Efficiently Finding Unusual Shapes in Large Image Databases[1]

LI WEI wli@cs.ucr.edu
EAMONN KEOGH eamonn@cs.ucr.edu
XIAOPENG XI xxi@cs.ucr.edu
Department of Computer Science and Engineering, University of California, Riverside, CA 92521
MELISSA YODER melissay@ucr.edu
Department of Entomology, University of California, Riverside, CA 92521

Abstract

Among the visual features of multimedia content, shape is of particular interest because humans can often recognize objects solely on the basis of shape. Over the past three decades, there has been a great deal of research on shape analysis, focusing mostly on shape indexing, clustering, and classification. In this work, we introduce the new problem of finding shape *discords*, the most unusual shapes in a collection. We motivate the problem by considering the utility of shape discords in diverse domains including zoology, microscopy, anthropology, and medicine. While the brute force search algorithm has quadratic time complexity, we avoid this untenable lethargy by using locality-sensitive hashing to estimate similarity between shapes which enables us to reorder the search more efficiently and thus extract the maximum benefit from an admissible pruning strategy we introduce. An extensive experimental evaluation demonstrates that our approach is empirically linear in time.

Keywords

Anomaly Detection; Shape; Rotation Invariance.

1. Introduction

Large image databases are used in an increasing number of applications in fields as diverse as entertainment, business, art, engineering, and science (Tanaka and Uehara, 2004). Among the visual features contained in an image (e.g. shape, color, and texture), shape is of particular importance since humans can often recognize objects on the basis of shape alone (Zhang and Lu, 2004). Because of this special property, shape analysis has received much research attention in the past three decades. Most research effort in the shape analysis community is focused on indexing, clustering, and classification.

In this work, we propose the new problem of finding the shape that is least similar to all other shapes in a dataset. We call such shapes *discords*. Figure 1 gives a visual intuition of a shape discord found in an image dataset of 1,301 marine creatures. Note that while

[1] A primary version of this work appears in ICDM 2006.

most creatures are represented in the dataset several times, the starfish only appears once, and is thus reasonably singled out as the most unusual shape.

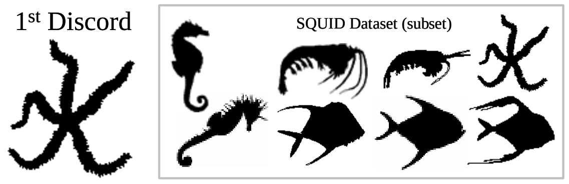


Figure 1: *Right)* Samples from a dataset of 1,301 images of marine creatures. *Left)* The No.1 shape discord found in this dataset is a starfish

Note also that any intuitive definition of shape discords will have to allow for invariance to the classic transformations that plague shape similarity measures. For example, the seahorse in the top left corner must be rotated approximately 40 degrees before matching the other seahorse, and eliminating itself as a contender for discord. The utility of shape discords is very clear; as shown in Figure 1 they allow a user to find surprising shapes in a massive database. Nevertheless, as we are introducing a new problem, we feel it appropriate to concretely motivate the utility of shape discords with several examples from diverse fields.

Medical Data Mining: *Drosophila melanogaster* is the species of fruit fly that has been most commonly used for genetic experiments in the last century. Drosophila is one of the most studied organisms in biological research, particularly in genetics and developmental biology. Drosophila is small and easy to breed in the laboratory, and its genome is short and “simple” (only four pairs of chromosomes). Genetic transformation techniques for this organism have been available since the late eighties. One common type of transformation is mutagenesis, where a specific gene is mutated and the developing organism is examined for changes in physiology or behavior. Figure 2 shows a subset of wing images collected for a mutagenesis experiment carried out at Florida State University (Zimmerman et. al., 2000), and the discord discovered by our algorithm. Note that the entire wing image was analyzed up to, but not including, the articulation (1 mm from wing attachment to thorax).

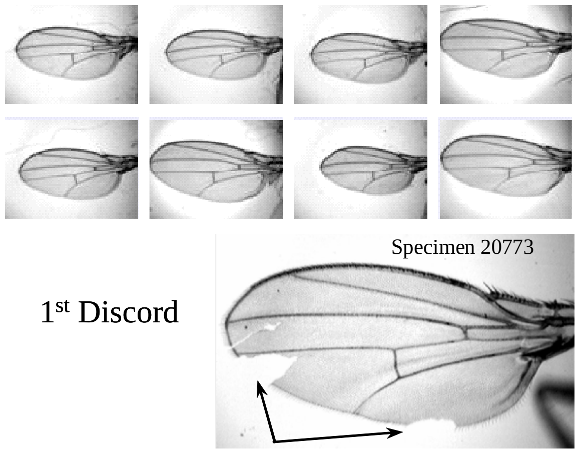


Figure 2: *Top)* A subset of 32,028 images of Drosophila wings. *Bottom)* The No.1 shape discord found in this dataset is a damaged wing

While the discord discovered in this example is likely due to a technicians mishandling of a sample and not due to the phenotypical mutation, the result still hints at the utility of shape discords. Note that a brute force attempt to find this discord would have required 512,880,378 shape comparisons.

Anthropological Data Mining: Anthropology offers many interesting challenges for data mining, particularly mining of *shapes* (Clark et. al., 2002). Examples of shapes which anthropologists may be interested in mining include petroglyphs, pottery (Clark et. al., 2002), projectile points ("arrowheads") (O'Brien et. al., 2001), and bones (Keogh et. al., 2006).

It is difficult to overstate the need for efficient algorithms when working with such datasets. For example, the number of projectile points in the collection at the authors' institution exceeds one million objects (Philip, 2006). We collected more than 16,000 projectile point images for an unrelated project, but can consider this dataset with our discord mining algorithm. As Figure 3 shows, the dataset comes from diverse sources, including photographs, onsite field sketches, and silhouettes. Since we are ignoring all but the shape information, this diversity of data sources makes no difference to the task at hand.

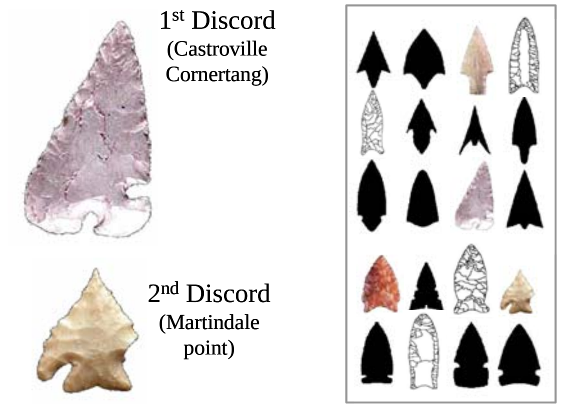


Figure 3: *Right)* A subset of 16,000 images of projectile points collected from diverse sources. *Left)* The top two discords found in the dataset

While some subjectivity exists, the two discords discovered are arguably the most unusual shapes in the dataset. While the vast majority of projectile points are symmetric, the first discord is an unusual asymmetric point from Texas. The second discord is a typical and common "Basal Notched" point, except this example has its right tang broken off. Discovering such anomalies by eye even in a mere 16,000 images is non-trivial, and motivates the need for a scalable algorithm.

Discords may have other uses. Shape clustering algorithms often suffer diminished utility due to a handful of outlier shapes in the dataset. We may reasonably expect discords to be among those tricky cases. Finding and removing them can serve as a preprocessing step for shape clustering algorithms. This idea may also be beneficial for image compression with techniques that use global bases, like Principal Component Analysis (PCA), since very unusual images are bound to introduce more bases or more reconstruction error among the 'true' bases (Jolliffe, 2002).

As we have shown, the notion of unusual shapes can be useful in different domains. However, to the best of our knowledge, the problem of finding these shapes has not yet been addressed. In this paper, we introduce a novel definition that defines *discord* as the shape that has the largest distance (as a measurement of difference) to its nearest neighbor. This definition has the advantage that the unusualness of a shape is not tied exclusively to its structure. Instead it depends on the similarity between it and its nearest neighbor, and is therefore context dependent. The definition eliminates the need of an explicit description of usual shapes. In addition, this definition requires zero parameters, which is especially suitable for data mining applications (Keogh et. al., 2004). In particular, as we demonstrate below, we can apply our algorithm in very diverse domains with a very little tuning or "tweaking".

The above examples also show that in real world applications, the size of the image database is very huge (usually the size of the database is much bigger than the length of the time series). Our definition of shape discord would be of little use to the data mining community without an efficient algorithm to discover it. The brute force algorithm requires a quadratic "all-to-all" comparison, which is simply untenable for large real-world datasets. We introduce an algorithm that uses locality-sensitive hashing to quickly discover likely candidates for discords, and use this information to generate heuristics to reorder the search in a more efficient way. The algorithm is able to avoid many fruitless calculations and can achieve three to four orders of magnitude speedup on real problems.

The rest of the paper is organized as follows. In Section 2, we discuss some background material and formally define the problem of shape discord discovery. In Section 3, we introduce a general framework for shape discord searching and provide observations for speeding it up. Section 4 presents an algorithm to enable an efficient search strategy based on locality-sensitive hashing. We perform a comprehensive empirical evaluation in Section 5 to demonstrate both the utility of shape discords and the efficiency of our search strategy. Finally Section 6 offers some conclusions and directions for future work.

2. Background and Related Work

This section begins with some necessary background material before introducing the formal definition of shape discords. We then discuss why existing novelty/outlier detection approaches are not suitable for the problem at hand.

2.1 Shape Representation and Distance Measures

We consider the shape of an object as a binary image representing the outline of the object (Loncarin, 1998). In order to find/index/classify a shape, the shape must be described or represented in some way. However this is a difficult task as shapes may be corrupted with noise, defects, arbitrary distortions, and occlusions. There are many shape representations and description techniques in the literature. Among them, one-dimensional representations have been shown to achieve comparable or superior accuracy in shape matching (Keogh et. al., 2006). Therefore this simple representation has been used by an increasingly large fraction of the literature (Davies, 1997; Keogh et. al., 2006; Van Otterloo, 1991).

There exist dozens of techniques to convert shapes into one-dimensional representations (also known as pseudo “time series”). We refer the interested reader to Loncarin, 1998 and Zhang and Lu, 2004 for excellent surveys. Note that our approach works for any of these representations. To give the reader a concrete idea, in Figure 4, we show the well-known *centroid distance* approach to convert a shape into a time series (Zhang and Lu, 2004). This approach is particularly attractive because it requires zero parameters and, after suitable normalization, it removes the effects of scale and offset. Rotation will be discussed below.

