

# A Complexity-Invariant Distance Measure for Time Series

Gustavo E.A.P.A. Batista<sup>1,2</sup>

Xiaoyue Wang<sup>1</sup>

Eamonn J. Keogh<sup>1</sup>

<sup>1</sup>University of California, Riverside <sup>2</sup>University of São Paulo - USP

gbatista@icmc.usp.br xwang@cs.ucr.edu eamonn@cs.ucr.edu

## ABSTRACT

The ubiquity of time series data across almost all human endeavors has produced a great interest in time series data mining in the last decade. While there is a plethora of classification algorithms that can be applied to time series, all of the current empirical evidence suggests that simple nearest neighbor classification is exceptionally difficult to beat. The choice of distance measure used by the nearest neighbor algorithm depends on the invariances required by the domain. For example, motion capture data typically requires invariance to warping.

In this work we make a surprising claim. There is an invariance that the community has missed, *complexity invariance*. Intuitively, the problem is that in many domains the different classes may have different complexities, and pairs of complex objects, even those which subjectively may seem very similar to the human eye, tend to be further apart under current distance measures than pairs of simple objects. This fact introduces errors in nearest neighbor classification, where complex objects are incorrectly assigned to a simpler class.

We introduce the first complexity-invariant distance measure for time series, and show that it generally produces significant improvements in classification accuracy. We further show that this improvement does not compromise efficiency, since we can lower bound the measure and use a modification of triangular inequality, thus making use of most existing indexing and data mining algorithms. We evaluate our ideas with the largest and most comprehensive set of time series classification experiments ever attempted, and show that complexity-invariant distance measures can produce improvements in accuracy in the vast majority of cases.

## Keywords

Time Series, Classification, Similarity Measures, Complexity

## 1. INTRODUCTION

Time series data occur in almost all domains, and this fact has created a great interest in time series data mining. There is a plethora of classification algorithms that can be applied to time series; however, all of the current empirical evidence suggests that simple nearest neighbor classification is very difficult to beat [5][14].

This only leaves open the question of *which* distance measure to use. The choice of distance measure depends on the domain. In particular, it depends on the invariances required by the domain. For example, motion capture data typically requires invariance to “warping” (local non-linear accelerations) [13], and gene expression data typically requires invariance to uniform scaling (linear “stretching”) [11]. With this in mind, the last decade has seen the introduction of hundreds of techniques designed to efficiently measure the similarity between time series with invariance to (various combinations of) the distortions of warping [13], uniform scaling [11], offset [7], amplitude scaling, phase [13], occlusions, uncertainty and wandering baseline.

Surprisingly, there is an invariance that the community has missed, *complexity invariance*. We defer a detailed discussion of complexity until later in this work, but for the moment the reader’s intuitive idea of “*having more peaks, valleys and features*” will suffice.

The problem lies in the fact that in many domains the different classes may have different complexities, and pairs of complex objects, even those which subjectively may seem very similar, tend to be further apart under current distance measures than pairs of simple objects. This fact introduces errors in nearest neighbor classification, because complex objects are assigned to a simpler class.

In this work we introduce the first complexity-invariant distance measure for time series, and show that it generally produces significant improvements in classification accuracy. Our complexity-invariant distance measure is simple, parameter-free, and increases the time complexity only by a barely perceptible amount.

We further show that this improvement does not compromise the efficiency of algorithms that make frequent calls to a distance measure (classification [5], clustering [6], motif discovery [17] and outlier detection [20][21]), since we can lower bound the measure and use a minor modification of triangular inequality, and thus avail of all existing indexing and data mining techniques.

It is critical to note that the problem we have observed is not solved or mitigated by generalizing from one nearest neighbor to  $k$ -nearest neighbors, or by smoothing the data, or using DTW, etc. We will

present extensive empirical evidence that our improvements in accuracy are valid, significant and result from the reason we claim.

The rest of this paper is organized as follows. In Section 2 we review the current set of known invariances, and more general related work. In Section 3 we present a distance measure that is invariant to the complexity of time series. In Section 4 we evaluate our ideas with a large and comprehensive set of time series classification experiments and show that complexity-invariant distance measures can produce improvements in accuracy in the vast majority of cases. In Section 5 we discuss some important properties of our proposed distance that allow it to be indexed by virtually all existing indexing schemes based on lower bounding and triangular inequality. Finally, in Section 6 we offer conclusions and suggest areas for future work.

## 2. BACKGROUND / RELATED WORK

We begin by introducing Euclidean distance, and use this as a starting point to consider other distance measures in the next section.

Suppose we have two time series,  $Q$  and  $C$ , of length  $n$ .

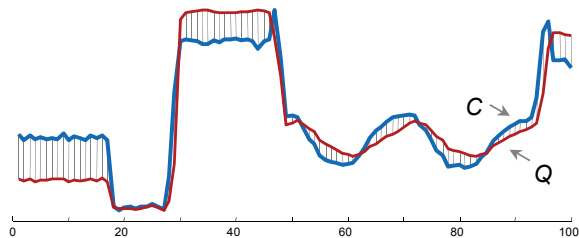
$$Q = q_1, q_2, \dots, q_i, \dots, q_n$$

$$C = c_1, c_2, \dots, c_i, \dots, c_n$$

If we wish to compare two time series, we can use the ubiquitous Euclidean distance:

$$ED(Q, C) \equiv \sqrt{\sum_{i=1}^n (q_i - c_i)^2}$$

This distance measure is visualized in Figure 1.



**Figure 1: The Euclidean distance between two time series can be visualized as the square root of the sum of the squared length of the vertical hatch lines**

While Euclidean distance is a simple measure, it is competitive for many problems [5]. Nevertheless, in many domains the data is *distorted* in some way, and either the distortion must be removed before using Euclidean distance, or a more robust measure must be used<sup>1</sup>. We will now review all known distortions and the techniques for achieving invariance to them.

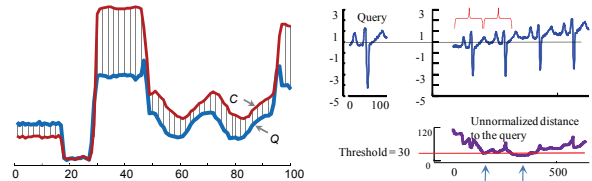
<sup>1</sup> In practice these two options can be logically equivalent. For example, DTW can be seen as a more robust distance measure, or it can be seen as using the Euclidean distance after a dynamic programming algorithm has removed the warping.

## 2.1 A Review of All Known Invariances

For any given domain the set of required invariances is domain dependent, and include:

**Amplitude Invariance/Offset Invariance:** If we try to compare two time series measured on different scales, say Celsius and Fahrenheit, they will not match well, even if they have similar shapes. To measure the true underlying similarity we must first make their amplitudes the same. As shown in Figure 2.*left*, the example of Euclidean distance shown in Figure 1 *already* had this normalization, and the original data has a significant difference in amplitude.

