# SAXually Explicit Images: Finding Unusual Shapes

Li Wei Eamonn Keogh Xiaopeng Xi
*Department of Computer Science and Engineering*
*University of California, Riverside*
*{wli, eamonn, xxi}@cs.ucr.edu*

## Abstract

*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, anthropology, and medicine. While the brute force search algorithm has quadratic time complexity, we avoid this by using locality-sensitive hashing to estimate similarity between shapes which enables us to reorder the search more efficiently. An extensive experimental evaluation demonstrates that our approach can speed up computation by three to four orders of magnitude.*

## 1. Introduction

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 [29]. 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.

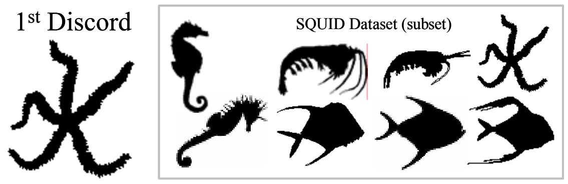


**Figure 1: *Right)* Samples from a dataset of 1,301 images of marine creatures. *Left)* The No.1 shape discord found in this dataset is a starfish**

Note that while 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. 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 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 [28], and the discord discovered by our algorithm.

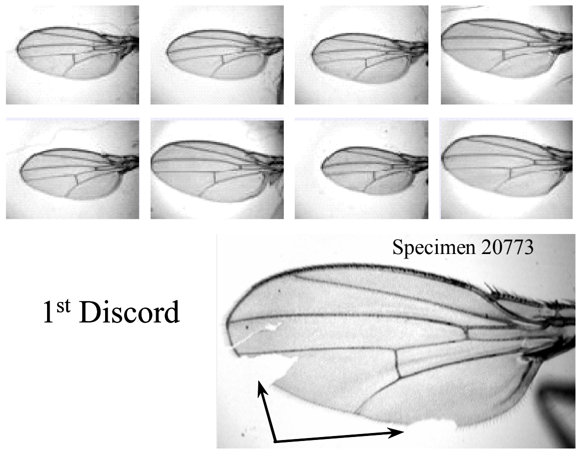


**Figure 2: *Top)* A subset of 32,028 images of Drosophila wings. *Bottom)* The No.1 shape discord found in this dataset is a damaged wing**

**Anthropological Data Mining:** Anthropology offers many interesting challenges for data mining, particularly mining of *shapes* [5]. Examples of shapes which anthropologists may be interested in mining include petroglyphs, pottery [5], projectile points ("arrowheads") [22], and bones [15]. 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. While some subjectivity exists, the two discords discovered are arguably the most unusual shapes in the dataset. 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.

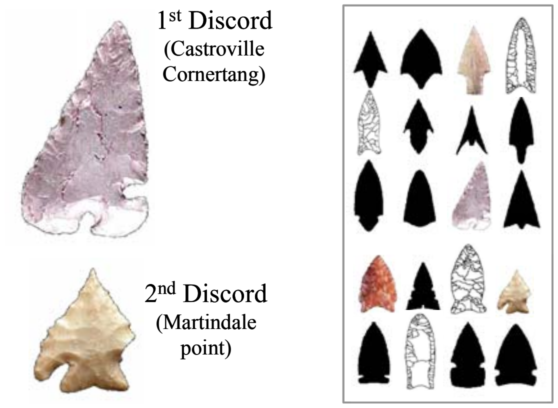


**Figure 3: *Right)* A subset of 16,000 images of projectile points collected from diverse sources. *Left)* The top two discords found**

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. 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 on Shapes

We consider the shape of an object as a binary image representing the outline of the object [19]. In order to find/index/classify a shape, the shape must be described or represented in some way. 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 [15]. Therefore this simple representation has been used by an increasingly large fraction of the literature [6][15][25].

There exist dozens of techniques to convert shapes into one-dimensional representations (also known as pseudo "time series"). We refer the interested reader to [19] and [29] 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 [29].