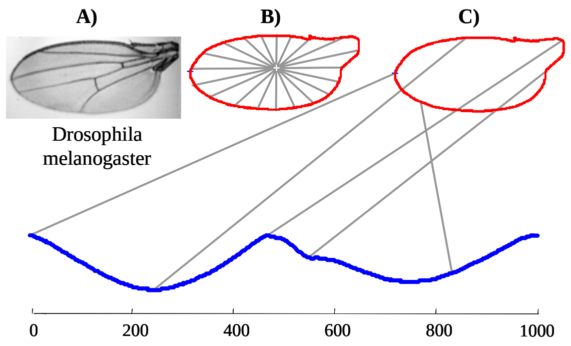


Figure 4: Shapes can be converted to time series. **A**) A bitmap of a fruit flies wing. **B**) The distance from every point on the profile to the center is measured and treated as the Y-axis value of a time series of length *n* (**C**)

Even though this one-dimensional representation is very simple, we claim that it is very effective in preserving features of the original shape. To show this, we conducted two experiments. The first one is a classification experiment on several publicly available shape datasets, as shown in Table 1. For each dataset, we converted the shapes to one-dimensional representations and performed one-nearest-neighbor classification using rotation invariant Euclidean distance (explained later). The error rates, as measured by leaving-one-out evaluation, are shown in the second column of Table 1. In the third column, we list the best error rates reported by other works, using more complicated shape representations/ distance measures.

Table 1: The classification error rate on several datasets

Dataset Name	Error Rate using 1D representation (%)	Error Rate using other representations (%)
Swedish Leaves	13.33%	17.82% (Söderkvist, 2001)
Chicken	19.96%	20.5% (Mollineda et. al, 2002)
MixedBag	4.375%	6% (Vlachos et. al., 2005)
Diatoms	27.53%	26% (Jalba et. al, 2005)

The second experiment is similarity-based retrieval on the MPEG-7 Core Experiment CE-Shape-1 dataset (Latecki et. al. 2000). This dataset was used by Latecki et. al. to compare the performance of several state-of-the-art shape descriptors, including correspondence of visual parts (Latecki and Lakamper 1999), Zernike moments (Khotanzan and Hong 1990), Wavelet descriptor (Chuang and Kuo 1996), Curvature-based descriptor (Mokhtarian and Machworth 1992), Directed Acyclic Graph (Blum 1973), etc. Among them, the method based on the correspondence of visual parts performs the best in experiment part B (performance of the similarity-based retrieval), where it got a retrieval rate of 76.45%. We conducted the same experiment using our one-dimensional representation and one-nearest-neighbor classification and achieved a retrieval rate of 82.27%.

In addition, we note that while there are dozens of shape representations in the literature, none of them can be easily hashed, which is required for our discord discovery algorithm. There is a well-known technique that *sounds* ideally suited for hashing, geometric hashing (Wolfson and Rigoutsos 1997). However, it is not useful for the problem at hand. Geometric hashing is based on the hashing of *vertices* of shapes. Most man-made objects do have well defined vertices, for example machine parts and architectural drawings have lots of right angles and 45 degree angles. In this paper, most of the datasets used (nematodes, butterflies, blood cells, fungus, drosophila wings and probably projectile points) do not have well defined vertices.

The above experiments and analysis show that the very simple time series representations of shapes and the simple Euclidean distance can be competitive to other more complex representations and distance measures. Therefore in this work, we only consider one-dimensional representation of the shapes.

From now on, we will use the words 'shape' and 'time series' interchangeably. Let us first formally define *time series*.

Definition 1. *Time Series*: A time series $C = c_1,...,c_n$ is an *ordered* set of n real-valued variables. In our case the ordering is not temporal but spatial; it is defined by a clockwise sweep of the shape boundary.

We assume that the length of the time series is much less than the size of the database. Recall that our task is to find the most unusual shape within a collection. We formally define the problem below.

Definition 2. *Shape Discord*: Given a collection of shapes *S*, shape *D* is the discord of *S* if *D* has the largest distance to its nearest match. That is, $\forall$ shape *C* in *S*, the nearest match M_C of *C* and the nearest match M_D of *D*, $Dist(D, M_D) > Dist(C, M_C)$.

Note that if there are two unusual shapes in the dataset and they are both unusual in the same way (the "twin freak problem"), we can simply change the definition to "find the shape that has the maximal distance to its second nearest neighbor". For simplicity, we will not consider this case in this work. However it is a trivial modification.

In many cases, we may be interested in examining the top *k* discords, which is a simple extension of the previous definition.

Definition 3. k^{th} *Shape Discord*: Given a collection of shapes *S*, shape *D* is the k^{th} discord of *S* if *D* has the k^{th} largest distance to its nearest match.

A critical component of the previous two definitions is the function to measure the distance between two time series, which we formally define below.

Definition 4. *Distance Function*: *Dist* is a function that has two time series *Q* and *C* (both of length *n*) as inputs and returns a nonnegative value as the distance from *Q* to *C*. For subsequent definitions to work, we require that the function be symmetric, that is, $Dist(Q, C) = Dist(C, Q)$.

As a concrete instantiation of a distance function, we define the most common distance measure for time series, *Euclidean distance* (Chiu et. al., 2003; Keogh and Kasetty, 2002).

Definition 5. *Euclidean Distance*: Given two time series *Q* and *C* of length *n*, the Euclidean distance between them is defined as

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

If the shapes are rotationally aligned, Euclidean distance will usually reflect the intuitive similarity. However if the shapes are not rotationally aligned, the corresponding time series will also be misaligned. In this case, Euclidean distance can produce extremely poor results. To overcome this problem, we need the distance function to be rotation invariant. To achieve this, we need to hold one shape fixed, rotate the other, and record the minimum distance of all possible rotations. We accomplish this in the time series space by representing all rotations of a shape by a *rotation matrix*.

Definition 6. *Rotation Matrix*: Given a time series *C* of length *n*, its possible rotations constitute a rotation matrix ***C*** of size *n* by *n*

$$\boldsymbol{C} = \left\{ \begin{matrix} c_1, c_2, \ldots, c_{n-1}, c_n \\ c_2, \ldots, c_{n-1}, c_n, c_1 \\ \vdots \\ c_n, c_1, c_2, \ldots, c_{n-1} \end{matrix} \right\}$$

where each row of the matrix is simply a time series shifted (rotated) by one time point from its neighbors. For notational convenience, we denote the i^{th} row as C^i, which allows us to denote the rotation matrix in the more compact form of $\boldsymbol{C} = \{C^1, C^2, \ldots, C^n\}$.

Note that we do not need to actually build the full matrix if space is premium, however doing this simplifies the notation and allows some optimizations ([Keogh et. al., 2006).

We can now define the *Rotation invariant Euclidean Distance* between two time series.

Definition 7. *Rotation invariant Euclidean Distance*: Given two time series *Q* and *C* of length *n*, the rotation invariant Euclidean distance between them is defined as

$$RED(Q,C) = \min_{1 \le j \le n} ED\left(Q, C^j\right)$$

The rotation invariant Euclidean distance provides an intuitive measure of the distance between two shapes, at the expense of efficiency. The time complexity to compare two time series of length *n* is $O(n^2)$.

This long discussion of rotation invariance is motivated by the observation that shapes are often rotated in real world domains. For example, Figure 5 shows two sample wing images from a collection of Drosophila images. Note that the rotation of images can vary even in such a controlled domain.

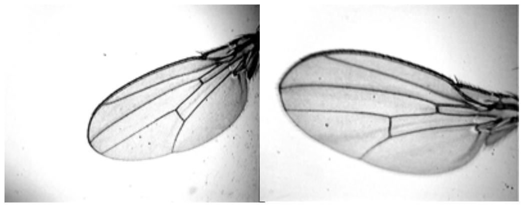


Figure 5: Two sample wing images from a collection of Drosophila images. Note that the rotation of images can vary even in such a structured domain

This distance definition can be easily generalized to handle enantiomorphic invariance (mirror image), which may be useful in some domains. For example, when matching faces, the best match may simply be facing the opposite direction. If enantiomorphic invariance is required, we can trivially achieve this by augmenting matrix ***C*** to contain C^i and reverse(C^i) for $1 \le i \le n$.

Before calling the distance function, each time series is normalized to have mean zero and a standard deviation of one, because it is well understood that it is generally meaningless to compare time series with different offsets and scales (Keogh and Kasetty, 2002). However, when dealing with time series that are derived from shapes, there is one important exception: if a shape is almost round, we should *not* do the normalization. The reason is, for nearly circular shapes, the distance from every point on the profile to the center is almost the same. So the resulting time series will be almost a flat line. However even a perfect circle will not produce a perfectly straight line due to rasterization effects. Normalization will exaggerate slight variations of the flat line, producing apparent features in the time series. We have shown such an example in Figure 6. Therefore, whenever we detect that the sum of the differences between each data point and the mean value of the time series is less than a predefined threshold, we will not do the normalization. Note that this is a very rare case that only happens in one of the seven domains we examined (see Section 5.1.2).

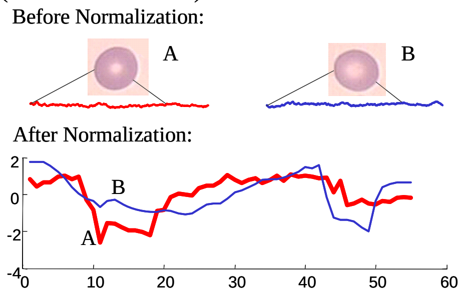


Figure 6: *Top)* The time series of two red cells are almost flat lines and are very similar to each other. *Bottom)* After normalization the two time series look very different because the tiny variances are enlarged

Before leaving this subsection, we would like to take a moment to discuss contour sampling. Dozens of papers have suggested that shape matching can be made faster by sampling the contours. For example, in Lee et. al, 2004 the authors note: "*it is first necessary to reduce the number of data points on the contour to a reasonable number that can be evaluated using shape similarity measurement*." These authors are interested in classifying fish. The fish shapes are reduced down to mere forty data points because they "*… found that a reduced data set of 40 points was sufficient to retain the important shape features for comparison*" (Lee et. al, 2004). This dramatic data reduction did make the similarity measure more tractable, but we wondered if the assumption that it "*retain(s) the important shape features*" was true. We compared their results, which after considerable parameter tuning claimed "*the highest recognition accuracy of 64%*", with the results we got by using rotation invariant Euclidean distance on the raw data. Surprisingly our simple, parameter-free method achieves 88.57% accuracy, which is *much* greater than the sampling approach. Since sampling the contours reduces accuracy greatly, in this paper we use the raw time series data.

2.2 Can Existing Approaches Help?

At this point, we have shown that shapes can be converted to time series. One might ask whether the existing time series novelty detection methods could solve the problem at hand. We believe the answer is *no*. We cannot leverage off the existing time series novelty detection techniques because most of them assume that time series subsequences are extracted by sliding a window across a long time series (Keogh et. al., 2002; Keogh et. al., 2005; Shahabi et. al., 2000), while we have *individual* time series here. Subsequences extracted from a long time series are continuous in nature. Therefore they usually have very high correlation, which can potentially be leveraged off (Keogh, 2001). In contrast, individual time series do not have this feature.

There are many different algorithms for efficiently finding distance-based outliers (Bay and Schwabacher, 2003; Ramaswamy et. al., 2000; Chen et. al., 2003; Knorr et. al., 2000; Tao et al., 2006). So another possibility would be to simply project the shape time series into n-dimensional space and use existing outlier detection methods. The problem with this approach is that most outlier detection methods require the distance function be a metric. While the Euclidean distance *is* a metric, we cannot simply embed objects in Euclidean space such that their distances are preserved under *rotation invariant* Euclidean distance. Even if we could bypass or mitigate this problem, most of the current outlier detection methods degrade to quadratic time complexity for high dimensional data. For example, Ghoting et. al. 2006 notes…"*When the data set consists of a mixture of a few distributions, with not many outlying points, ORCA's complexity is near quadratic*". Others have made similar claims (Tao et. al., 2006).

