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of SIGKDD03: ACM International Conference on Data Mining August 24-27, 2003, Washington, DC, USA*, 2003.

[Wang and Dubitzky, 2005] H. Wang and W. Dubitzky. A flexible and robust similarity measure based on contextual probability. In *Proceedings of IJCAI'05*, pages 27–32, 2005.

[Yang *et al.*, 2003] Albert C. C. Yang, Shu-Shya Hseu, Huey-Wen Yien, Ary L. Goldberger, and C. K. Peng. Linguistic analysis of the human heartbeat using frequency and rank order statistics. *Phsical Review Letters*, 90(10), 2003.

A Tables

k	1	4	7	10	13	16	19
acs_vhgk	97.5	97.5	97.5	97.5	97.5	97.5	97.5
lcs_vhgk	97.5	97.5	97.5	97.5	97.5	97.5	97.5
acs_w_euc	100	100	100	97.5	90.0	87.5	87.5
lcs_w_euc	100	100	100	95.0	87.5	85.0	85.0
acs_w_ncm	100	100	100	100	100	100	100
lcs_w_ncm	100	100	100	100	95.0	95.0	95.0

Table 1: Leave-one-out classification accuracy (%) on a sample ('come' and 'girl') of Auslan.

k	1	4	7	10	13	16	19
acs_vhgk	83.5	84.0	85.0	85.5	85.5	85.5	85.5
lcs_vhgk	83.5	85.0	84.5	85.0	85.0	84.5	84.0
acs_w_euc	90.5	90.0	89.0	86.5	85.5	84.5	84.5
lcs_w_euc	90.5	90.0	90.5	88.0	86.0	86.0	84.5
acs_w_ncm	93.0	93.5	93.5	93.0	93.0	93.5	93.0
lcs_w_ncm	94.5	94.0	94.0	93.0	92.0	90.5	90.5

Table 2: Leave-one-out classification accuracy (%) on all of Auslan.

k	1	4	7	10	13	16	19
acs_vhgk	90.0	90.0	90.0	90.0	90.0	90.0	90.0
lcs_vhgk	90.0	80.0	65.0	50.0	50.0	50.0	60.0
acs_w_euc	95.0	95.0	95.0	95.0	95.0	95.0	95.0
lcs_w_euc	95.0	95.0	80.0	75.0	70.0	65.0	60.0
acs_w_ncm	95.0	95.0	95.0	95.0	95.0	95.0	95.0
lcs_w_ncm	95.0	95.0	80.0	75.0	65.0	65.0	65.0

Table 3: Leave-one-out classification accuracy (%) on Ecg2c.

k	1	4	7	10	13	16	19
acs_vhgk	75.0	75.0	75.0	75.0	75.0	75.0	75.0
lcs_vhgk	80.6	80.6	80.6	75.0	75.0	75.0	75.0
acs_w_euc	91.7	91.7	91.7	91.7	91.7	91.7	91.7
lcs_w_euc	91.7	88.9	86.1	80.6	77.8	72.2	69.4
acs_w_ncm	91.7	91.7	91.7	88.9	86.1	86.1	86.1
lcs_w_ncm	91.7	88.9	83.3	80.6	77.8	72.2	72.2

Table 4: Leave-one-out classification accuracy (%) on Ecg3c.

B Acknowledgements

Thanks to Eamonn Keogh for providing all of the datasets used in this paper, for his comments on the paper, and for pointing out the work in [Lin *et al.*, 2005; Yang *et al.*, 2003].