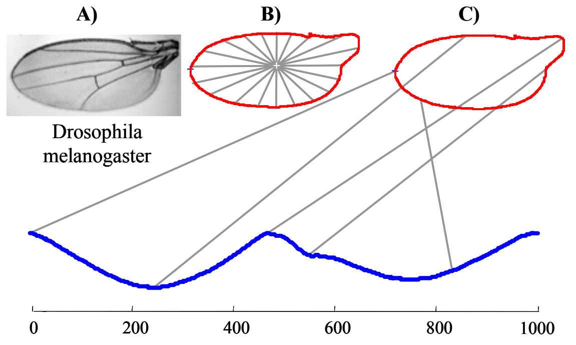


**Figure 4: Shapes can be converted to time series. A) A bitmap of a fruit flies wing. B) The distance from every point on the profile to the center is measured and treated as the Y-axis value of a time series of length *n* (C)**

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.

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 $D$, $Dist(D, M_D) > Dist(C, M_C)$.

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* [4][12].

**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, Euclidean distance can produce extremely poor results. Therefore we need the distance function to be rotation invariant. One way to achieve this is to hold one shape fixed, rotate the other, and record the minimum distance of all possible rotations. In time series space, this is accomplished 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 $\boldsymbol{C}$ of size $n$ by $n$

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

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 $\boldsymbol{C} = \{C^1, C^2, \ldots, C^n\}$.

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

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

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 $O(n^2)$.

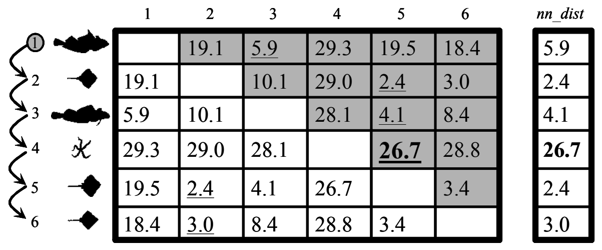
## 3. Shape Discords Discovery Framework

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 1.

**Table 1: Brute Force Discord Discovery**

```
    Function [ dist, index ] = BruteForce_Search($S$)
1   best_so_far_dist = 0
2   best_so_far_index = NaN
3   for $p$ = 1 to |$S$|                                   // begin outer loop
4     nearest_neighbor_dist = infinity
5     for $q$ = 1 to |$S$|                                 // begin inner loop
6       if $p$ != $q$                                      // do not compare to self
7         if $Dist(C_p, C_q)$ < nearest_neighbor_dist
8           nearest_neighbor_dist = $Dist(C_p, C_q)$
9         end
10      end
11    end                                                  // end inner loop
12    if nearest_neighbor_dist > best_so_far_dist
13      best_so_far_dist = nearest_neighbor_dist
14      best_so_far_index = $p$
15    end
16  end                                                    // end outer loop
17  return [ best_so_far_dist, best_so_far_index ]
```

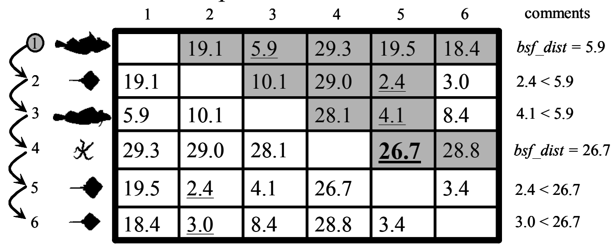
While the brute force algorithm is intuitive, we will show a running example to develop some intuition on how to improve this algorithm. Figure 5 shows a "trace" of the brute force algorithm on a simple dataset of six marine creatures. The first discord happens to be the starfish (shape 4), whose distance to its nearest match is 26.7 (shown in bold in Figure 5). 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.



| | 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 5: 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**

