# Approximation Algorithm for the Cyclic Swap Problem

Yoan José Pinzón Ardila[1], Costas S. Iliopoulos[1], Gad M. Landau[2], and Manal Mohamed[1]

[1] King's College London, Dept. of Computer Science, London WC2R 2LS, UK
e-mail: Yoan.Pinzon@kcl.ac.uk , e-mail: ⟨csi,manal⟩@dcs.kcl.ac.uk

[2] Dept. of Computer Science, Haifa University, Haifa 31905, Israel
e-mail: landau@cs.haifa.ac.il

**Abstract.** Given two $n$-bit (cyclic) binary strings, $A$ and $B$, represented on a circle (necklace instances). Let each sequence have the same number $k$ of 1's. We are interested in computing the cyclic swap distance between $A$ and $B$, *i.e.*, the minimum number of swaps needed to convert $A$ into $B$, minimized over all rotations of $B$. We show that this distance may be approximated in $O(n + k^2)$ time.

## 1 Introduction

Cyclic string comparison is important for different domains where linear strings represent cyclic sequences, for example, in computational biology the genetic material is sequenced from circular DNA or RNA molecules. Bacterial, chloroplasts and mitochondrial genomes are in majority circular [2]. Small circular DNA molecules that have the ability to replicate on their own, are extensively used in biotechnology [2]. All such cyclic molecules are represented as linear strings by choosing an arbitrary starting point. It follows that the comparison of two such sequences needs to consider all possible cyclic shifts of one of the sequences. DNA, as well as RNA, are oriented molecules, therefore, in some cases, *e.g.*, for Expressed Sequence Tags, sequences must be compared in each orientation [1]. Other domains are pattern representation and recognition [6]. There, polygonal shapes are encoded into linear strings by choosing arbitrarily a start position on the contour. Determining if two shapes are similar requires to compare one string with all cyclic shifts of the other.

Another domain in which cyclic strings arise is computational music analysis. Mathematics and music theory have a long history of collaboration dating back to at least Pythagoras [11]. More recently the emphasis has been mainly on analysing string pattern matching problems that arise in music theory [7, 8, 9, 10]. A fundamental problem in music theory is to measure the similarity between rhythms, with many applications such as copyright infringement resolution and music information retrieval.

Six examples of 4/4 time clave and bell timelines are given in Fig. 1. The left-hand side shows the rhythms with standard Western music notation using the smallest convenient notes and rests. The right-hand side shows a popular way of representing