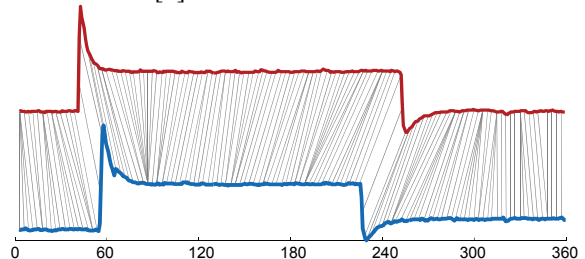
Similarly, even if two time series have identical amplitudes, they may have different offsets (different mean values). As shown in Figure 2.*right*, even a small change in offset rapidly dominates the Euclidean distance, leading to false negatives; i.e., missed heartbeats by a heartbeat detector.



**Figure 2: *left*) If compared before amplitude scaling, these two time series appear *very* different. *right*) When matching a heartbeat query against a stream, the first two match well, but subsequently the drifting offset means the remaining heartbeats are not discovered**

A classic example of where these invariances are critical is gait recognition from video, where zoom and pan/tilt correspond to amplitude and offset, respectively. Both these invariances can be trivially achieved by z-normalizing the data [7].

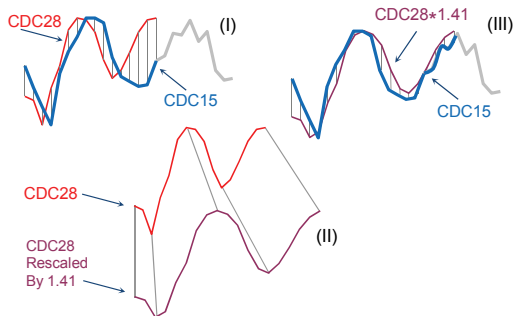
**Local Scaling (“warping”) Invariance:** This invariance is necessary in almost all biological signals, including gait, motion capture, handwriting and ECGs. Figure 3 shows an example of two insect behaviors which match only when one is locally warped to align with the other [1].



**Figure 3: Two time series of insect behavior matched with invariance to warping. The alignment was calculated by the DTW algorithm**

Given the ubiquity of the domains in which this invariance is required, there are hundreds of papers on the topic. However, recent empirical evidence strongly suggests that a forty-year-old technique, Dynamic Time Warping (DTW), works exceptionally well [5].

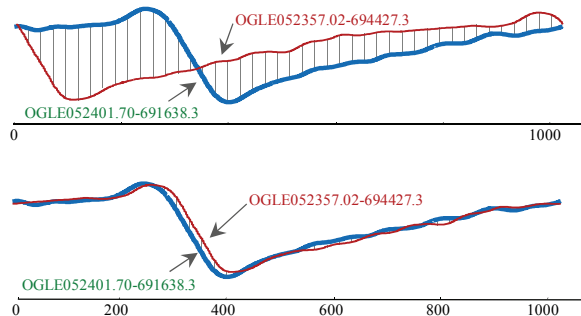
**Uniform Scaling Invariance:** In contrast to the *localized* scaling that DTW deals with, in many data sets we must also deal with *global* scaling. For example, Figure 4 shows two yeast cell-cycle gene expression time series which are known to be related [16]. If we try to match the shorter one against the prefix of the longer one, it matches poorly. However, if we globally stretch it by a factor of 1.41, it becomes a much better match.



**Figure 4: (I) A full gene expression, CDC28, matches poorly to the prefix of a related gene, CDC15. (II) If we rescale it by a factor of 1.41, it matches CDC15 much more closely (III)**

The main difficulty in creating uniform scaling invariance is that we typically do not know the scaling factor ahead of time, and are thus condemned to testing all possibilities within a given range [11].

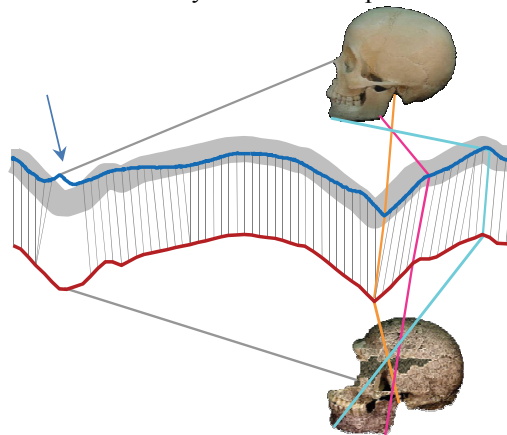
**Phase Invariance:** This form of invariance is important when matching periodic time series such as star-light curves [13], gait, heartbeats, etc. It is also important when matching two-dimensional shapes which have been converted to one-dimensional “time” series, a representational trick which has gained popularity in recent years (cf. Figure 6) [13].



**Figure 5: top) Two star light curves are obviously out of phase. bottom) By holding one time series fixed, and testing all circular shifts of the other, we can achieve phase invariance**

Several authors have suggested achieving this invariance by finding a cardinal alignment to which all time series are aligned. However, recent evidence suggests that this may be very brittle [23], and the only currently known way to guarantee phase invariance is to test all possible alignments, as shown in Figure 5.

**Occlusion Invariance:** This form of invariance occurs in domains where a small subsequence of a time series may be missing. As a simple example, imagine that we want to build a robust index for sign language motion capture data, such that the query “one small step for a man” will find “one small step for man,” in spite of the fact that one sequence is missing the “a.” Figure 6 shows a more visually intuitive example.



**Figure 6: Occlusion invariance can be achieved by selectively declining to match subsections of a time series. In this case it is robust to the missing nose region of the ancient skull (bottom)**

## 2.2 On Multiple Invariances

We conclude with a simple experiment to demonstrate what the reader will have surely anticipated: that some problems require *multiple* invariances.

We created a 3-class star light curve data set with a test set of 128 examples (two of which are shown in Figure 5), and a training set of 1024 objects<sup>2</sup>.

While the original light curves have periods that range from hours to weeks, they are normalized to have a standard length of 1,024 to give them **uniform scaling** invariance. Furthermore, since it is the *relative*, not *absolute*, changes in apparent brightness that matter, the data is z-normalized to adjust the **amplitude** and **offset**. The data is then universally phased using a state-of-the-art domain aware algorithm to achieve **phase invariance** [20].

We measured the accuracy of this data set using 1-nearest neighbor classifier with Euclidean distance, achieving a respectable 80.47%. We then checked to see if using DTW to achieve **local scaling** invariance

<sup>2</sup> This experiment, like all others in this work, is 100% reproducible; see Section 4 for our experimental philosophy.

would help, and indeed our accuracy jumped to 86.72%.

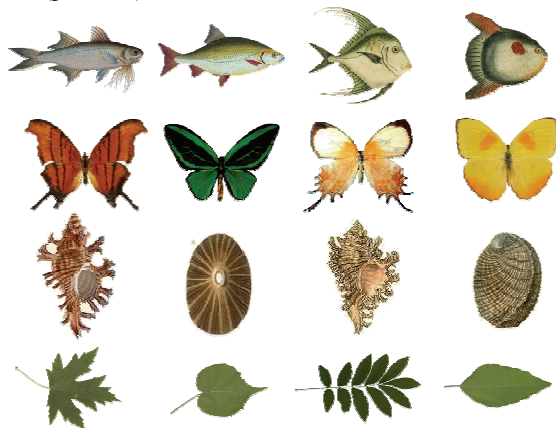
However, rather than stopping here, we decided to test the universal phasing assumption. Suppose we ignored it and tested DTW for *all* possible alignments/shifts. After testing the **phase**-invariant version of DTW [13], we found that the accuracy increased to an impressive 91.4%. Clearly, the current universal phasing algorithm does not produce perfect alignments.