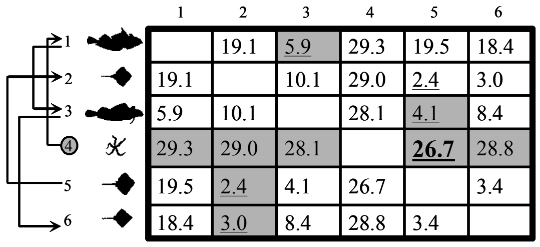
The brute force algorithm is easy to implement and produces exact results. However, it has O($m^2$) time complexity (recall that $m$ is the size of the dataset $S$). The astute reader may have already noticed a way to speed up the search. In the inner loop, we do not need to find the true nearest neighbor to the current candidate. 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. We call this simple optimization *early abandoning*. If we apply this optimization to the marine creature dataset, we only need to compute the distance between 12 pairs of shapes. Figure 6 shows a trace of the search process.



| | 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 6: 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. However, 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. 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 7), 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 7: 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 2) and the additional outer and inner heuristics. The pseudo code is shown in Table 2.

**Table 2: 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 ]
```

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, which is the topic of the next section.

## 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). With these time constraints, the problem at hand is much harder. 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 2 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 early terminate in 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. There are many different symbolic approximations of time series in the literature [1][7][9]. We choose the **S**ymbolic **A**ggregate Appro**X**imation (SAX) representation introduced by Lin, et al. [17], 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) [11].

**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 $\bar{C} = \bar{c}_1, \ldots, \bar{c}_w$ where

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

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 [17]. 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 $\beta_i$ to $\beta_{i+1} = 1/|\Sigma|$ ($\beta_0$ and $\beta_{|\Sigma|}$ are defined as $-\infty$ and $\infty$, respectively).

Once the breakpoints have been obtained, we can assign the same letter to all PAA coefficients that belong to the same interval. Figure 8 shows an example.

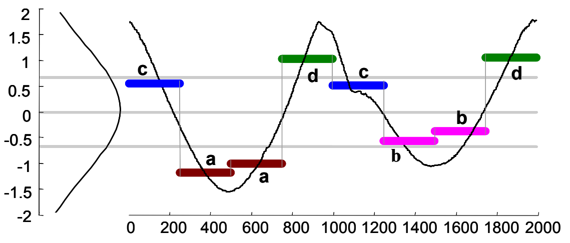


**Figure 8: A time series (thin black line) is discretized by first obtaining a PAA approximation (heavy gray line) and then using predetermined breakpoints to map the PAA coefficients into symbols (bold letters). In the example above, with *n* = 1992, *w* = 8, and |Σ| = 4, the time series is mapped 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 $\alpha_i$ denote the $i^{th}$ element of the alphabet, i.e., $\alpha_1$ = **a** and $\alpha_2$ = **b**. Then the mapping from a PAA approximation $\bar{C}$ to a word $\hat{C}$ is obtained as follows:

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