There are some work on improving the efficiency of outlier detection algorithms. One state-of-the-art approach is Ramaswamy et. al. 2000, in which the authors presented two novel algorithms to detect outliers. The first assumes the dataset is stored in a spatial index, like the R-tree, and uses it to compute the distance of each dataset object from its k^{th} nearest neighbor. Note that it takes O($m$log$m$) steps just to build the index, which is much worse than our empirical O(m) total cost. In any case, they only tested on datasets with a *maximum* of ten dimensions, because index structures break down as the dimensionality increases. One of the most recent high quality papers on the topic is Tao et al. 2006. They introduce an outlier detection algorithm called SNIF (ScaN with prIoritized Flushing). The proposed technique retrieves the outliers by scanning the dataset two or three times. In simple words, SNIF keeps a representative sample of objects of size s and compares all other objects to them. This alone needs s*m number of comparisons of two objects. In our case, each object itself is a time series of length n and the comparison of two time series takes O(n^2) time. So this approach is not suitable for the problem at hand. Furthermore, they only test on 2D, 3D and 5D data. Recall however that we have dimensionalities in the hundreds. In fairness to the authors they are focusing more on low dimensional disk resident data, and we on very high dimensional memory resident data.

The above analysis suggests that existing algorithms are of little utility for finding shape discords. This motivates the introduction of our original algorithm in the next section. Our work is closest in spirit to the unusual time series detection work of Keogh et. al. 2005, from which we take the name "discord". However, detecting unusual time series is a considerably easier problem because they do not have to deal with the rotation invariance problem, which (as we shall show below) considerably adds to the complexity of the problem.

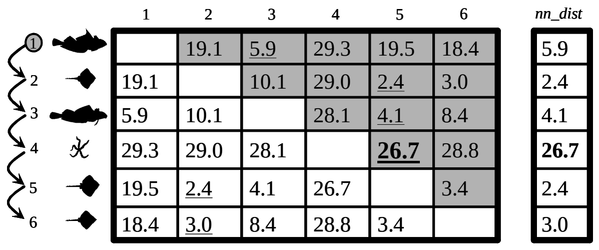
3. Shape Discords Discovery Framework

Given a shape dataset *S* of size *m*, the brute force algorithm for finding the discord is simple and obvious. We simply take each time series and find its distance to its nearest match. The one that has the greatest such value is the discord. The pseudo code of this simple algorithm is shown in Table 2.

Table 2: Brute Force Discord Discovery

```
Function [ dist, index ] = BruteForce_Search($S$)
best_so_far_dist = 0
best_so_far_index = NaN
for $p$ = 1 to |$S$|                                  // begin outer loop
  nearest_neighbor_dist = infinity
  for $q$ = 1 to |$S$|                                // begin inner loop
    if $p$!= $q$                                      // do not compare to self
      if $Dist(C_p, C_q)$ < nearest_neighbor_dist
        nearest_neighbor_dist = $Dist(C_p, C_q)$
      end
    end
  end                                                 // end inner loop
  if nearest_neighbor_dist > best_so_far_dist
    best_so_far_dist = nearest_neighbor_dist
    best_so_far_index = $p$
  end
end                                                   // end outer loop
return [ best_so_far_dist, best_so_far_index ]
```

While the brute force algorithm is intuitive, we will show a running example to develop some intuition on how to improve this algorithm. Figure 7 shows a "trace" of the brute force algorithm on a simple dataset of six marine creatures. The first discord happens to be the starfish (shape 4), whose distance to its nearest match is 26.7 (shown in bold in Figure 7). To find the discord, the brute force algorithm searches the dataset with nested loops, where the outer loop searches over the rows for each candidate shape, and the inner loop scans across the columns to identify the candidate's nearest rotation invariant match. The brute force algorithm needs to compute the distance between 6*(6-1)/2 = 15 pairs of shapes (the upper triangle of the distance matrix).

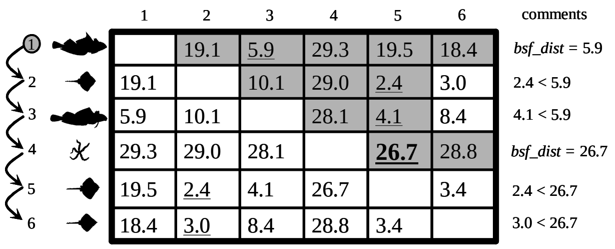


	1	2	3	4	5	6	nn_dist
1		19.1	5.9	29.3	19.5	18.4	5.9
2	19.1		10.1	29.0	2.4	3.0	2.4
3	5.9	10.1		28.1	4.1	8.4	4.1
4	29.3	29.0	28.1		**26.7**	28.8	**26.7**
5	19.5	2.4	4.1	26.7		3.4	2.4
6	18.4	3.0	8.4	28.8	3.4		3.0

Figure 7: The brute force algorithm searches from top to bottom and left to right. It needs to compute 15 cells of the distance matrix (the shaded cells). Note that cells on the diagonal are blank because we do not compare a shape to itself

The brute force algorithm is easy to implement and produces exact results. However, it has O(m^2) time complexity (recall that *m* is the size of the dataset *S*), which is simply untenable for even moderately large datasets. Note that even though the time complexity is quadratic, the space needed by the brute force algorithm is constant. We show the full distance matrix only for clarity. In actual implementation, we can build and examine only one cell at a time.

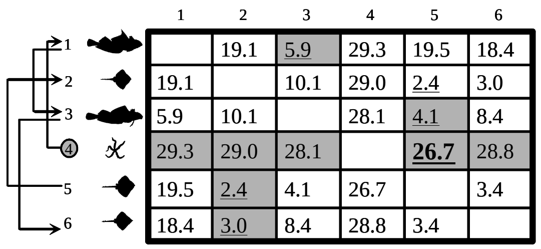
In Bay and Schwabacher 2003, the authors proposed ORCA algorithm for distance-based outlier detection. They showed that the above simple nested loops based algorithm can be modified to yield near linear time. The idea is to prune discord candidates based on the closest neighbor found so far. In other words, in the inner loop, once we find any shape that is closer to the current candidate than the best_so_far_dist variable, we can stop the search in that row, safe in the knowledge that the current candidate could not be the shape discord. If we apply this optimization to the marine creature dataset, we only need to compute the distance between 12 pairs of shapes. Figure 8 shows a trace of the search process. We call this simple optimization *early abandoning*.



	1	2	3	4	5	6	comments
1		19.1	5.9	29.3	19.5	18.4	*bsf_dist* = 5.9
2	19.1		10.1	29.0	2.4	3.0	2.4 < 5.9
3	5.9	10.1		28.1	4.1	8.4	4.1 < 5.9
4	29.3	29.0	28.1		**26.7**	28.8	*bsf_dist* = 26.7
5	19.5	2.4	4.1	26.7		3.4	2.4 < 26.7
6	18.4	3.0	8.4	28.8	3.4		3.0 < 26.7

Figure 8: Every time we find a shape having a closer distance to one of its neighbors than the best_so_far_dist value, we can stop the search. This technique reduces the number of computed cells to 12

With early abandoning, we can save some computation. The utility of this optimization depends on the order in which the outer loop considers the candidates for the discord, and the order in which the inner loop visits the other shapes. Note that there is nothing special about the top-to-bottom (outer loop), left-to-right (inner loop) ordering that we have been using. It is simply the classic default of nested "for" loops. As far as the brute force algorithm is concerned, any permutation of the orders is acceptable. However, the early abandoning algorithm *can* benefit from certain permutations. For example, imagine that a friendly oracle gives us the best possible orderings as follows. For outer loop, shapes are sorted in descending order of the distance to their nearest neighbor, so that the true discord is the first object examined. For inner loop, shapes are sorted in ascending order of the distance to the current candidate. With these orderings, the first invocation of the inner loop will run to completion. Thereafter, all subsequent invocations of the inner loop will be abandoned during the very first iteration. Returning to our running example (see Figure 9), we only need to compute the distance between 9 pairs of shapes by searching in the best orderings.



	1	2	3	4	5	6
1		19.1	5.9	29.3	19.5	18.4
2	19.1		10.1	29.0	2.4	3.0
3	5.9	10.1		28.1	4.1	8.4
4	29.3	29.0	28.1		**26.7**	28.8
5	19.5	2.4	4.1	26.7		3.4
6	18.4	3.0	8.4	28.8	3.4	

Figure 9: In a hypothetical situation where we know the best orders for outer and inner loops, the search is much more efficient. We consider the rows in the order of 4, 1, 3, 6, 5, 2. For shape 4 the inner loop runs to completion. For other shapes, the inner loops will early abandon on the first iteration. In total, only 9 cells of the distance matrix are computed (the shaded cells)

The above example motivates the introduction of a slightly expanded version of the brute force algorithm. The new algorithm is augmented by the early abandoning technique (line 7 in Table 3) and the additional outer and inner heuristics. The pseudo code is shown in Table 3.

Table 3: Heuristic Discord Discovery

```
    Function [ dist, index ] = Heuristic_Search(S, Outer, Inner )
1   best_so_far_dist = 0
2   best_so_far_index = NaN
3   for each index p given by heuristic Outer        // begin outer loop
4     nearest_neighbor_dist = infinity
5     for each index q given by heuristic Inner      // begin inner loop
6       if p!= q                                     // do not compare to self
7         if Dist(C_p, C_q) < best_so_far_dist
8           break                                    // break out of inner loop
9         end
10        if Dist(C_p, C_q) < nearest_neighbor_dist
11          nearest_neighbor_dist = Dist(C_p, C_q)
12        end
13      end
14    end                                            // end inner loop
15    if nearest_neighbor_dist > best_so_far_dist
16      best_so_far_dist = nearest_neighbor_dist
17      best_so_far_index = p
18    end
19  end                                              // end outer loop
20  return [ best_so_far_dist, best_so_far_index ]
```

There is a key difference between our algorithm and ORCA. ORCA algorithm passively assumes that the data is random. If the assumption did not hold then the performance can be very poor. Our approach actively tries to make the data *non-randomly* ordered (in a "worst first" order, to allow maximum punning). More details are given in Section 4.

Now we have reduced the discord discovery problem to a generic framework where all one needs to do is to specify the heuristics. Our goal then is to find the best possible approximation to the optimal ordering. The optimal ordering is the objects sorted in reverse order of their distance to their nearest neighbor.

4. Approximating the Optimal Ordering

When trying to approximate the optimal ordering, we need to keep one thing in mind: we should not attempt to "cheat" the algorithm. For example, we could provide very good orderings if we are allowed to compute the distances between each pair of the shapes beforehand! However this is simply hiding the time complexity in a different part of the implementation. Therefore we must insist that the outer heuristic (invoked only once) takes at most O(*m*) to calculate and the inner heuristic (invoked *m* times) takes O(1). Note that this requirement precludes the possibility of using R-trees, K-d trees or other classic indexing algorithms: they require at least O(log(*m*)) time per lookup, but we can spare only O(1) time.