Finally, we note the obvious, that in some cases invariances can *decrease* accuracy. An obvious example is in the classification of shapes converted to time series (as in Figure 6). If we wanted to discriminate between the shapes 'p' and 'd', this would be trivial, but adding phase (rotation) invariance would make it impossible.

The complexity invariance that we are proposing in this work is not a special case of any of the above or any combination of the above, but rather a new invariance whose need has escaped the attention of the community.

### 3. COMPLEXITY-INVARIANT DISTANCE FOR TIME SERIES

We are finally in a position to introduce the core contribution of this work. As we have seen, the research community has proposed a diverse set of invariances for time series in the last two decades. However, there is one invariance that has been missed, *complexity invariance* (for the moment, the reader's natural intuition of complex as having many peaks and values will suffice.) In many (perhaps *most*) domains, different classes may have widely varying complexities. We can most readily see this on time series derived from shapes (as in Figure 6). For example, as we show in Figure 7, leaves range in complexity from a simple pointed ellipse (ovate), shown bottom right, to a jagged-edged familiar maple leaf (palmate), shown bottom left.

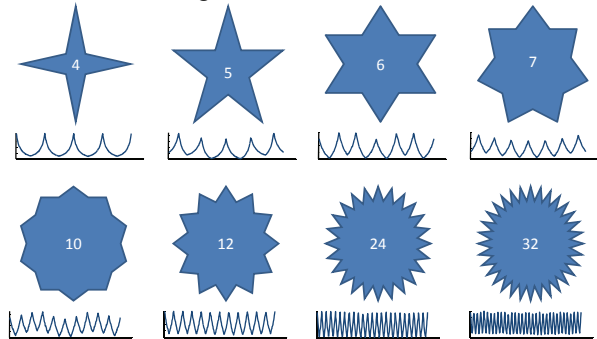


**Figure 7: Examples of objects from a single domain, which have different shape complexities**

The reason why this matters is that pairs of complex objects, even those which subjectively may seem very similar to the human eye, tend to be further apart under current distance measures than pairs of simple objects. This fact introduces errors in nearest neighbor classification, because complex objects are incorrectly assigned to a simpler class.

We first illustrate the necessity for a complexity-invariant distance for time series using a synthetic data set of figures with different shape complexities. Note that our use of synthetic data here is for clarity; as we hinted at in Figure 7 and will show later, the complexity problem occurs in many real data sets.

Figure 8 presents some geometric figures with increasing shape complexity. Each figure is labeled with its number of edges for reference. As in Figure 6, the two-dimensional shapes are converted to a single-dimensional “time” series by calculating the distance between the central point and the figure contour. Figure 8 also presents the z-normalized time series created for each figure.



**Figure 8: A set of geometric figures with increasing shape complexity and the respective “time” series extracted by calculating the distance between the central point and its contour**

We calculated distances between every pair of time series in Figure 8 using Euclidean distance and present the results in Table 1.

**Table 1: Euclidean distance matrix for the geometric figures data set**

	4	5	6	7	10	12	24	32
4		1.000	1.122	1.231	1.181	1.048	1.155	1.170
5			1.318	1.068	1.103	1.153	1.165	1.180
6				1.088	1.097	1.103	1.186	1.200
7					1.217	1.199	1.198	1.191
10						1.263	1.195	1.214
12							1.135	1.199
24								1.191
32								

We can use the numbers in Table 1 (and the dendrogram created from them, as shown in Figure 10.*left*) to explore the most similar shapes according

Euclidean distance. Considering the figures with simple shapes, the results are intuitive: The 4-edge star is the most similar object to the 5-edge star and vice-versa; the 7-edge figure is the most similar to the 6-edge star, etc. However, when we consider figures with complex shapes, the results are not so clear. The simplest figure, the 4-edge star, is the most similar to the 32-edge and the 12-edge figures. For the 24-edge figure, even though the 12-edge is the nearest one, the second and third most similar objects are the 4 and 5-edges stars, respectively.

This example illustrates a phenomenon that can be summarized in a simple statement: the distance between pairs of complex time series is frequently greater than the distance between pairs of simple time series. In fact, complex time series are commonly considered more similar to simple time series than to other complex time series they look like.

### 3.1 CID

Complexity invariance uses information about complexity differences between two time series as a correction factor for existing distance measures. In this section we restrict our discussion to Euclidean distance, and we consider the use of complexity invariance in other distance measures in Section 4.3.

The Euclidean distance,  $ED(Q, C)$ , between two time series  $Q$  and  $C$ , can be made complexity-invariant by introducing a correction factor:

$$CID(Q, C) = ED(Q, C) \times CF(Q, C)$$

Where  $CF$  is a complexity correction factor defined as:

$$CF(Q, C) = \frac{\max(CE(Q), CE(C))}{\min(CE(Q), CE(C))}$$

And  $CE(T)$  is a complexity estimate of a time series  $T$ . Before discussing how  $CE$  can be implemented, it is worth noticing that  $CF$  accounts for differences in the complexities of the time series being compared.  $CF$  forces time series with very different complexities to be further apart. In the case that all time series have the same complexity,  $CID$  simply degenerates to Euclidean distance.

This formulation leaves only the question of *how* to measure the complexity. The complexity of a time series can be estimated by a multitude of different approaches, such as Kolmogorov complexity [15], many variants of entropy [1][2], the number of zero crossings, etc.

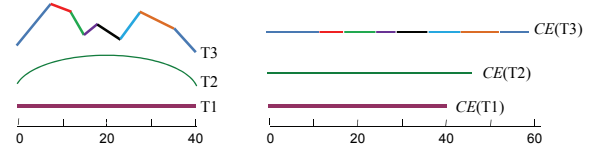
We can consider the desirable properties any such complexity measure should have:

- It should have low time and space complexity;
- It should have few parameters, ideally none;
- It should be intuitive and interpretable.

Given these above desideratum, we now present *one* possible complexity measure. It has  $O(1)$  space and  $O(n)$  time complexity, is completely parameter-free, and has a natural interpretation. It also was empirically

the best on average of the dozen or so possible measures we tried<sup>3</sup>. Nevertheless, we emphasize that we are not claiming this is the optimal measure (if such a thing is even well defined). Recall that the contribution of this work is to point out the complexity invariance problem, and to show an existence proof of a method to mitigate it.

Our approach to estimate the complexity of a time series is very simple. It is based on the physical intuition that if we could “stretch” a time series until it becomes a straight line, a complex time series would result in a longer line than a simple time series. Figure 9 illustrates this idea with some examples.



**Figure 9: left) Three time series can have their complexity measured by stretching them and measuring the length of the resulting lines (right)**

Our complexity estimate can be computed as follows:

$$CE(Q) = \sqrt{\sum_{i=1}^{n-1} (q_i - q_{i+1})^2}$$

The CID approach for complexity estimation can be easily implemented in any programming language. In order to reinforce this statement, Table 2 lists not the pseudo-code algorithm, but the *entire* Matlab code to compute CID.