## 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 [21][25][24], etc. Among the rich literature, the locality-sensitive hashing search technique proposed by Indyk and Motwani [10] 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 11.** *Locality-sensitive Hash Function*: Consider a string $s$ of length $w$ over an alphabet $\Sigma$ 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 [10].

Unfortunately, this property does not hold for shapes because of the rotation variance. For example, the two arrowheads in Figure 9 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.

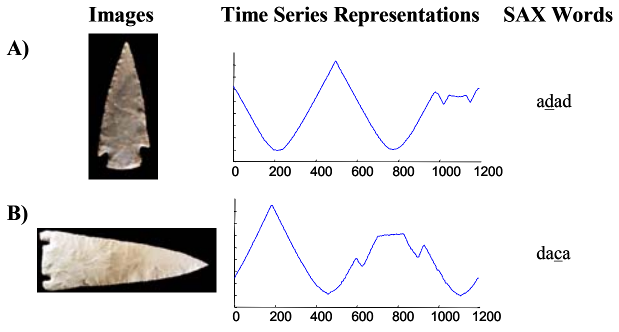


**Figure 9: Image A) and B) are very similar to each other, but have different orientations. The time series extracted from the two images look different (shifted by some offset). Note that the corresponding SAX words are not only shifted but also different by one symbol (the underscored position)**

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

**Definition 12.** *Rotation invariant Locality-sensitive Hash Function*: Consider a string $s$ of length $w$ over an alphabet $\Sigma$ 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 9. Using rotation invariant hashing, both of them are mapped to the LSH value "aa", as shown in Figure 10.

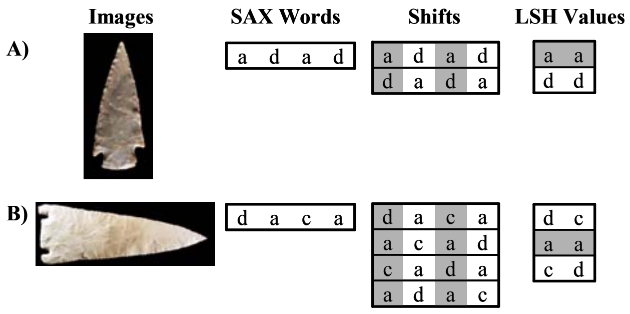


**Figure 10: The same two images as in Figure 9 are both mapped to LSH value "aa". Here $w$ = 4, $\Sigma$ = {$a$, $b$, $c$, $d$}, and $k$ = 2. The indices chosen by the hash function are {1, 3}**

## 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 $\Sigma$. 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 11 gives a visual intuition of this, where both $w$ and $|\Sigma|$ are set to 4. 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 11, 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. 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.

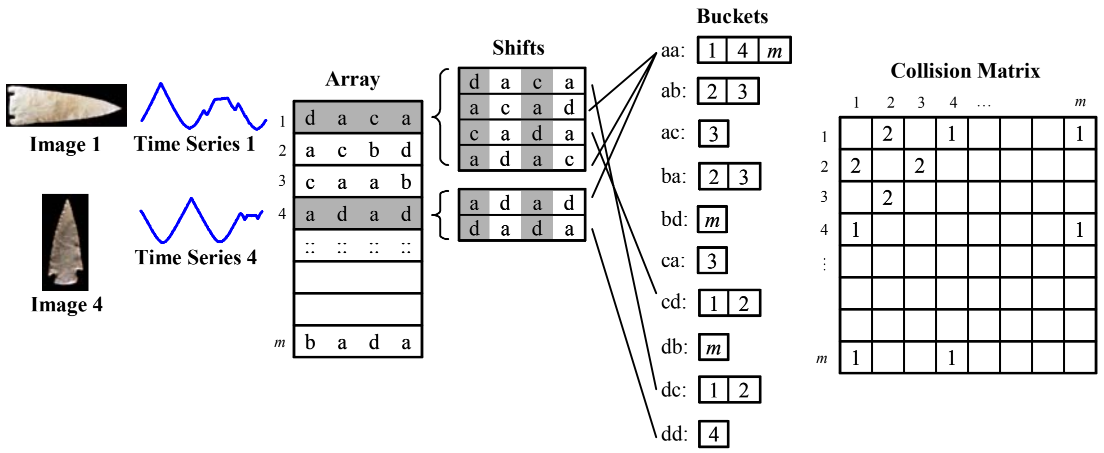


**Figure 11: Illustration of the similarity estimation process. *Left)* The time series extracted from images are converted to SAX words and then inserted into an array. *Middle)* A hash function maps SAX words into buckets. *Right)* Collisions are recorded by incrementing the corresponding cells in the collision matrix**

The above process of time series extracting, discretizing, hashing, and collision recording is formalized in Table 3. 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.

**Table 3: 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 3).

**Outer Heuristic:** We scan the collision matrix row by row to find the largest number of collisions each shape has with others. Then we sort the shapes in ascending order using this value. The resulting ordering is given to the outer loop.

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 2 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 [2][4][14][16][18][23]). 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.