With these time constraints, the problem at hand is much harder. The optimal heuristic requires a perfect ordering of shapes in the inner loop, and any perfect ordering (i.e., sorting) requires at least O(*m*log*m*), but we are only allowed O(1). Furthermore, the only known way to produce the perfect ordering of shapes in the outer loop requires O(m^2) time (i.e. run the entire brute force algorithm), but we are only allowed O(*m*) time. The following two observations, however, offer us some hope for a fast algorithm.

Observation 1: In the outer loop, we do not actually need to find a perfect ordering to achieve dramatic speedup. All we really require is that, among the first few shapes being examined, there is at least one that has a large distance to its nearest neighbor. This will give the best_so_far_dist variable a large value early on, which will make the conditional test on line 7 of Table 3 be true more often, thus allowing more early abandonment of the inner loop.

Observation 2: In the inner loop, we also do not actually need to find a perfect ordering to achieve dramatic speedup. All we really require is that, among the first few shapes being examined there is at least one that has a distance to the candidate that is less than the current value of the best_so_far_dist variable. This is a sufficient condition to allow early termination of the inner loop.

Based on these two observations, we propose an algorithm that uses locality-sensitive hashing to estimate similarity between pairs of shapes, and then generates heuristics to order the outer and inner loops.

4.1 Symbolizing the Time Series

The first step of our approximation algorithm is the symbolization of time series. Symbolization serves several important purposes. First, it provides us with a lower dimensional representation that reduces the noise effect in the raw time series while preserving its properties. Second, it gives us a string representation that will be used in the subsequent step by the location-sensitive hash function.

There are many different symbolic approximations of time series in the literature (Andre-Jonsson and Badal, 1997; Daw et. al., 2002; Huang and Yu, 1999). In this work, we choose the **S**ymbolic **A**ggregate Appro**X**imation (SAX) representation introduced by Lin, et al., 2003, because it allows both dimensionality reduction and lower bounding. Below, we give a brief review of the SAX representation. We start with the Piecewise Aggregate Approximation (PAA) (Keogh et. al, 2001).

Definition 8. *Piecewise Aggregate Approximation (PAA)*: Given a time series $C = c_1, c_2, ..., c_j, ..., c_n$ and the desired lower dimensionality *w*, the Piecewise Aggregate Approximation of time series *C* is a *w*-dimensional vector $\overline{C} = \overline{c}_1, \ldots, \overline{c}_w$ where

$$\bar{c}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j$$

In other words, the time series is divided into segments of equal length and each segment is substituted by its mean value.

Having the PAA representation, we can apply a further transformation to obtain a discrete representation. It is desirable to have a discretization technique that will produce symbols with equiprobability. This is easily achieved since normalized time series have highly Gaussian distributions (Lin, et al., 2003). We can simply determine the "breakpoints" that will produce equal-sized areas under a Gaussian curve.

Definition 9. *Breakpoints*: Breakpoints are a sorted list of numbers $B = \beta_1,\ldots, \beta_{|\Sigma|-1}$, where $|\Sigma|$ is the size of the alphabet, such that the area under a $N(0,1)$ Gaussian curve from β_i to $\beta_{i+1} = 1/|\Sigma|$ (β_0 and $\beta_{|\Sigma|}$ are defined as $-\infty$ and ∞, respectively).

These breakpoints may be determined by referring to a statistical table. For example, Table 4 gives the breakpoints for values of $|\Sigma|$ from 3 to 6.

Table 4: A lookup table that contains the breakpoints for dividing a Gaussian distribution into an arbitrary number (from 3 to 6) of equiprobable regions

β_i \ $|\Sigma|$	3	4	5	6
β_1	-0.43	-0.67	-0.84	-0.97
β_2	0.43	0	-0.25	-0.43
β_3		0.67	0.25	0
β_4			0.84	0.43
β_5				0.97

Once the breakpoints have been obtained, we can assign the same letter to all PAA coefficients that belong to the same interval. For example, all PAA coefficients that are below the smallest breakpoint are mapped to the symbol "**a**", all coefficients greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol "**b**", etc. Figure 10 illustrates this idea.

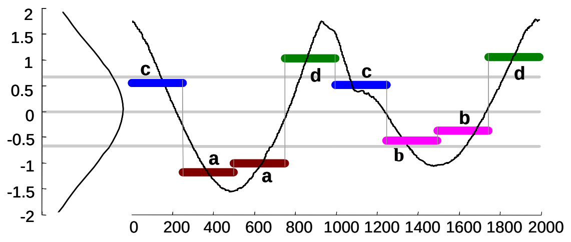


Figure 10: A time series (thin black line) is discretized by first obtaining a PAA approximation (heavy gray line) and then using predetermined breakpoints to map the PAA coefficients into symbols (bold letters). In the example above, with n = 1992, w = 8, and $|\Sigma|$ = 4, the time series is mapped to the word **caadcbbd**

Note that in this example, the four symbols, "**a**", "**b**", "**c**", and "**d**" are equiprobable, as desired. We call the concatenation of symbols that represent a time series a *word*.

Definition 10. *Word*: A time series C of length n can be represented as a *word*

$\hat{C} = \hat{c}_1, \ldots, \hat{c}_w$. Let α_i denote the i^{th} element of the alphabet, i.e., α_1 = **a** and α_2 = **b**. Then the mapping from a PAA approximation $\bar{C}$ to a word $\hat{C}$ is obtained as follows:

$$\hat{c}_i = \alpha_i \quad \text{iff} \quad \beta_{j-1} \le \bar{c}_i < \beta_j$$

By converting time series to SAX words, we reduce the dimensionality of time series. Clearly some information is lost during the conversion. To measure how well the SAX representation approximates the original time series, we define SAX reconstruction error. While the term "reconstruction error" is well defined for other representations such as wavelets and Fourier approximations, it is not generally used for symbolic representations of discrete data. Here we show that we can quantitively measure the reconstruction error of SAX representation.

Definition 11. *SAX Reconstruction Error*: Given a time series $C = c_1, c_2, \ldots c_n$ and its SAX word $\hat{C} = \hat{c}_1, \ldots, \hat{c}_w$, the SAX Reconstruction Error is the sum of the distance between each data point in the time series and the middle line of the SAX symbol that the data point maps, or more formally as:

$$error = \sqrt{\sum_{i=1}^{n} (c_i - \beta_{\frac{2w}{n}(i-1)+1})^2}$$

where c_i is the i^{th} data point of the time series, $\alpha_{\frac{w}{n}(i-1)+1}$ is the SAX symbol that c_i maps to, and $\beta_{\frac{2w}{n}(i-1)+1}$ is the value that divides the region of SAX symbol $\alpha_{\frac{w}{n}(i-1)+1}$ into two equiprobable parts.

For example, in Figure 11, a time series is converted into SAX word **cadcac**. The alphabet size is four. According to Table 4, the breakpoints are (-0.67, 0, 0.67), as shown by the left Y-axis of Figure 11. The values that divide each symbol region to two equiprobable parts are (-1.15, -0.32, 0.32, 1.15), shown in the right Y-axis of Figure 11. These can again be looked up from Table 4 since dividing each region (the shaded area in Figure 11) to two parts is equivalent to doubling the alphabet size to eight. The SAX Reconstruction Error is the square root of the sum of the square lengths of the purple hatch lines shown in Figure 11.

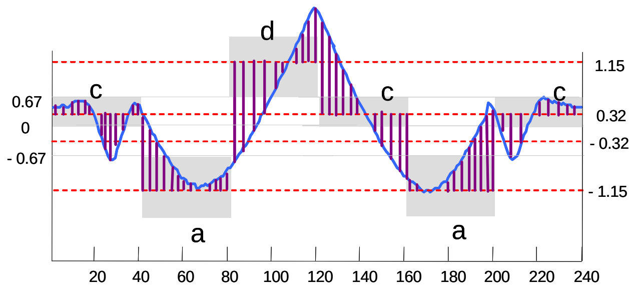


Figure 11: A visual illustration of SAX reconstruction error. The reconstruction error is calculated as the sum of the distance between each data point and the middle line (the red dot line) of the SAX symbol that the data point maps to

In Figure 4 we have shown that we can convert any time series into a SAX word with a simple "unwinding" process. Note that the starting point for this process is completely arbitrary. This observation allows an optimization, because it may happen that some of the arbitrary starting points will lead to better SAX approximations.

For example, assume we have an arrow image. We convert it to time series starting from two different points (one is 15 degrees clockwise than the other). The resulting time series and the corresponding SAX representation are shown in Figure 12.

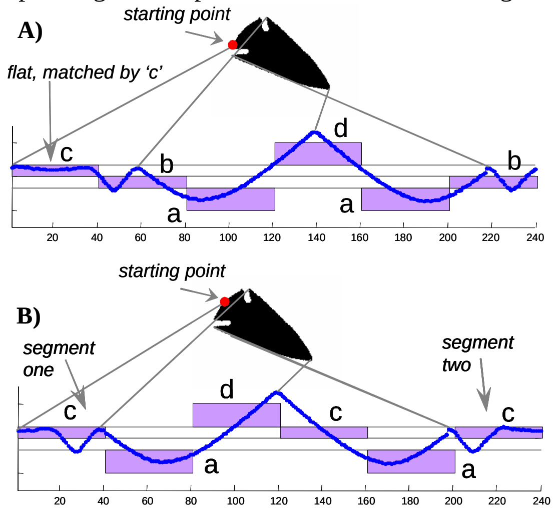


Figure 12: An arrowhead image is converted to time series starting from two different points. *Top*) The first SAX symbol **c** approximates the first 40 data points perfectly. *Bottom*) However the same plateau time series is divided into two parts (the first and last segments of the time series)

At the first sight, the two SAX representations are quite similar. However, note that with the first starting point (see Figure 12.A), the first symbol **c** of the resulting SAX word matches perfectly with a plateau in the time series, while starting from the second starting point (see Figure 12.B), this plateau segment spreads across two segments (the first and the last segments). Intuitively, we may expect that the SAX word **cbadab** gives better approximation than bottom one **cadcac**. In fact, this is the case; the reconstruction errors are 106.35 and 144.65 respectively. Based on this observation, every time we convert a shape time series into a SAX word, we test all possible circular shifts of the time series and choose the one that has the smallest reconstruction error. We apply this optimization throughout the paper.

We have now completely defined SAX representation. Note that the tendency of time series to have Gaussian distributions (Lin, et al., 2003) is not critical to the *correctness* of any algorithms that use SAX, including those in this work. A pathological dataset that violates this assumption will only affect the *efficiency* of the algorithms.

4.2 Locality-Sensitive Hashing

As we noted earlier, to give relatively good outer and inner orders, we need to quickly approximate the similarities between all shapes. Estimation of similarity based on sparse sampling of positions from feature vectors has been used in diverse areas for different purposes, including high-dimensional search (Narayanan and Karp, 2004), multimedia indexing (Van Otterloo, 1991), and motif discovery (Tompa and Buhler, 2001), etc. Among the rich literature, the locality-sensitive hashing search technique proposed by Indyk et. al., 1997 is perhaps the most referenced in this area. Since this technique is a cornerstone of our contribution, we give the formal definition of *locality-sensitive hashing* below.