**Table 2: Matlab code for Complexity Invariance Distance**

1	function d = CID(Q, C)
2	CE_Q = sqrt(sum(diff(Q).^2));
3	CE_C = sqrt(sum(diff(C).^2));
4	d = sqrt(sum((Q - C).^2)) * ...
5	(max(CE_Q, CE_C) / min(CE_Q, CE_C));

The code in Table 2 is computationally efficient since computing the complexity estimates is  $O(n)$ . Therefore, the time complexity of the entire code is  $O(n)$ . However, we can make this code a little more efficient if the complexity estimates are pre-computed and stored in a table. For example, in an exhaustive  $k$ -nearest neighbor search, each query must be compared to the entire database. Complexity estimates for the query and all time series in the database can be pre-computed and stored with negligible overhead. In this case, lines 2 and 3 in Table 2 can be replaced by a simple table lookup.

<sup>3</sup> We direct the interested reader to the paper's website for a detailed discussion of other complexity estimation functions [3].



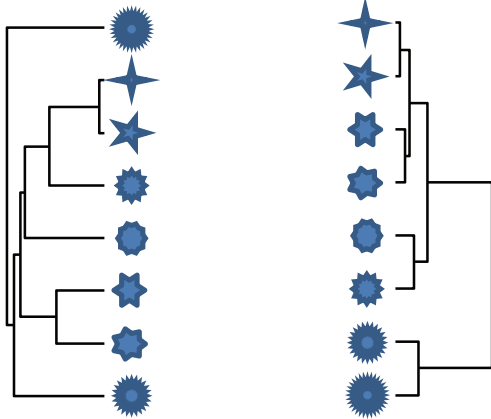
In order to provide consistent complexity estimates across an entire data set, CE requires that the time series under comparison should have the same number of observations, the same sampling rate and normalized amplitudes.

We can get our first hint of the utility of CID by considering the shape data set of the previous section. Table 3 presents the pairwise CID distances. Note that CID distances tend to be small between time series with similar complexities and increase as we compare time series with different complexities.

**Table 3: CID matrix for the geometric figures**

	4	5	6	7	10	12	24	32
4		1.000	1.061	1.229	1.707	1.839	3.843	5.042
5			1.307	1.138	1.700	2.159	4.135	5.422
6				1.096	1.599	1.953	3.982	5.214
7					1.651	1.977	3.744	4.819
10						1.439	2.580	3.396
12							2.018	2.761
24								1.446
32								

We also clustered this data set using hierarchical clustering with average linkage. Figure 10 presents the results for Euclidean distance (*left*) and CID (*right*).



**Figure 10: Hierarchical clustering of geometric figures data set using Euclidean distance (*left*) and CID (*right*)**

Here the results for Euclidean distance are counterintuitive, with complex figures linked directly to clusters formed by simpler figures.

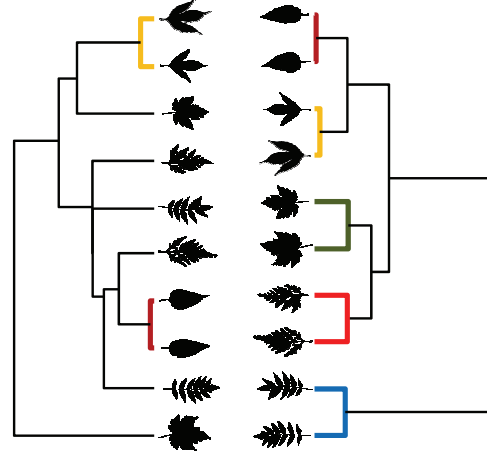
### 3.2 CID on a Natural Problem

The skeptical reader may wonder if the need for a complexity-invariant distance is restricted to our contrived synthetic example. Before applying CID to a large set of classification data sets, we present in this section one example of CID applied to natural data.

This example consists of images of leaves that have been converted to time series. The classification of leaves is an important problem in agriculture and in

understanding biodiversity, and has attracted significant recent interest [9].

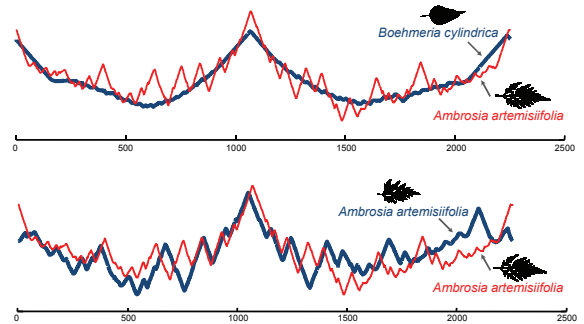
Pairs of leaf images with different shape complexity were randomly selected and clustered with Euclidean distance and CID using average linkage. Figure 11 illustrates the results that were obtained.



**Figure 11: Hierarchical clustering of leaf images with Euclidean distance (*left*) and CID (*right*)**

Even though all the pairs of leaves are quite similar to the human eye, Euclidean distance can only correctly identify two clusters (marked by heavy lines in Figure 11.*left*). Not surprisingly, these two clusters are formed by the two pairs of leaves with the simplest shapes. Complex leaves do not form their own clusters and are attached directly to clusters formed by simpler leaves. In contrast, CID was able to identify clusters of pairs of leaves with complex shapes, as shown in Figure 11.*right*.

We can use the time series extracted from leaf images to better illustrate why Euclidean distance frequently considers complex time series to be more similar to simple time series than to other complex (but apparently similar) time series. Figure 12 presents a comparison of three leaves.



**Figure 12: top) A comparison of a simple-shaped leaf (*Boehmeria cylindrica*) and a complex-shaped leaf (*Ambrosia artemisiifolia*) that results in smaller Euclidean distance than when two complex-shaped leaves (*Ambrosia artemisiifolia*) are compared to each other (bottom)**

Figure 12 presents a comparison of a leaf with a simple shape, *Boehmeria cylindrical*, and a leaf of complex shape, *Ambrosia artemisiifolia* (*top*); and a comparison of two exemplars of *Ambrosia artemisiifolia* (*bottom*). According to Euclidean distance, the distance between *Boehmeria cylindrical* and *Ambrosia artemisiifolia* (the distance between the top two time series) is smaller than the distance between the two exemplars *Ambrosia artemisiifolia* (the two bottom time series). The intuition behind this unintuitive result is that simple time series frequently present an “average” pattern that helps to minimize the Euclidean distance. In contrast, complex time series have a large number of peaks, in different quantities, amplitudes and durations. It is very unlikely that all these features match, even to another similar time series, thus the many minor local differences rapidly accumulate to produce a large global Euclidean distance.

#### 4. EMPIRICAL EVALUATION

We begin by stating our experimental philosophy. We have designed all experiments such that they are not only reproducible, but easily reproducible. For example, if a reader wishes to reproduce any figure in this paper, they can simply run the script we provide to do so. The scripts are designed to be easily expandable in order to include new data sets or distance measures, for example.

To this end, we have built a webpage which contains all data sets and code used in this work, together with spreadsheets which contain the raw numbers displayed in all of the figures. In addition, this webpage contains many additional experiments which we could not fit into this work; however, we note that this paper is completely self-contained. We have made the script available in Matlab, since this is essentially free for anyone in academia, and a completely free clone version (Octave) is available.

Finally, we note that in this paper we are testing *every* publicly available class-labeled time series data set in the world. Twenty of these data sets have been available for five years at [14], and used in more than one hundred papers. The remainder will be publicly released with the publication of this paper.