## 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.

We note that the quality of illustrations here suffers from monochromic printing. We urge the interested reader to a longer version of the paper for large-scale color reproductions and additional details.

## 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 [3]. Here we restrict our attention to shape. The large size of such collections motivates the use of scalable algorithms. For example, the Morphbank archive [20] currently has approximately 2,000 butterfly images online, and many lepidopterists (butterfly collectors) have much larger personal collections.

Consider the image dataset in Figure 12, which purportedly shows a (subset of) collection of *Heliconius erato* (Red Passion Flower) Butterflies.

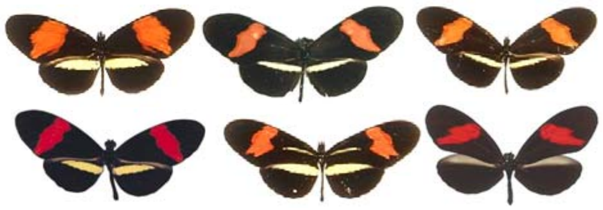


**Figure 12: Six examples of butterflies, ostensibly all *Heliconius erato* (Red Passion Flower) Butterflies**

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 12. 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 13 clearly shows why the one butterfly was singled out from the collection.

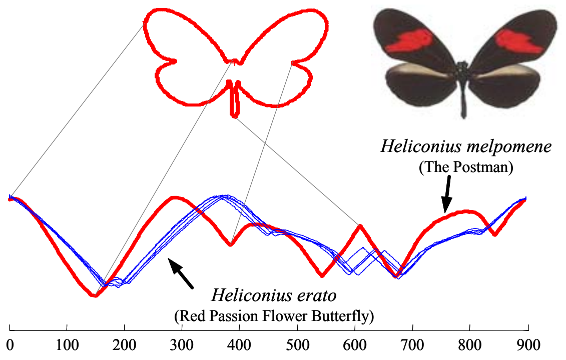


**Figure 13: The six butterflies from Figure 12 shown in their time series representations. One butterfly, shown with a bold line, is a different species to the rest**

### *5.1.2 Red Blood Cell Dataset*

We ran some tests on images extracted from red blood cell data. Figure 14 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.

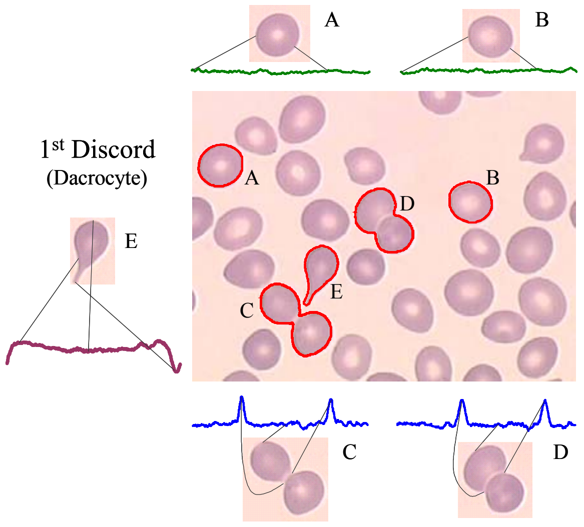


**Figure 14: The discord discovered in an image of red blood cells is a teardrop shaped cell, or *dacrocyte*, which is indicative of several blood disorders**

### *5.1.3 Fungus Dataset*

We perform similar experiments on images taken at even higher resolutions. Figure 15 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.

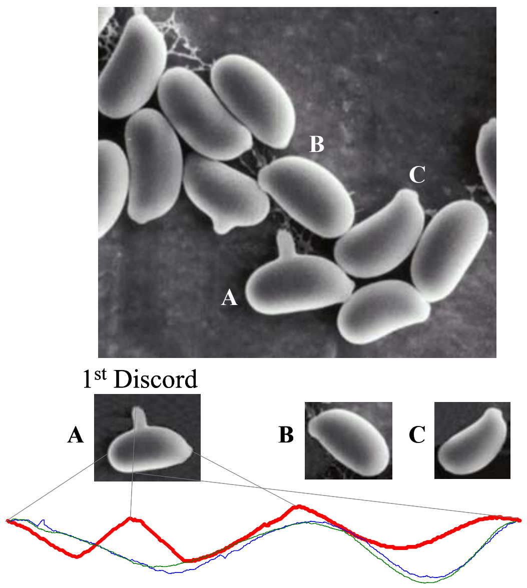


**Figure 15: Some spores produced by a fungus. One spore is different because it has a germ tube (Image by Charles Mims, University of Georgia)**

## 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. The measurement we use is the number of times that the distance function is called on line 7 in Table 2. 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 [12][13][26].

For our approximate optimal heuristic, we include a startup cost of O($m$), which is the time complexity required to build the collision matrix. 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 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 16. 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.

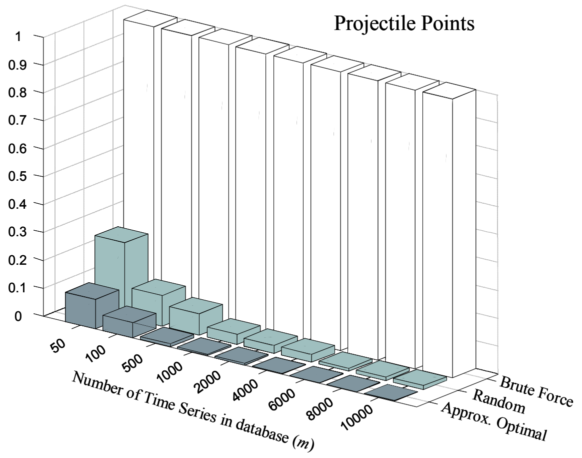


**Figure 16: The relative performance for three heuristic strategies on the Projectile Points dataset**

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.

## 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.

## 8. References

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