Definition 12. *Locality-sensitive Hash Function*: Consider a string s of length w over an alphabet Σ and k indices $i_1, \ldots, i_k$ chosen uniformly at random from the set $\{1, \ldots, w\}$. Define the locality-sensitive hash function $f : \Sigma^w \to \Sigma^k$ by

$$f(s) = \langle s[i_1], s[i_2], \ldots, s[i_k] \rangle$$

In other words, the locality-sensitive hash function concatenates characters from, at most, k distinct positions of string s. The resulting length k string is called an *LSH value*. Clearly, strings similar to each other are more likely to be hashed to the same LSH value. This is the most important property of locality-sensitive hashing, which enables efficient search, indexing, and many other works (Indyk et. al., 1997).

Unfortunately, this property does not hold for shapes because of the rotation variance. For example, the two arrowheads in Figure 13 are quite similar but differently aligned. Their time series representations are shifted by some offset and the resulting SAX words are completely different. They will not be hashed to the same LSH value no matter which k positions are chosen.

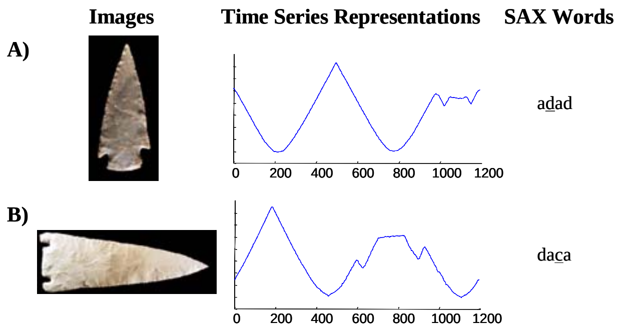


Figure 13: Image A) and B) are very similar to each other, but have different orientations. The time series extracted from the two images look different (shifted by some offset). Note that the corresponding SAX words are not only shifted but also different by one symbol (the underscored position)

Considering the above problem, we define a *rotation invariant locality-sensitive hash function*.

Definition 13. *Rotation invariant Locality-sensitive Hash Function*: Consider a string s of length w over an alphabet Σ and k indices $i_1, \ldots, i_k$ chosen uniformly at random from the set $\{1, \ldots, w\}$. Define the rotation invariant locality-sensitive hash function f': $\Sigma^w \to (\Sigma^k)^w$ by

$$f'(s) = \{\langle p[i_1], p[i_2], \ldots, p[i_k] \rangle \mid p \in LSHIFTS(s)\}$$

where $LSHIFTS(s)$ is the set of all possible left shifts of string s.

The rotation invariant locality-sensitive hash function maps a string s to a set of length k strings, each of which is the LSH value of one shift of s. By doing this, similar shapes (even with different orientations) are more likely to be mapped together to some LSH value. For example, consider the same two images in Figure 13. Using rotation invariant hashing, image A is mapped to {aa, dd} and image B is mapped to {dc, aa, cd}. They have LSH value “aa” in common. The mapping process is shown in Figure 14.

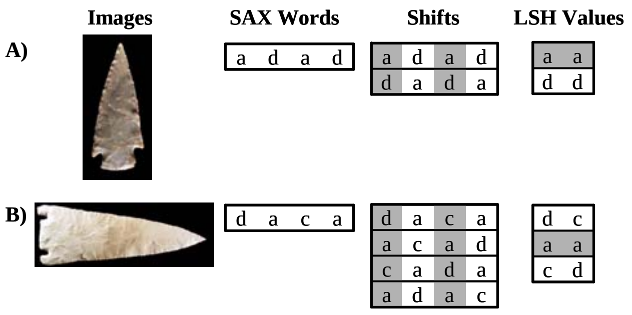


Figure 14: The same two images as in Figure 13 are both mapped to LSH value “aa”. Here w = 4, Σ = {a, b, c, d}, and k = 2. The indices chosen by the hash function are {1, 3}

4.3 Similarity Estimation and Optimal Ordering Approximation

At this point, we are ready to present our algorithm of estimating the similarity among shapes and approximating the optimal ordering of early abandoning search.

Suppose we have a dataset S of m shapes (each has been converted to a time series of length n), the SAX word size w, and the SAX alphabet Σ. We begin by converting all time series to SAX words and placing them in an array. Note that each row index of the array refers back to the original shapes. Figure 15 gives a visual intuition of this, where both w and |Σ| are set to 4. Note that in spite of the redundancy of multiple rotations, the size of the array is much smaller than the time series data and inconsequential in size compared to the raw images.

Once the array has been constructed, we randomly choose a rotation invariant locality-sensitive hash function and use it to hash SAX words into buckets. For example in Figure 15, we choose indices {1, 3}. Therefore the SAX words in the array are hashed into buckets based on the first and third columns of their shifts: first SAX word *daca* is hashed to buckets *dc*, *aa*, and *cd*; fourth SAX word *adad* is hashed to buckets *aa* and *dd*; so on and so forth. If two shapes corresponding to SAX words i and j are hashed to the same bucket, we increase the count of cell(i, j) in the collision matrix by one, which has been initialized to all zeros. In our example, we increase the count of cell(1, 4) and cell(4, 1).

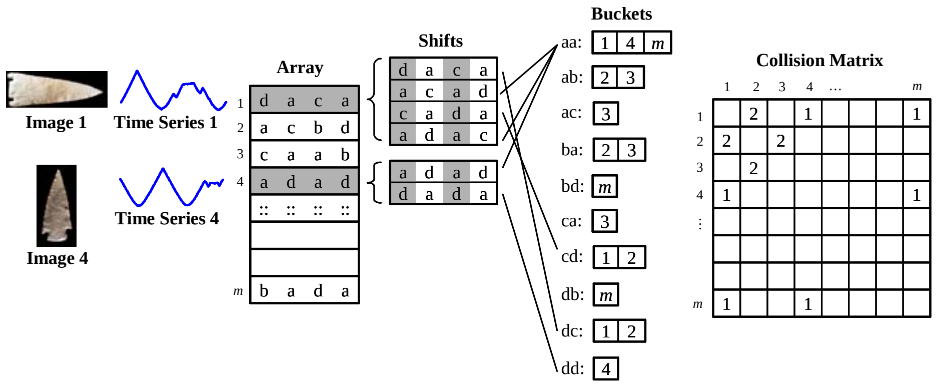


Figure 15: Illustration of the similarity estimation process. *Left)* The time series extracted from images are converted to SAX words and then inserted into an array. *Middle)* A hash function maps SAX words into buckets. *Right)* Collisions are recorded by incrementing the corresponding cells in the collision matrix

Note that the words corresponding to shapes 2 and 3 are also hashed into a common bucket *ba*, yet they are not similar to each other (even after considering all rotations). This problem can be solved simply by repeating the hashing process a number of times, each time with a new, randomly chosen hash function.

The above process of time series extracting, discretizing, hashing, and collision recording is formalized in Table 5. Note that the collision matrix is initialized to all zeros and is being accumulated throughout the entire process, while the hash buckets cannot be reused from one iteration to another.

At a first glance, the collision matrix appears to be quite demanding in terms of space requirements. In general, however, we can expect it to be *extremely* sparse, and thus worth the slight time overhead to implement it as a sparse matrix. In the worst case, the number of cells in the collision matrix which has a non-zero entry is $m*w*i$, where m is the number of shapes in the dataset, w is the SAX word size, i is the number of iterations. As shown by us in next subsection and by many other research works (Chiu et. al., 2003, Yankov et. al., 2005), the number of iterations is on the order of 10 to 100 and the SAX word size is usually a small value (for example, we use w = 20 in this work). Thus in practice, the size of the sparse collision matrix is linear in m.

Table 5: Hash-based Optimal Order Approximation

```
    Function [Outer, Inner] = Optimal_Approximation (S, n, w, Σ)
1   convert each shape in S to time series of length n
2   convert time series to SAX word with length w and alphabet Σ
3   insert each SAX word to array A
4   initialize collision matrix CM to all zeros
5   while (not stopping)
6     randomly choose a rotation invariant locality-sensitive hash function f'
7     for each SAX word in array A
8       apply f' on the SAX word
9       insert the index of the SAX word to all buckets it maps to
10    end
11    for all pairs (i, j) in the same bucket
12      CM (i, j) ++
13    end
14    delete all buckets
15  end
16  generate Outer and Inner heuristics based on CM
17  return [ Outer, Inner]
```

After repeating the process an appropriate number of times, we examine the collision matrix. If two shapes are similar, we expect the corresponding cell in the collision matrix to have a large value. If two shapes are different, the collision matrix tells us nothing. However, we can infer from the lack of a value in the collision matrix that two shapes are probably different. So we can use collision matrix as a guideline to decide the outer and inner orderings (line 16 in Table 5).

Outer Heuristic: For each shape, we scan the collision matrix to find the largest number of collisions it has with others. Then we sort the shapes in ascending order using this value. The resulting ordering is given to the outer loop. Here we can take the advantage of the sparsity of the collision matrix. As we will show in next subsection, the time complexity of this step is linear in *m*.

The intuition behind our outer heuristic is that unusual shapes are very likely to have fewer collisions with others. By considering the candidate shapes in the ascendant order of the number of collisions they have, we have an excellent chance of giving a large value to the best_so_far_dist variable early on, which (as noted in observation 1) will make the conditional test on line 7 of Table 3 be true more often, thus allowing more early abandonment of the inner loop.

Inner Heuristic: When candidate shape *i* is considered in the outer loop, the inner loop examines the shapes in the descending order of the number of collisions they have with shape *i*.

The intuition behind our inner heuristic is that shapes which frequently collide with each other are very likely to be highly similar. This fact is at the heart of more than twenty research efforts (Bentley and Sedgewick, 1997; Chiu et. al., 2003; Keogh et. al., 2004; Kitaguchi, 2004; Lin et. al., 2004; Tanaka and Uehara, 2004). As noted in observation 2, we just need to find one such shape that is similar enough (having a distance to the candidate less than the current value of the best_so_far_dist variable) to terminate the inner loop.

There are several minor optimizations we can apply to the heuristic search algorithm. For example, imagine we are considering candidate C_i in the outer loop, and as we traverse through the inner loop, we find that time series C_j is close enough to it to allow early abandonment. In addition to saving time with the early abandonment, we can also delete C_j from the list of candidates in the outer loop (if it has not already been visited). The key observation is that since we are assuming a symmetric distance measure, if nearness to C_j disqualifies candidate C_i from being the discord, then the same nearness to C_i would also disqualify candidate C_j from being the discord. Empirically, this simple optimization gives a speedup factor of approximately two. In addition, there are several well-known optimizations to the Euclidean distance (Keogh and Kasetty, 2002) and some special optimizations to rotation invariant Euclidean distance (Keogh et. al., 2006) that we can use.

4.4 Time Complexity and Parameter Settings

Recall that we require the outer heuristic take at most O(m) time to calculate and the inner heuristic take O(1). We will now show that our optimal order approximation algorithm fulfills these requirements.