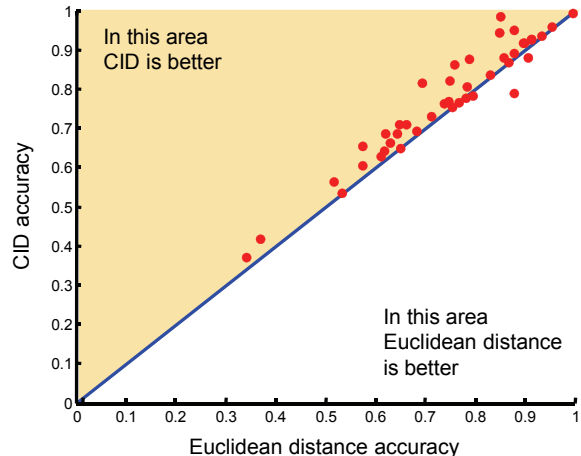
##### 4.1 Basic Accuracy

We performed our evaluation using the largest set of time series classification data sets ever attempted. In total, the evaluation includes 43 data sets from different domains, including medicine, entomology, engineering, astronomy, signal processing, and many others. A detailed description of the data sets is available at the paper website [3].

We tested the accuracy of both Euclidean distance and CID using the five-year-old predefined splits for 20 of the data sets that came from [14], and defined similar splits for the remaining 23 data sets. We emphasize

that in the latter case, we split the data just once, before testing any accuracy results.

Recall that neither distance measure has any parameters to be learned in the training stage. Our experimental results are visually summarized in Figure 13; the raw numbers are available at [3].



**Figure 13: CID compared with Euclidean distance using 1-Nearest Neighbor classifier accuracy. Each point represents one data set, with its X-axis value being its Euclidean distance accuracy, and its Y-axis value being its CID accuracy**

We believe that the results speak for themselves. However, notice that we are publishing results for *all* classification benchmark data sets we have at hand, and *not* for a subset of any kind. Nevertheless, CID improved classification accuracy in 32 of the 43 data sets, had the same performance as Euclidean distance in 3 data sets and decreased the accuracy in only 8.

We conclude this section by observing that CID improves accuracy because of the reasons we claim; i.e., CID avoids considering overly simple objects as similar to complex objects. In order to verify this hypothesis we performed a simple experiment: we collected data on all time series (from all data sets) that were misclassified by Euclidean distance and correctly classified by CID, and compared the complexity of the nearest neighbors provided by each distance measure. In 87.54% of the cases, CID assigned a more complex nearest neighbor than Euclidean distance did. It is reasonable to assume that, if CID could improve accuracy for any other (unknown) reason, we would expect that only approximately 50% of the CID nearest neighbors would be more complex than the Euclidean distance nearest neighbors.

##### 4.2 The Texas Sharpshooter Fallacy

As impressive as these results are, we must be careful to avoid a simple logic error that seems pervasive in time series classification papers. Many papers in the last three or four years test their algorithm/distance measure on all 20 data sets in the UCR archive, and

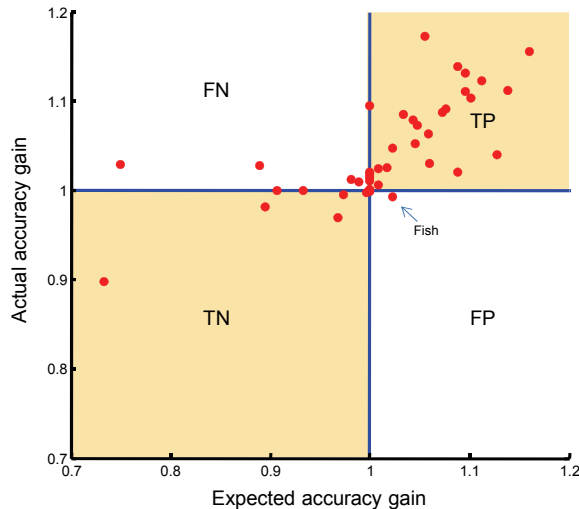
note that their method wins on some, ties on many, and loses on some. They then claim “*our method has better accuracy in some domains; therefore we could use it in those domains, and thus it has value.*” However, it is not useful to have an algorithm that can be accurate on *some* problems unless you can tell in advance on *which* problems it will be more accurate (this is a subtle version of the Texas sharpshooter fallacy.)

In order for the algorithm to be useful, we must show that we can predict ahead of time when our method will have superior accuracy. The obvious way (but not the only possible way) to do this is to test the accuracy of both Euclidean distance and CID looking *only* at the training data, and using these results to choose which algorithm to use to classify an object from the testing set.

We measured the accuracy of Euclidean distance and CID on training data using leaving-one-out cross-validation and calculated the expected accuracy gain as:

$$\text{gain} = \frac{\text{accuracy CID}}{\text{accuracy Euclidean}}$$

Obviously, gain values greater than one indicate that we expect CID will outperform Euclidean distance on a given data set; and gain values lower than one indicate the opposite. We also measured the *actual* accuracy gain (or loss) using testing data. The results are shown in Figure 14.



**Figure 14: Expected accuracy gain calculated on training data versus actual accuracy gain on testing data**

Note that the figure is essentially a real-valued version of a contingency table, and we have labeled the four regions with the four familiar labels. Let us consider the four cases we observe in the order of the evidence of utility for CID:

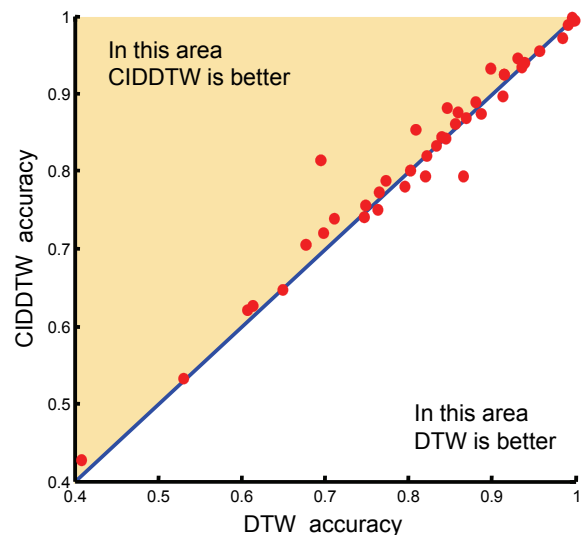
- **TP)** In this region we claimed ahead of time that CID would improve accuracy, and we were

correct. Gratifyingly, the vast majority of data points are in this region;

- **TN)** In this region we *correctly* claimed ahead of time that CID would decrease accuracy;
- **FN)** In this region we claimed ahead of time that CID would decrease accuracy, but the accuracy actually increased. This represents a lost opportunity to improve, but note that we are no worse off than if we had not tried CID;
- **FP)** This region is the only truly bad case for our method. Data points falling in this region represent cases where we thought we could improve accuracy, but did not. The good news is that only one case falls into this category, the Fish data set. For this data set we predicted an improvement of only 2% and obtained an accuracy reduction of 0.7%. In general, all cases near point (1,1) are not of special interest, since they represent marginal increases/decreases in accuracy.