[2] Bentley, J. L. and Sedgewick, R. Fast algorithms for sorting and searching strings. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp 360-369, 1997.

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

[4] Chiu, B., Keogh, E., and Lonardi, S. Probabilistic discovery of time series motifs. In *Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 493-498, 2004.

[5] Clark, J. T., Bergstrom, A., Landrum, J. E. III, Larson, F., and Slator, B. Digital archive network for anthropology (DANA): three-dimensional modeling and database development for internet access. In *Proc. of the VAST Euroconference*, Arezzo, 2000. BAR International Series 1075.

[6] Davies, E. R. *Machine Vision: Theory, Algorithms, Practicalities*. Academic Press, pp 171-191, 1997.

[7] Daw, C. S., Finney, C. E. A., and Tracy, E. R. Symbolic Analysis of experimental Data. *Review of Scientific Instruments*, 2002-7-22.

[8] Grass, J. and Zilberstein, S. Anytime algorithm development tools. *Sigart Artificial Intelligence*, 7(2), 1996.

[9] Huang, Y. and Yu. P. S. Adaptive query processing for time-series Data. In *Proc. of 5th International Conference on Knowledge Discovery and Data Mining*, pp 282-286, 1999.

[10] Indyk, P., Motwani, R., Raghavan, P., and Vempala, S. Locality-preserving hashing in multidimensional spaces. In *Proc. of the 29th annual ACM symposium on Theory of computing*, pp 618-625, 1997.

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

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

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

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

[15] Keogh, E., Wei, L., Xi, X., Lee, S., and Vlachos, M. LB_Keogh allows exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proc. of the 32nd International Conference on Very Large Data Bases*, to appear, 2006.

[16] Kitaguchi, S. Extracting feature based on motif from a chronic hepatitis dataset. In *Proc. of the 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI)*, 2004.

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

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

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

[20] Morphbank. http://morphbank2.csit.fsu.edu/

[21] Narayanan, M. and Karp, R.M. Gapped Local Similarity Search with Provable Guarantees. In *Proc. of the 4th Workshop on Algorithms in Bioinformatics (WABI)*, pp 74-86, 2004.

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

[23] Tanaka, Y. and Uehara, K. Motif discovery algorithm from motion data. In *Proc. of the 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI),* 2004.

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

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

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

[27] Wei, L. http://www.cs.ucr.edu/~wli/publications/ICDM06_Discords.pdf

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

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