As explained above, our algorithm has three steps. Keep in mind that in our time analysis**, the atomic operation is the scan of one time series of length *n***, which itself can be considered O(n) if the basic operation is to read t_i in time series $<t_1, t_2, \ldots, t_n>$. The first step of our algorithm is to convert shape time series to SAX words, requiring only one pass through the time series. So the time for this step is linear in the size of the dataset, m. Secondly, we apply rotation invariance hash functions on SAX words to separate them into buckets. This again only requires one pass through the SAX words for each iteration. The basic operation here is the scan of one SAX word (which is of length w), which takes less time than the *atomic operation* (because clearly $w << n$). So it is safe to say that the time complexity for this step is O(m). In the third step, we pair the indices in the same buckets and update the collision matrix accordingly. As we explained in Section 4.3, in practice, the number of the pairs, which is equal to the number of non-zero entries in the collision matrix, is linear in m. Also note that the basic operation here is the scan of one bucket, which takes much less time than our *atomic operation*. So the time complexity for this step is again O(m). Since each of the steps takes O(m), the time complexity of the optimal order approximation step is O(m).

For the optimal order approximation algorithm, we must choose three parameters: the SAX alphabet size $|\Sigma|$, the SAX word size w, and the number of iterations i. Recall that we would like the shape discords hash to some rarely used buckets, while all other shapes hash to popular buckets. If we choose very large values for $|\Sigma|$ and/or w, almost all shapes will map to unique words; if we choose very small values for $|\Sigma|$ and/or w, all shapes will map to just a small handful of words. Either of these situations is bad for separating shapes into buckets.

The good news is that there is little freedom for the $|\Sigma|$ parameter. Extensive experiments carried out by the current authors (Chiu et. al., 2003; Keogh et. al., 2002; Keogh et. al., 2004; Keogh et. al., 2005) and dozens of other researchers worldwide (Bentley and Sedgewick, 1997; Kitaguchi, 2004; Rombo and Terracina, 2004; Tanaka and Uehara, 2004) suggest that a value of either 3 or 4 is best for virtually any task on any dataset. After empirically confirming this on the current problem with experiments on more than 50 datasets, we will simply hardcode $|\Sigma| = 4$ for the rest of this work. Having fixed $|\Sigma|$, we performed an exhaustive empirical examination of the role of the *w* parameter. The best value for this parameter depends on the data. In general, relatively smooth and slowly changing datasets favor a smaller value of *w*, whereas more complex time series favor a larger value of *w*. Empirical studies show that the speedup does not critically depend on *w* parameter. We will simply use *w* = 20 for the rest of this work.

For the setting of the number of iterations *i*, there are three possibilities. We can do a fixed number of iterations, or we can stop when the user is not willing to wait more time. Since the algorithm keeps the best_so_far_index variable, it always has a candidate discord to show at any point of time after the collision matrix has been built. This actually makes it an anytime algorithm (Grass and Zilberstein, 1996). Or we can stop when the collision matrix "converges" (the change of the values of its entries are less than some threshold). Again, for simplicity, we fix the number of iterations to 30 in this work.

At the risk of redundancy, we emphasize once again that our algorithm produces exact answers. None of these parameters affects the *correctness* of the algorithm. Setting these parameters inappropriately will only affect the *efficiency* of the algorithm.

Note that so far we have only explained the algorithm to find the top one discord from a dataset. Since we have shown that our top one discord discovery algorithm is (empirically) linear, it is obvious and trivial to find the top *k* discords. We can find the top one discord and remove it from the dataset, then run the algorithm again to find the new discord, which is of course the second discord relative to the original dataset. We repeat the process again and again until we find the top *k* discords.

5. Empirical Evaluations

We begin this section by showing the utility of the shape discords in diverse domains, and continue by demonstrating that the algorithm can find discords very efficiently using the approximate optimal heuristic. For all the experiments in this work, we use rotation invariant Euclidean distance to measure the distance between two shapes.

5.1 The Utility of Shape Discord

To demonstrate the usage of shape discords in real world applications, we conducted discord searches on datasets from diverse domains.

5.1.1 Butterfly Wings Dataset

Butterfly wings are an interesting domain in which to test image mining algorithms. Depending on the area of research, the shape, color, texture or even fractal dimension of the wings may be of interest (Castrejon-Pita et. al., 2005). Here we restrict our attention to shape. The large size of such collections motivates the use of scalable algorithms. For example, the Morphbank archive (Morphbank, 2006) currently has approximately 2,000 butterfly images online, and many lepidopterists (butterfly collectors) have much larger personal collections.

Consider the image dataset in Figure 16, which purportedly shows a (subset of) collection of *Heliconius erato* (Red Passion Flower) Butterflies.

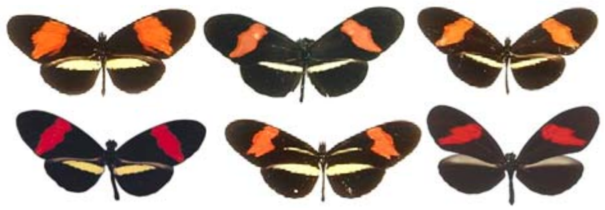


Figure 16: Six examples of butterflies, ostensibly all *Heliconius erato* (Red Passion Flower) Butterflies

We ran our algorithm on the entire collection to find the most unusual butterfly, which happens to be the one shown in the lower right of Figure 16. We ask entomologist Dr. Agenor Mafra-Neto to explain the result. In fact, the butterfly in question is *not* an example of *Heliconius erato*, it is an example of *Heliconius melpomene* (The Postman). The uncanny resemblance of the two is not a coincidence, but an example of Müllerian mimicry. In brief, it is believed that the two species originally looked different, and (perhaps independently) evolved the defense mechanism of tasting unpalatable. Being foul tasting is only useful if you advertise the fact, and once one species had evolved the orange/red flashes to communicate this to predators, the other species leveraged off the predators' avoidance by mimicking their appearance. Figure 17 clearly shows why the one butterfly was singled out from the collection.

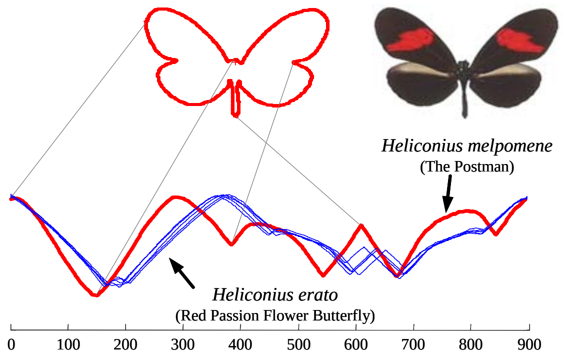


Figure 17: The six butterflies from Figure 16 shown in their time series representations. One butterfly, shown with a bold line, is a different species to the rest

5.1.2 *Red Blood Cell Dataset*

We ran some tests on images extracted from red blood cell data. Figure 18 shows a subset of an image. The discord discovered is a teardrop shaped cell, or *dacrocyte*. Such cells are indicative of several blood disorders, including myelofibrosis, metaplasia and anemia. Note that for those cells which are almost round, we did not normalize the time series derived (cf Figure 6).

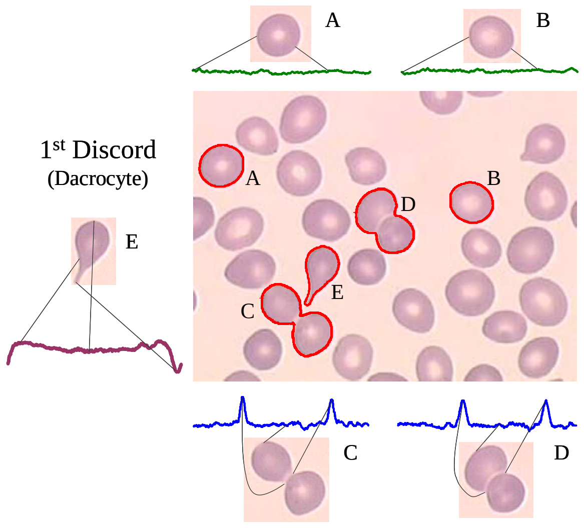


Figure 18: The discord discovered in an image of red blood cells is a teardrop shaped cell, or *dacrocyte*, which is indicative of several blood disorders

5.1.3 Fungus Dataset

We perform similar experiments on images taken at even higher resolutions. Figure 19 shows an image taken with a scanning electron microscope. The image shows some spores produced by a rust (fungus) known as Gymnosporangium, which is a parasite of apple and pear trees. Note that one spore has sprouted an “appendage” known as a germ tube, and is thus singled out as the discord.

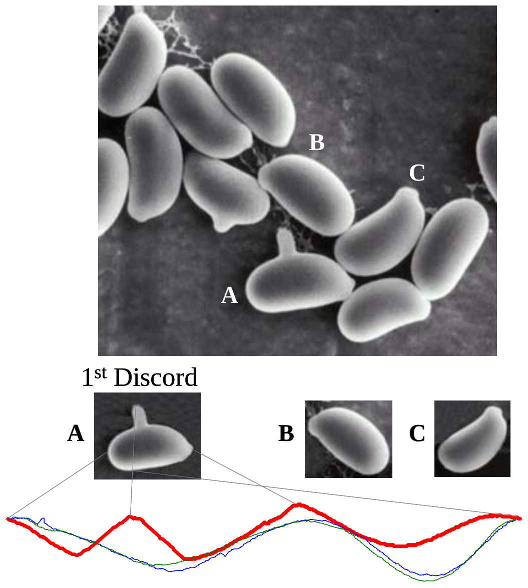


Figure 19: Some spores produced by a fungus. One spore is different because it has a germ tube (Image by Charles Mims, University of Georgia)

Note that the spore above and to the left of the discord is *just* beginning to start sprouting a germ tube. If it also had a full germ tube, then the discovered discord would not longer be so far away from its nearest neighbour (the "*twin freak problem*", discussed in Section 2.1). As noted earlier we could mitigate this problem by a simple change in the definition of discord.

5.1.4 Preliminary Results on a Massive Nematode Database

Our final example of the utility of shape discords is motivated by a joint project we have undertaken with the Department of Nematology at the University of California-Riverside. Nematodes are (typically) microscopic worms, and are one of the most common phyla of animals, with more 20,000 different species described, and an untold number of species yet to be described. Nematodes are very important for a variety of reasons: A handful of them are beneficial to mankind, for example by killing garden pests. Others, such as Root-knot nematodes (genus *Meloidogyne*) attack plants and cause millions of dollars in damage each year. More insidiously, nematodes can cause severe medical problems for humans. For example, toxocariasis is caused by a parasitic nematode, and results in permanent blindness for thousands of people each year. The Department of Nematology at the University of California-Riverside is leading a long term effort understanding the diversity of nematodes, and constructing an identification key which can be used by researchers, agriculturists, and medical professionals worldwide. This effort requires a massive commitment to microscopy, and they have already spent thousands of hours capturing images, such as the six examples of *Carcharolaimus* shown in Figure 20.