### 4.3 Is There Some Other Way?

The reader may wonder if the impressive results of CID are simply a reinterpretation of some other invariance. In particular, it might be imagined that DTW could mitigate some of the effects of differing complexities. In brief, the answer is no. Complexity invariance is *not* subsumed by any other known invariance. To see this we applied the correction factor of Section 3.1 to DTW, and named this new measure of CIDDTW. We then compared the classification accuracy of both DTW and CIDDTW using a one-nearest neighbor classifier. Figure 15 presents the results we obtained.



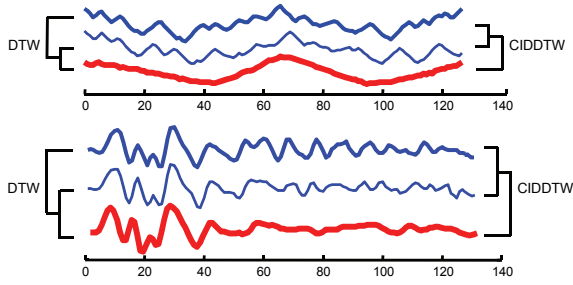
**Figure 15: CIDDTW compared with DTW distance using one nearest neighbor classifier accuracy.**

For DTW, we searched for the best Sakoe-Chiba band size using the one-nearest neighbor classifier on training data only. For CIDDTW we did *not* search for



the best band size, and simply used the window sizes obtained for DTW. Therefore, the results in Figure 15 might be pessimistic for CIDDTW. Nevertheless, CIDDTW outperformed DTW in 24 data sets, drew in 7 and lost in 12.

DTW is one of the most popular distance measures for time series due to its ability to align, in a non-linear manner, time series that are locally out of phase (cf. Figure 3)[5][10]. This feature can be used to *partially* deal with the complexity problem. Intuitively, DTW can be used to align peaks and valleys of complex time series, reducing the distance between them. However, complex time series frequently have *different amounts* of peaks and valleys and aligning some of them does not fully solve the problem. In Figure 16 we show that there are instances where DTW can be “fooled” by simple time series in several classification domains.



**Figure 16: Some examples misclassified by DTW with a one-nearest neighbor classifier, and correctly classified by CIDDTW. Data sets are SwedishLeaf (top) and FaceAll (bottom)**

## 5. USEFUL PROPERTIES OF CID

As we are producing a new distance measure, the vast majority of our empirical evaluation has focused on questions of *accuracy* (cf. Sections 4.1 and 4.2). However, *given* that we have demonstrated that CID is an accurate measure, the next natural question to ask is about its efficiency. As the CID measure is only  $O(n)$ , it is clearly efficient for simple main memory similarity search problems. In fact, the time difference between Euclidean distance and CID is only perceptible with very careful experiments.

However, the issue is less clear for disk-resident problems, or for higher-level problems that require (at least in principle) a number of comparisons that are quadratic in the number of objects. Examples of such problems include motif discovery [17], some types of clustering and outlier detection [20][21].

Essentially all algorithms in the literature that mitigate the potential intractability of disk-resident or higher-level data mining use one (or both) of two well-known ideas, *lower bounding* and the *triangular inequality*. Many of the novel distance measures proposed for time series do not (or initially did not) allow leveraging off lower bounding and the triangular inequality [5]. For example, the rotation invariance that greatly

improved the classification accuracy of star light curves in Section 2.2 does not allow triangular inequality acceleration [13], and DTW initially only allowed very weak lower bounds until the invention of envelope-based lower bounds [10]. In this section we consider each speedup technique with reference to CID.

### 5.1 Lower Bounding of CID

Since the introduction of the GEMINI framework [7] in 1994 the idea of lower bounding has been a cornerstone technique for speeding up the indexing and mining of time series (and many other kinds of data). The basic idea is to produce a low-dimensionality representation of the data and produce a distance measure defined by the reduced data such that:

$$D_{\text{reduced data}}(A, B) \leq D_{\text{original data}}(A, B)$$

Once this has been done, it is trivial to index the data with any off-the-shelf multidimensional index structure. While the details differ slightly for each index structure, the basic idea is always the same. The query is projected into the same reduced dimensionality space, and using  $D_{\text{reduced data}}$  we search the approximate data (held in the main memory). We then proceed to load the most promising object from the disk and measure its distance to the query in the original dimensional space, updating the *best-so-far* variable. We continue to retrieve the most promising candidates, updating the *best-so-far* if appropriate, until the next most promising candidate has a value (measured in the  $D_{\text{reduced data}}$  space) that is greater than the *best-so-far*. At this point we can admissibly abandon the search.

Can we use lower bounding with CID? The answer is affirmative and general. We can use *any* of the dozens of lower bounding representations for time series [5] with CID by only changing a single line of code. The intuition behind this result is to note that CID can only be greater than or equal to ED. For example, the first paper in the GEMINI framework used the Fourier Transform (DFT) for lower bounding [7]:

$$D_{\text{DFT}}(A, B) \leq D_{\text{original data}}(A, B)$$

But CID is always greater than or equal to ED, hence:

$$D_{\text{DFT}}(A, B) \leq D_{\text{original data}}(A, B) \leq D_{\text{CID}}(A, B)$$

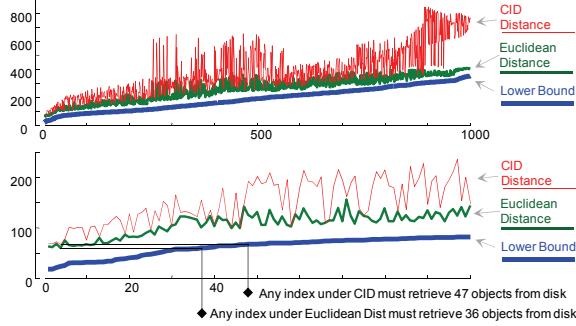
So we automatically inherit the lower bounding:

$$D_{\text{DFT}}(A, B) \leq D_{\text{CID}}(A, B)$$

We must regard this news with some caution. It is always possible to create a lower bound to *any* distance measure by hard coding  $D_{\text{reduced data}}$  to zero. In that case, the index structure degenerates to a brute force search. What we need is for the lower bound to be a *tight* estimate of the distance. We can see how we fare here with a simple experiment. We created a database of 1,000 random walk time series of length 128. We approximated the data with an eight-dimensional PAA

approximation<sup>4</sup>. We sorted the main-memory approximations of the candidates by listing the *most promising first*, producing the heavy (blue) line in Figure 17.

The first item in this list has a smallest value of 16.8. This is not a particularly tight lower bound, as the true ED, plotted above it in medium (green) is 63.1, and the CID plotted above them, both in light (red), is 68.6.



**Figure 17: *top*) A visual intuition of using lower bounding to accelerate disk-based search. *bottom*) A zoom-in of the above. For ED or CID we must keep retrieving objects from the disk until the smallest *true* distance calculated is less than the next candidate's lower-bound distance. For ED, this happens after we have seen 36 objects, and for CID after 47 objects**

Note that we can tell from this plot the *best* possible pruning for any indexing structure given our assumed parameters. We can prune off at most 964 objects under ED, and at most 953 objects under CID. For all intents and purposes, these numbers are the same: CID allows 98.9% of the pruning that ED allows. Furthermore, for larger data sets (recall this was a mere 1,000 objects) this percentage grows ever closer to 100%.

In summary, CID can use the hundreds of indexing and mining algorithms that exploit ED lower bounding [5][7] with the most trivial coding effort and a negligible loss of efficiency.