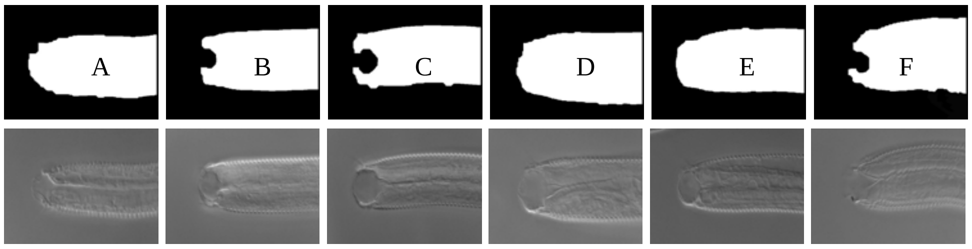


Figure 20: *Bottom*) Six examples of *Carcharolaimus* nematodes. *Top*) A binary segmentation of the images appears to reveal two distinct types {A,D,E} and {B,C,F}, but this is an artifact caused by the difficulty of image segmentation of almost transparent objects

Such massive collections of images cry out for automatic analysis, including the detection of unusual shapes. For example, nematodes can be sexual or asexual, and if sexual they can be highly sexually dimorphic. In some case only females have been found, and it is not known if males even exist. While we have only just begun to turn our attention to nematodes, initial experiments suggest our shape discord framework may be of utility. Consider Figure 21, which shows another example of *Carcharolaimus*, highlighted as the 1st shape discord by our algorithm.

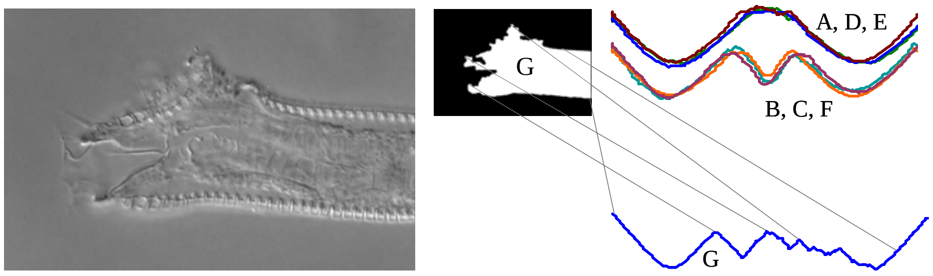


Figure 21: *Left*) Another example of a *Carcharolaimus*, which is the 1st discord found by our algorithm. *Right*) A comparison of its one-dimensional representation with those of the six *Carcharolaimus* shown in Figure 20 explains why it was singled out

This example has a burst epidermis (a layered cuticle made of keratin). Because nematodes are always under hydrostatic pressure (this is the reason they are round), the burst epidermis has allowed internal features to vent, changing the nematodes shape. While we have yet to turn up a discovery of a new species, our experiments suggest that at the very least our work could have utility for data cleaning. We plan to revisit the problem after making further progress in the image segmentation step.

5.2 The Utility of Heuristic Ordered Search

We compare the performance of the discord discovery algorithm using our approximate optimal heuristic, the random heuristic (deciding the orders of outer and inner loops randomly), and brute force search. Note that the random heuristic can actually be seen as the ORCA algorithm (Bay and Schwabacher, 2003), a simple algorithm that is shown to be very effective for distance-based outlier detection.

The measurement we use is the number of times that the distance function is called on line 7 in Table 3. A simple analysis of the pseudo code (confirmed with a profiler) tells us that this single line of code accounts for more than 99% of the running time for the algorithm. This metric is implementation-free so it avoids the bias introduced by examining wall clock or CPU time, a problem noticed by many researchers (Keogh and Kasetty, 2002; Keogh et. al., 2005; Vlachos et. al., 2005).

For our approximate optimal heuristic, we include a startup cost of O(*m*), which is the time complexity required to build the collision matrix (cf. Section 4.4). For brute force search, the number of times that the distance function is called depends only on *m* and can simply be *computed* (recall *m* is the size of the dataset). If we had to actually *run* the brute force search for all the experiments in this work, it would take several years.

We first tested a homogeneous dataset of 10,000 projectile point images. The time series derived are all of length 251. The results are shown in Figure 22. Each time we use a subset of the database (the size varies from 50 to 10,000) and measure the number of distance function calls required by each strategy, divided by the number of calls required by brute force. We can see that the cost of building the collision matrix is dwarfed by rotation invariant comparisons. The approximate optimal heuristic is faster even for small dataset of size 50. As we expect, the approximate optimal heuristic performs better as the size of the dataset increases. By the time we have examined the entire database, our approximate optimal heuristic is one order of magnitude faster than the random heuristic and several orders of magnitude faster than brute force.

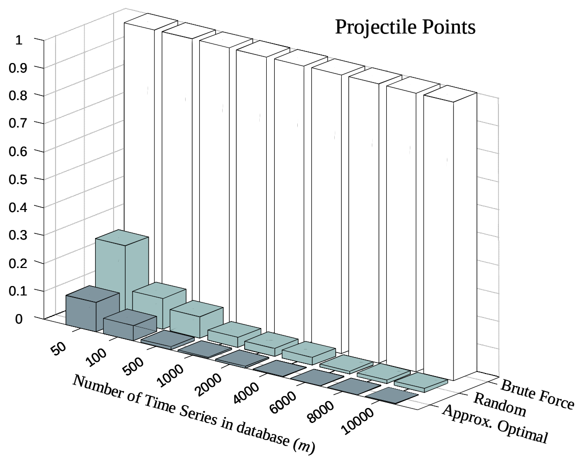


Figure 22: The relative performance for three heuristic strategies on the Projectile Points dataset

Sometimes techniques that work well for highly homogeneous datasets do not work well for heterogeneous datasets, and vice versa. We consider this possibility by testing on a heterogeneous dataset in Figure 23. The heterogeneous dataset consists of all the data used in the classification experiments (cf. Table 1), plus 1,000 projectile points. In total, it contains 5,844 images and the derived time series are of length 512. In this dataset, it takes our approximate optimal heuristic slightly longer to beat random heuristic. However by the time we have seen 200 objects, we have already broken even and thereafter rapidly race towards beating random heuristic by one order of magnitude and brute force search by several orders of magnitude.

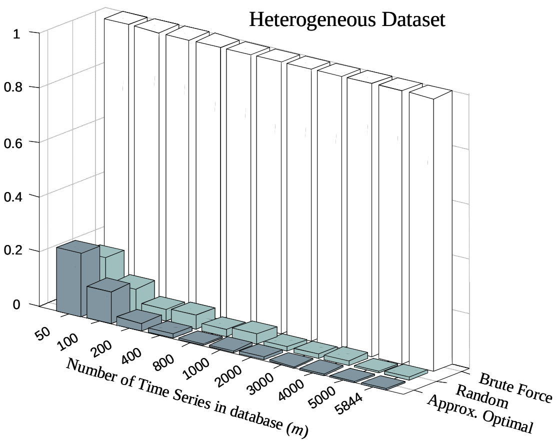


Figure 23: The relative performance for three heuristic strategies on the Heterogeneous dataset

In our last experiment, we make extra efforts comparing with the approach proposed in Angiulli et. al. 2006. We choose this paper because the authors have half dozen papers on the topic of distance-based outlier detection and are well referenced. Angiulli et. al. introduced the concept of outlier detection solving set, a subset of the input data set representing a model that can be used to predict outliers. They also provided algorithms to compute the solving set and the top *n* outliers with subquadratic time.

We tested on 10,000 randomly generated time series, each of length 256. By using randomly generated data, we are trying to mimic the *worst* case of our algorithm – when the data has no structure. Also, we deliberately make all data of length 256, because later we will use Fast Fourier Transform (FFT) to reduce the dimensionality and FFT is fastest if the objects length is a power of two.

We first tried all approaches on the raw data with no dimensionality reduction. The time taken by the solving set approach is actually worse than the brute force algorithm for all database sizes. In fairness to the authors, they would have predicted this. Their algorithm scales well in the number of instances, but not so well for dimensionality.

We next tested by reducing the dimensionality with FFT. A detailed discussion of how this can be done can be found in Keogh et. al. 2006. We had to tweak the solving set algorithm in a few ways to make this work, since they are expecting real data, not a lower bounding, low dimensional approximation of it. Also, the solving set algorithm needs several input parameters, for example, the number of neighbors to consider, an integer, and a rational number etc. We searched over all possible values of the input parameters and reported only the best result. Figure 24 shows the results averaged over 50 runs. For trivially small datasets our approach and the solving set approach are about the same (because both are dominated by "setup time"). But for larger datasets we are over an order of magnitude faster and the gap is widening fast.

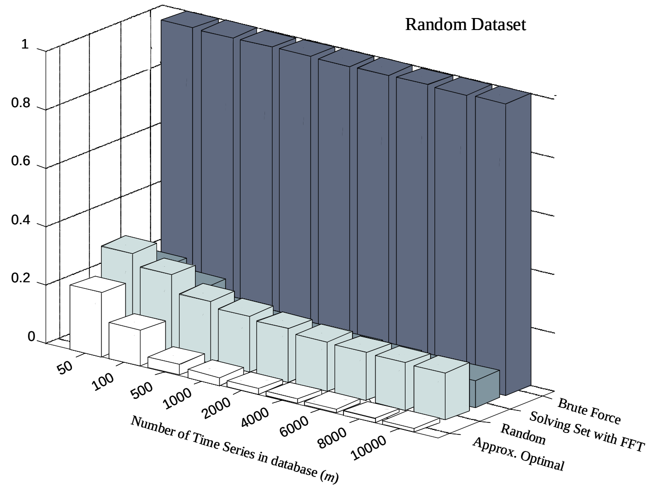


Figure 24: The relative performance for three heuristic strategies and Solving Set algorithm on a random dataset

Note that we had to "cheat" for the rival approach. We pretended that the results it gave on the reduced dimensionality were the same as the results on the raw data, whereas we should have really coded up an extra step that verifies/adjusts the answers with some accesses to the raw data (in the spirit of the GEMINI framework of Faloutsos et. al., 1994). Since without this step the approach is already slower than ours, we did not bother to code this.

As a final sanity check, we also measured the wall clock time of our best implementation of all methods. The results are essentially identical to those shown above, and are omitted in the sake of brevity.

6. Conclusions

In many applications, it can be useful to discover the most unusual shape in a collection of images. In this work, we introduce a novel definition of such shapes: the discord. This definition is particularly attractive to data mining applications because it is parameter-free. In addition, we propose an efficient algorithm to discover the shape discords. The algorithm uses locality-sensitive hashing to estimate similarity between pairs of shapes and generates heuristics to reorder the search in a more efficient way. On real problems, our algorithm is three to four orders of magnitude faster than the brute force algorithm.

There are many directions in which this work may be extended. We intend to investigate image discords not only using shapes but also textures. In addition, we plan to conduct a field study of shape discord discovery in anthropology and archeology. Finally, our shape representation works best for relatively simple rigid shapes. However, some domains of interest may have shapes with parts that articulate, for example a human silhouette has a relatively rigid trunk, but the arms and legs may move in almost arbitrary ways. Such shapes are best matched with graph based representations such as shock graphs (Siddiqi et al. 1998), we are beginning to consider definitions of outliers for such objects, and if successful we will attempt to create speedup mechanisms to make outlier discovery tractable in these domains.

7. Acknowledgement

All images are used with permission, and remain copyright of their respective owners. Thanks to Dr. Stefano Lonardi; Dr. Michail Vlachos; Dr. Agenor Mafra-Neto (entomology); Biosciences Electron Microscopy Facility, University of British Columbia; Dr. Leslie A. Quintero and Dr. Philip J. Wilke (projectile points); Karolina Maneva-Jakimoska (Morphbank); Jill Brady; Daniel von Dincklage. We also thank the anonymous reviewers for their detailed and helpful comments.

8. References

Adamek, T. and O'Connor, N.E. 2004. A multiscale representation method for nonrigid shapes with a single closed contour. IEEE Circuits and Systems for Video Technology, 14(5):742-753.

Andre-Jonsson, H. and Badal, D. 1997. Using signature files for querying time-series data. Proc. of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery, Trondheim, Norway, pp. 211-220.

Angiulli, F., Basta, S., and Pizzuti, C. 2006. Distance-based detection and prediction of outliers. IEEE Transactions On Knowledge and Data Engineering, 18(2): 145-160.

Bentley, J. L. and Sedgewick, R. 1997. Fast algorithms for sorting and searching strings. Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, pp. 360-369.

Bay, S. and Schwabacher, M. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 29-38.

Blum, H. 1973. Biological shape and visual science. Journal of Theory. Biol., 38:205-287.

Castrejon-Pita, A. A., Sarmiento-Galan, A., Castrejon-Pita, J. R., and Castrejon-Garcia, R. 2005. Fractal dimension in butterflies' wings: a novel approach to understanding wing patterns? J. Math. Biol. 50:584–594.

Chen, Z., Fu, A., and Tang, J. 2003. On complementarity of cluster and outlier detection schemes. Proc. of the 5th International Conference on Data Warehousing and Knowledge Discovery, Prague, Czech Republic, pp. 234-243.

Chiu, B., Keogh, E., and Lonardi, S. 2003. Probabilistic discovery of time series motifs. Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, D.C., pp. 493-498.

Chuang, G. and Kuo, C.-C. 1996. Wavelet descriptor of planar curves: Theory and applications. IEEE Trans. On Image Processing, 5:56-70.

Clark, J. T., Bergstrom, A., Landrum, J. E. III, Larson, F., and Slator, B. 2002. Digital archive network for anthropology (DANA): three-dimensional modeling and database development for internet access. Proc. of the VAST Euroconference, Arezzo.

Davies, E. R. 1997. Machine Vision: Theory, Algorithms, Practicalities. Academic Press, New York, pp. 171-191.

Daw, C. S., Finney, C. E. A., and Tracy, E. R. 2002. Symbolic analysis of experimental data. Review of Scientific Instruments, 74(22):915-930.

Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. 1994. Fast subsequence matching in time-series databases. Proc. of ACM SIGMOD International Conference on Management of Data, Minneapolis, MN, pp. 419-429.

Ghoting, A., Parthasarathy, S. and Otey, M. 2006. Fast mining of distance-based outliers in high dimensional datasets. Proc. of the 6th SIAM International Conference on Data Mining, Bethesda, MD, pp. 608-612.

Grass, J. and Zilberstein, S. 1996. Anytime algorithm development tools. Sigart Artificial Intelligence, 7(2):20-27.

Huang, Y. and Yu. P. S. 1999. Adaptive query processing for time-series Data. Proc. of 5th International Conference on Knowledge Discovery and Data Mining, San Diego, CA, pp. 282-286.

Indyk, P., Motwani, R., Raghavan, P., and Vempala, S. 1997. Locality-preserving hashing in multidimensional spaces. Proc. of the 29th Annual ACM symposium on Theory of Computing, El Paso, Texas, pp. 618-625.

Jalba, A. C., Wilkinson, M. H. F., Roerdink, J. B. T. M., Bayer, M. M., and Juggins, S. 2005. Automatic diatom identification using contour analysis by morphological curvature scale spaces. Machine Vision and Applications, 16(4):217-228.

Jolliffe , I. T. 2002. Principle Component Analysis. Springer, 2nd edition.

Keogh, E. 2001. Similarity Search in Massive Time Series Databases. Ph.D. Thesis, University of California, Irvine.

Keogh, E., Chakrabati, K., Pazzani, M., and Mehrotra, S. 2001. Dimensionality reduction for fast similarity search in large time series databases. Journal of Knowledge and Information Systems, 3(3):263-286.

Keogh, E. and Kasetty, S. 2002. On the need for time series data mining benchmarks: a survey and empirical demonstration. Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Alberta, Canada, pp. 102-111.

Keogh, E., Lin, J., and Fu, A. 2005. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. Proc. of the 5th IEEE International Conference on Data Mining, Houston, Texas, pp. 226-233.

Keogh, E., Lonardi, S., and Chiu, W. 2002. Finding surprising patterns in a time series database in linear time and space. Proc. of the 8th International Conference on Knowledge Discovery and Data Mining, Alberta, Canada, pp. 550-556.

Keogh, E., Lonardi, S., and Ratanamahatana, C. 2004. Towards parameter-free data mining. Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, pp. 206-215.

Keogh, E., Wei, L., Xi, X., Lee, S., and Vlachos, M. 2006. LB_Keogh allows exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. Proc. of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, pp. 882-893.

Khotanzan, A. and Hong, Y. H. 1990. Invariant image recognition by Zernike moments. IEEE Trans. PAMI, 12:489-497.

Kitaguchi, S. 2004. Extracting feature based on motif from a chronic hepatitis dataset. Proc. of the 18th Annual Conference of the Japanese Society for Artificial Intelligence, Kanazawa, Japan.

Knorr, E., Ng, R., and Tucakov, V. 2000. Distance-based outliers: algorithms and applications. VLDB Journal, 8(3-4):237-253.

Latecki, L. J. and Lakamper, R. 1999. Contour-based shape similarity. Proc. of the International Conference on Visual Information Systems, Amsterdam, pp. 617-624.

Latecki, L. J., Lakaemper, R. and Eckhardt, U. 2000. Shape descriptors for non-rigid Shapes with a single closed contour. Proc. of the IEEE Conference on Computer Vision and Patten Recognition, Hilton Head Island, SC, pp. 424-429.

Lee, D-J., Schoenberger, R., Shiozawa, D., Xu, X., and Zhan, P. 2004. Contour matching for a fish recognition and migration monitoring system. Proc. of the SPIE Optics East, Two and Three-Dimensional Vision Systems for Inspection, Control, and Metrology II, vol. 5606-05, Oct. 25-28, 2004.

Lin, J., Keogh, E., Lonardi, S., and Chiu, B. 2003. A symbolic representation of time series, with implications for streaming algorithms. Proc. of the 8th ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, CA, pp. 2-11.

Lin, J., Keogh, E., Lonardi, S., Lankford, J. P., and Nystrom, D. M. 2004. Visually mining and monitoring massive time series. Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, pp. 460-469.

Loncarin, S. 1998. A Survey of Shape Analysis Techniques. Pattern Recognit., 31(5):983-1001.

Mokhtarian, F. and Machworth, A. K. 1992. A theory of multiscale, curvature-based shape representation for planar curves. IEEE Trans. PAMI, 14:789-805.

Mollineda, R. A., Vidal, E., and Casacuberta, F. 2002. Cyclic sequence alignments: approximate versus optimal techniques. International Journal of Pattern Recognition and Artificial Intelligence, 16(3):291-299.

Morphbank. 2006. http://morphbank.csit.fsu.edu/. Florida State University, School of Computational Science.

Narayanan, M. and Karp, R.M. 2004. Gapped local similarity search with provable guarantees. Proc. of the 4th Workshop on Algorithms in Bioinformatic, Bergen, Norway, pp. 74-86.

Qamra, A., Meng, Y., and Chang, E. 2005. Enhanced perceptual distance functions and indexing for image replica recognition. IEEE transactions on Pattern Analysis and Machine Intelligence, 27(3):379-391.

O'Brien, M.J., Darwent, J., and Lyman, R.L. 2001. Cladistics is useful for reconstructing archaeological phylogenies: Paleoindian points from the southeastern United States. Journal of Archaeological Science, 28, 1115–1136.

Philip, J. W. 2006. Personal Communication. http://anthropology.ucr.edu/lithic.

Ramaswamy, S., Rastogi, R., and Shim, K. 2000. Efficient algorithms for mining outliers from large data sets. Proc. of the International Conference on Management of Data, Dallas, TX, pp. 427-438.

Rombo, S. and Terracina, G. 2004. Discovering representative models in large time series models. Proc. of the 6th International Conference On Flexible Query Answering Systems, Lyon, France, pp. 84-97.

Sadakane, K. 2000. Compressed text databases with efficient query algorithms based on the compressed suffix array. Proc. of the 11th Annual International Symposium on Algorithms and Computation, Taipei, Taiwan, pp. 410–421.

Shahabi, C., Tian, X., and Zhao, W. 2000. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries. Proc. of the 12th International Conference on Scientific and Statistical Database Management, Berlin, Germany, pp. 55-68.

Siddiqi, K., Shokoufandeh, A., Dickinson, S. J. and Zucker S. W. 1998. Shock Graphs and Shape Matching. Sixth International Conference on Computer Vision, Bombay, India, 222-229.

Söderkvist, O. J. O. 2001. Computer vision classification of leaves from swedish trees. Master thesis, Linkoping University, Sweden.

Tanaka, Y. and Uehara, K. 2004. Motif discovery algorithm from motion data. Proc. of the 18th Annual Conference of the Japanese Society for Artificial Intelligence, Kanazawa, Japan.

Tao, Y., Xiao, X., and Zhou, S. 2006. Mining distance-based outliers from large databases in any metric space. Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, pp. 394-403.

Tompa, M. and Buhler, J. 2001. Finding motifs using random projections. Proc. of the 5th International Conference on computational Molecular Biology, Montreal, Canada, pp. 67-74.

Van Otterloo, P. J. 1991. A contour-oriented approach to shape analysis. Prentice-Hall International (UK) Ltd, Englewood Cliffs, NJ, pp. 90-108.

Veltkamp, R. and Hagedoorn, M. 1999. State of the art in shape matching. Utrecht, The Netherlands, Tech. Rep. UU-CS-1999-27.

Vlachos, M., Vagena, Z., Yu, P. S., and Athitsos, V. 2005. Rotation invariant indexing of shapes and line drawings. Proc. of the 4th ACM Conference on Information and Knowledge Management, Bremen, Germany, pp. 131-138.

Wolfson, H. J., Rigoutsos, I. 1997. Geometric hashing: an overview. IEEE Computational Science and Engineering, 4(4): 10-21.

Yankov, D., Keogh, E., Lonardi, S., and Fu, A. W. 2005. Dot plots for time series analysis. Proc. of the 17th IEEE International Conference on Tools with Artificial Intelligence, Hongkong, China, pp. 159-168.

Zimmerman, E., Palsson, A., and Gibson, G. 2000. Quantitative trait loci affecting components of wing shape in Drosophila melanogaster. Genetics 155: 671–683.

Zhang, D. and Lu, G. 2004. Review of shape representation and description techniques. Pattern Recognition, 37(1): 1-19.