## 5.2 CID Obeys the $\rho$ -relaxed Triangular Inequality

There are literally dozens of indexing and data mining algorithms for metric spaces that rely on triangular inequality to prune distance calculations [4][6][18]. Although the details of how triangular inequality is employed vary for different algorithms, we consider a concrete technique to illustrate the general idea.

### 5.2.1 A Brief Review of Orchard's Algorithm

One of the simplest indexing techniques known is Orchard's algorithm [15]. The general idea behind it is to store a table with the distances between every pair

of objects in a database<sup>5</sup>. In order to decide which database instance is the nearest neighbor to a user-given query  $Q$ , Orchard's algorithm first calculates the distance  $D(Q, C_i)$  between  $Q$  and a randomly chosen database instance  $C_i$ .

Using  $D(Q, C_i)$  and the triangular inequality property:

$$D(A, B) \leq D(A, C) + D(C, B)$$

We can safely prune some distance calculations between  $Q$  and other training instances. Suppose that we want to know if we can prune the distance calculation to a given instance  $C_j$ . Using triangular inequality:

$$D(C_i, C_j) \leq D(C_i, Q) + D(Q, C_j)$$

Reordering the last equation to isolate the distance between the query object and the pruning candidate  $C_j$ ,  $D(Q, C_j)$ :

$$D(Q, C_j) \geq D(C_i, C_j) - D(C_i, Q)$$

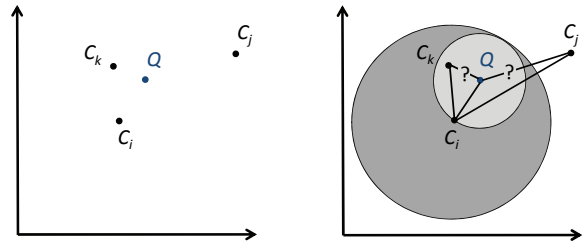
Note that when we have:

$$D(C_i, C_j) \geq 2 D(C_i, Q) \quad (5.1)$$

We can safely conclude that:

$$D(Q, C_j) \geq D(C_i, Q)$$

Therefore, when the distance between the database instances  $C_i$  and  $C_j$  is at least twice the distance between  $C_i$  and  $Q$ , we can safely prune  $C_j$ , since it cannot be closer to the query object  $Q$  than  $C_i$ . Notice that no distance calculations were necessary at classification time, just constant time table lookups. This is because  $D(C_i, C_j)$  is a known quantity since Orchard's algorithm stores a table with distances between all pairs of training instances. Figure 18 illustrates this idea.



**Figure 18: *left*) Assume we know the pairwise distances between  $C_i$ ,  $C_j$  and  $C_k$ . A newly arrived query  $Q$  must be answered. *right*) After calculating the distance  $D(Q, C_i)$  we can conclude that items with a distance to  $C_i$  less than or equal to  $2 \times D(Q, C_i)$  (i.e., the dark gray area) *might* be the nearest neighbor of  $Q$ , but everything else, including  $C_j$  in this example, can be excluded from consideration**

When a distance calculation cannot be pruned, such as in the case of  $C_k$  in Figure 18, the distance  $D(C_k, Q)$  is

<sup>4</sup> For data lengths/reduced dimensionality that are powers of two, PAA is exactly equivalent to Haar wavelets [5].

<sup>5</sup> Therefore, Orchard's algorithm requires  $O(m^2)$  in space, where  $m$  is the number of database objects. However, [22] shows how to significantly reduce the space requirement, while producing nearly identical speedup.

calculated and if  $D(C_k, Q) < D(C_i, Q)$ ,  $C_k$  becomes the new nearest neighbor currently known (i.e., the *best-so-far*), consequently reducing the pruning radius. Our interest in Orchard's algorithm is due to the fact that this algorithm relies *exclusively* on the triangular inequality to improve search performance. However, many other indexing schemes are strongly related [8].

### 5.2.2 The Relaxed Triangular Inequality

CID does *not* obey the triangular inequality; however, it does obey a relaxed version of this property:

$$D_{CID}(A, B) \leq \rho (D_{CID}(A, C) + D_{CID}(C, B))$$

With  $\rho = CF(A, B)$ . We postpone a formal proof of this property to Appendix A<sup>6</sup>.

The  $\rho$ -relaxed triangular inequality property implies that we can search in CID space using the same algorithms designed for metric spaces, after some scaling [4]. In practice, all metric indexing techniques can be adapted to CID by changing just a few lines of source code. For concreteness we illustrate this with Orchard's algorithm.

Suppose we have calculated the distance  $D_{CID}(Q, C_i)$ , and want to know if an instance  $C_j$  can be safely pruned.

We start with the  $\rho$ -relaxed triangular inequality:

$$D_{CID}(C_i, C_j) \leq \rho (D_{CID}(C_i, Q) + D_{CID}(Q, C_j))$$

And reorder to isolate  $D_{CID}(Q, C_j)$  on the left-hand side of the equation:

$$\rho D_{CID}(Q, C_j) \geq D_{CID}(C_i, C_j) - \rho D_{CID}(C_i, Q)$$

This equation is equivalent to:

$$D_{CID}(Q, C_j) \geq \frac{1}{\rho} D_{CID}(C_i, C_j) - D_{CID}(C_i, Q)$$

Therefore, we require that:

$$D_{CID}(C_i, C_j) \geq 2 \rho D_{CID}(C_i, Q) \quad (5.2)$$

With  $\rho = CF(C_i, C_j)$  in order to safely prune the calculation of  $D(Q, C_j)$ .

Thus, to adapt Orchard's algorithm to CID we just need to make two simple modifications:

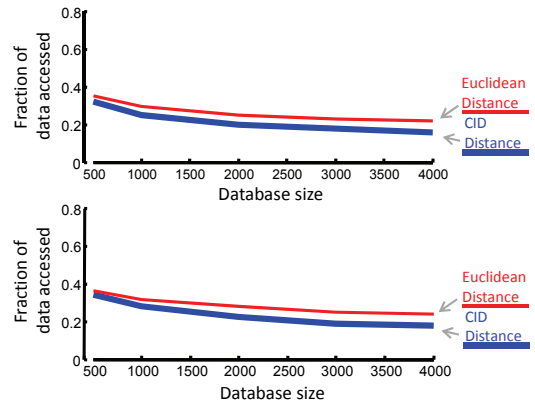
- We must use Equation 5.2 instead of Equation 5.1 when pruning database instances;
- We must store the complexity estimates,  $CE$ , of each database instance. The space overhead for storing the complexity estimates is  $O(m)$ , where  $m$  is the number of database objects, and thus is small relative to the typical overhead for most indexing data structures.

In order to illustrate that pruning with  $\rho$ -relaxed triangular inequality under CID can be effective, we ran a series of experiments using a database of random walk time series indexed with Orchard's algorithm.

<sup>6</sup> Notice that  $\rho = CF(A, B)$  is not a bounded value; however, for indexing purposes this fact has no major consequences.  $\rho$ -relaxed triangular inequality implies  $2\rho$ -inframetric inequality. This means that the following property also holds:  $D_{CID}(A, B) \leq 2 \rho \max(D_{CID}(A, C), D_{CID}(C, B))$

Our experiment consisted of running paired tests; in other words, the *same* database, queries and other parameters were used to measure performance for both CID and Euclidean distance. We used the mean percentage of the database accessed in each query as the main form to assess the performance of each technique.

Figure 19 presents the results for database sizes ranging from 500 up to 4,000 time series. For each database size we ran a series of 1000 nearest neighbor queries. We ran experiments with time series lengths of 32, 64, 128, 256, 512 and 1024 observations. In every experiment CID outperformed Euclidean distance in the number of distance calculations pruned. For brevity, we show in Figure 19 the results for time series of 32 (*top*) and 1024 (*bottom*) observations.



**Figure 19: A comparison of fraction of data accessed with Euclidean distance and CID using random walk time series with 32 (*top*) and 1024 (*bottom*) observations indexed with Orchard's algorithm**

To be clear, we are not claiming that CID outperforms Euclidean distance for all indexing algorithms in general or for every data set when using Orchard's algorithm. For brevity, we reserve a more detailed performance analysis of indexing CID space for future work.

## 6. CONCLUSION AND FUTURE WORK

In this work we have surveyed all the existing invariances for time series similarity measures, and demonstrated the previously unknown need for complexity invariance. We have introduced CID, a simple and parameter-free method to mitigate the problem, and demonstrated its utility in improving classification accuracy on dozens of data sets.

We have further shown that the use of CID need not compromise efficiency, since it can be used with the vast majority of indexing and mining algorithms with only very minor changes. In future work we plan to consider the utility of CID for the problem of motif discovery [17] and outlier detection [20][21].

## 7. ACKNOWLEDGMENTS

Thanks to Abdullah Mueen and Pavlos Protopapas for their help with the star light curve experiments, to Bing Hu and Yuan Hao for their help preparing some of the datasets, and to Thanawin Rakthanmanon, Ronaldo C. Prati and Edson T. Matsubara for their suggestions on a draft version of this paper. This work was funded by NSF awards 0803410 and 0808770, FAPESP award 2009/06349-0 and a gift from Microsoft.

## 8. REFERENCES

- [1] S. L. G. Andino, et al, *Measuring the complexity of time series: an application to neurophysiological signals*. Human Brain Mapping, 11(1), pages 46-57, 2000.
- [2] W. Aziz and M. Arif, *Complexity analysis of stride interval time series by threshold dependent symbolic entropy*, EJAP, 98 (1), pages 30-40, 2006.
- [3] G.E.A.P.A. Batista, *Website for this paper*: <http://www.icmc.usp.br/~gbatista/cid>, 2011.
- [4] E. Chávez, G. Navarro, R. Baeza-yates, J. L. Marroquín, *Searching in metric spaces*, ACM Computing Surveys, 33, pages 273-321, 1999.
- [5] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang and E. Keogh, *Querying and mining of time series data: experimental comparison of representations and distance measures*. VLDB, pages 1542-1552, 2008
- [6] C. Elkan, *Using the Triangle Inequality to Accelerate k-Means*, ICML, pages 147-153, 2003.
- [7] C. Faloutsos, C., M. Ranganathan, Y. Manolopoulos, *Fast subsequence matching in time-series databases*. ACM SIGMOD Record 23(2), pages 419-429, 1994.
- [8] G.R. Hjaltason, H. Samet, *Index-driven similarity search in metric spaces*, ACM Transactions on Database Systems, 28(4), pages 517-580, 2003.
- [9] D.J. Hearn, *Shape analysis for the automated identification of plants from images of leaves*. Taxon, 58(3), pages 934-954, 2009.
- [10] E.J. Keogh, *Exact indexing of dynamic time warping*. VLDB, pages 406-417, 2002.
- [11] E. J. Keogh, *Efficiently finding arbitrarily scaled patterns in massive time series databases*, PKDD, pages 253-265, 2003.
- [12] E. J. Keogh, S. Lonardi, C. A. Ratanamahatana, L. Wei, S. Lee, J. Handley, *Compression-based data mining of sequential data*, DMKD 14(1), pages 99-129, 2007.
- [13] E. J. Keogh, L. Wei, X. Xi, M. Vlachos, S. Lee, P. Protopapas, *Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures*, VLDB Journal 18(3), pages 611-630, 2009.
- [14] E. J. Keogh, X. Xi, L. Wei, C. Ratanamahatana, *The UCR time series classification/clustering homepage*: [www.cs.ucr.edu/~eamonn/time\\_series\\_data/](http://www.cs.ucr.edu/~eamonn/time_series_data/), 2006.
- [15] M. Li, P. Vitanyi, *An introduction to Kolmogorov complexity and its applications*, Second Edition, Springer Verlag, 1997.
- [16] K. Li, M., Yan, and S. Yuan, *A simple statistical model for depicting the CDC15-synchronized yeast cell-cycle regulated gene expression data*. Statistica Sinica 12, pages 141-158, 2002
- [17] A. Mueen, E.J. Keogh, N. B. Shamlo, *Finding time series motifs in disk-resident data*. ICDM, pages 367-376, 2009.
- [18] A. Moore, *The anchors hierarchy: using the triangle inequality to survive high dimensional data*. UAI, pages 397-405, 2000.
- [19] M. T. Orchard, *A fast nearest-neighbor search algorithm*. ICASSP, pages 2297- 2300, 1991.
- [20] P. Protopapas, J. M. Giammarco, L. Faccioli, M. F. Struble, R. Dave, C. Alcock, *Finding outlier light curves in catalogues of periodic variable stars*. Monthly Notices of the Royal Astronomical Society, 369(2), pages 677-696, 2006.
- [21] D. Yankov, E.J. Keogh, U. Rebbapragada, *Disk aware discord discovery: finding unusual time series in terabyte sized datasets*. KAIS, 17 pages 241-262, 2008.
- [22] L. Ye, X. Wang, E. Keogh, A. Mafrá-Neto, *Autocannibalistic and anyspace indexing algorithms with applications to sensor data mining*. SIAM SDM, pages 85-96, 2009.
- [23] J. Zunic, P. Rosin, L. Kopanja, *Shape orientability*. ACCV, pages 11-20, 2006.

## APPENDIX A: $\rho$ -RELAXED

### TRIANGULAR INEQUALITY PROOF

In this section we prove that CID obeys the  $\rho$ -relaxed triangular inequality:

$$D_{CID}(A, B) \leq \rho(D_{CID}(A, C) + D_{CID}(C, B))$$

We start our proof by stating the triangular inequality of Euclidean distance:

$$ED(A, B) \leq ED(A, C) + ED(C, A)$$

Remember that the complexity correction factor  $CF$  is a quantity greater than or equal to one; therefore, we can multiply both sides of the inequality by  $CF(A, B)$ :

$$ED(A, B)CF(A, B) \leq CF(A, B)(ED(A, C) + ED(C, A))$$

The left-hand side of the inequality is our definition of CID; hence:

$$D_{CID}(A, B) \leq \rho(ED(A, C) + ED(C, A))$$

With  $\rho = CF(A, B)$ .

Finally, we can again use the fact that  $CF$  is greater than or equal to one to change Euclidean distances to CID on the right-hand side of the equation:

$$D_{CID}(A, B) \leq \rho(ED(A, C)CF(A, C) + ED(C, A)CF(C, A))$$

And, therefore:

$$D_{CID}(A, B) \leq \rho(D_{CID}(A, C) + D_{CID}(C, A))$$