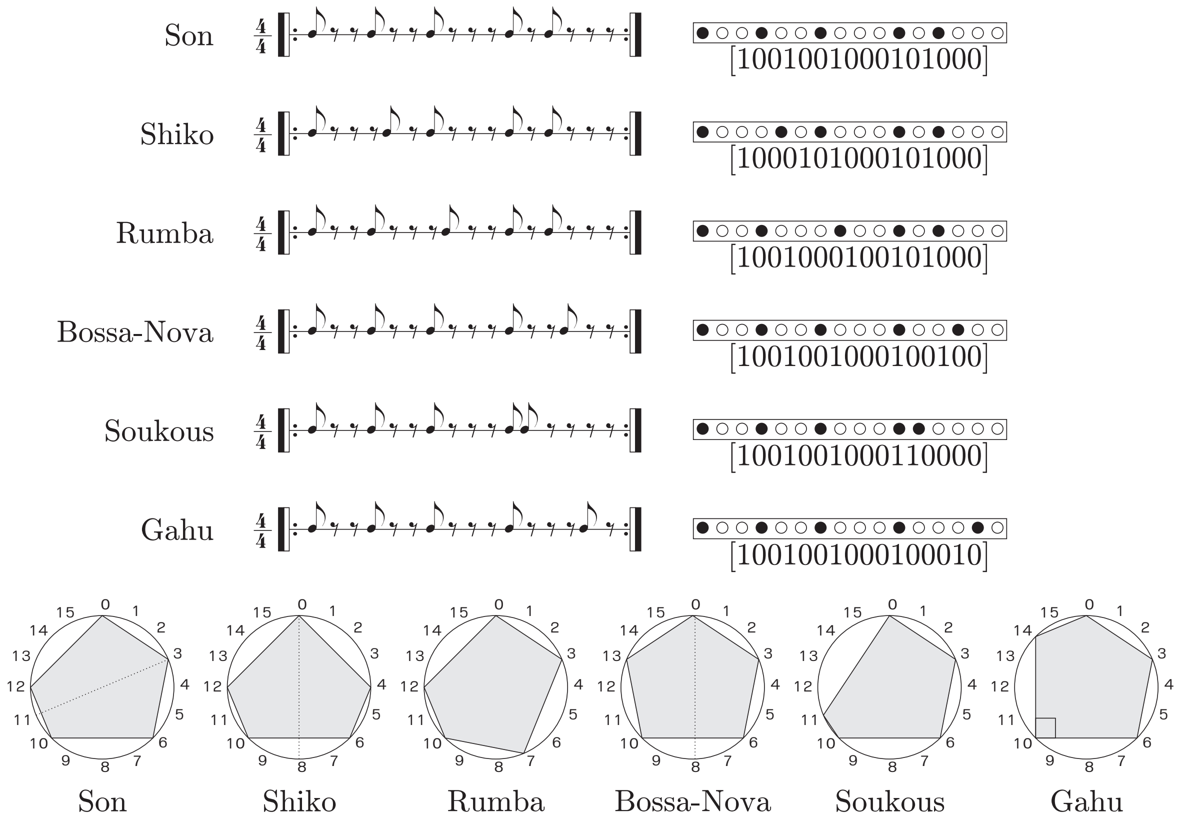


Figure 1: Six fundamental 4/4 time *clave* rhythms. The left-hand side uses the Western music notation and the right-hand size the box notation and binary representation. The bottom line shows a common geometric representation using convex polygons. The dashed lines indicate an axis of mirror symmetry (*e.g.* for the *son clave*, if the rhythm is started at location 3 then it sounds the same whether it is played forward or backwards).

rhythms for percussionists that do not read music. It is called the Box Notation Method developed by Philip Harland at the University of California in Los Angeles in 1962 and is also known as TUBS (Time Unit Box System). The box notation method is convenient for simple-to-notate rhythms like bell and clave patterns as well as for experiments in the psychology of rhythm perception, where a common variant of this method is simply to use one symbol for the note and another for the rest. The *Clave Son* is the most popular among these rhythms and can be heard a lot in Son and Salsa music as well as much other music around the world. For our purpose, A rhythm is represented as a cyclic binary sequence where a zero denotes a rest (silence) and a one represents a beat or note onset, for example, the clave Son would be written as the 16-bit binary sequence: [1001001000101000]. This rhythm can also be thought as a point in a 16-dimensional space (the hypercube). A natural measure of the difference between two rhythms represented as binary sequences is the well known Hamming distance, which counts the number of positions in which the two rhythms disagree. Although the Hamming distance measures the existence of a mismatch, it does not measure how far the mismatch occurs, that is why, Toussaint [15] proposed a distance measure termed the *swap distance*. A swap is an interchange of a one and a zero (note duration and rest interval) that are adjacent in the sequence. The swap distance between two rhythms is the minimum number of swaps required to convert one rhythm to the other. The swap distance measure of dissimilarity was shown in [14] to be more appropriate than several other measures of rhythm similarity including the Hamming distance, the Euclidean interval-vector distance, the interval-difference distance measure of Coyle and Shmulevich, and the chronotonic distance measures of Gustafson and Hofmann-Engl.

More formally, given two $n$-bit (cyclic) binary strings, $A$ and $B$, represented on a circle (necklace instances). Let each sequence have the same number $k$ of 1's. We are interested in computing the cyclic swap distance between $A$ and $B$, *i.e.*, the minimum number of swaps needed to convert $A$ into $B$, minimized over all rotations of $B$. We show that this distance may be computed in $O(n + k^2)$.

The outline of the paper is as follows: Some preliminaries are described in Section 2. A Naive Solution is presented in Section 3 and in Section 4 we present a better-than-naive solution. Conclusions are drawn in Section 5.

## 2 Preliminaries

Let $X[0..n-1]$ be a $n$-bit *cyclic string* over $\Sigma = \{0, 1\}$, with $n \geq 0$. By $X[i]$ we denote the $(i + 1)$-st bit in $X$, $0 \leq i <$length$(X)$. Let $\ell$=ones$(X)$ be the number of 1's in $X$. Let $X^r$ be the $r$-rotation of $X$ such that $X^r[i] = X[i \oplus r]$ for $i \in [0..n-1]$, and any integer $r$ ($i \oplus r = \text{mod}(i + r, n)$[1]).

Let $x$ be the increasing sequence of 1's indices $(x_0, x_1, \ldots, x_{\ell-1})$ such that $X[x_i] = 1$ for $i \in [0..\ell-1]$. For $u = (u_0, u_1, \ldots, u_n)$, $v = (v_0, v_1, \ldots, v_n)$ and some integer $e$, we denote by $u \cdot v = (u_0, u_1, \ldots, u_n, v_0, v_1, \ldots, v_n)$ the *sequence concatenation operation* and by $u + e = (u_0 + e, u_1 + e, \ldots, u_n + e)$ the *sequence transposition operation.*

Given $X$ and $Y$, two $n$-bit (cyclic) binary strings with the same number of 1's, the *cyclic swap problem* is to find the cyclic swap distance between $X$ and $Y$, *i.e.*, the minimum number of swaps needed to convert $X$ into $Y$, minimized over all rotations of $Y$. A swap is an interchange of a one and a zero that are adjacent in the binary string.

### 2.1 Mappings and Rotations

A mapping is a bijection function $\mathcal{M} : x \to y$. Since no two adjacent 1's can be swaped, no two mappings should cross. Hence, there are $\ell$ possible mappings. For instance, if $\ell = 3$, we could have the following three mappings $\{(x_0 \to y_0), (x_1 \to y_1), (x_2 \to y_2)\}$, $\{(x_0 \to y_1), (x_1 \to y_2), (x_2 \to y_0)\}$ and $\{(x_0 \to y_2), (x_1 \to y_0), (x_2 \to y_1)\}$. So we define

$$\mathcal{M}^k = \{ \ (x_i \to y_{i \oplus k}) \ \mid \ i, k \in [0..\ell-1] \ \} \tag{1}$$

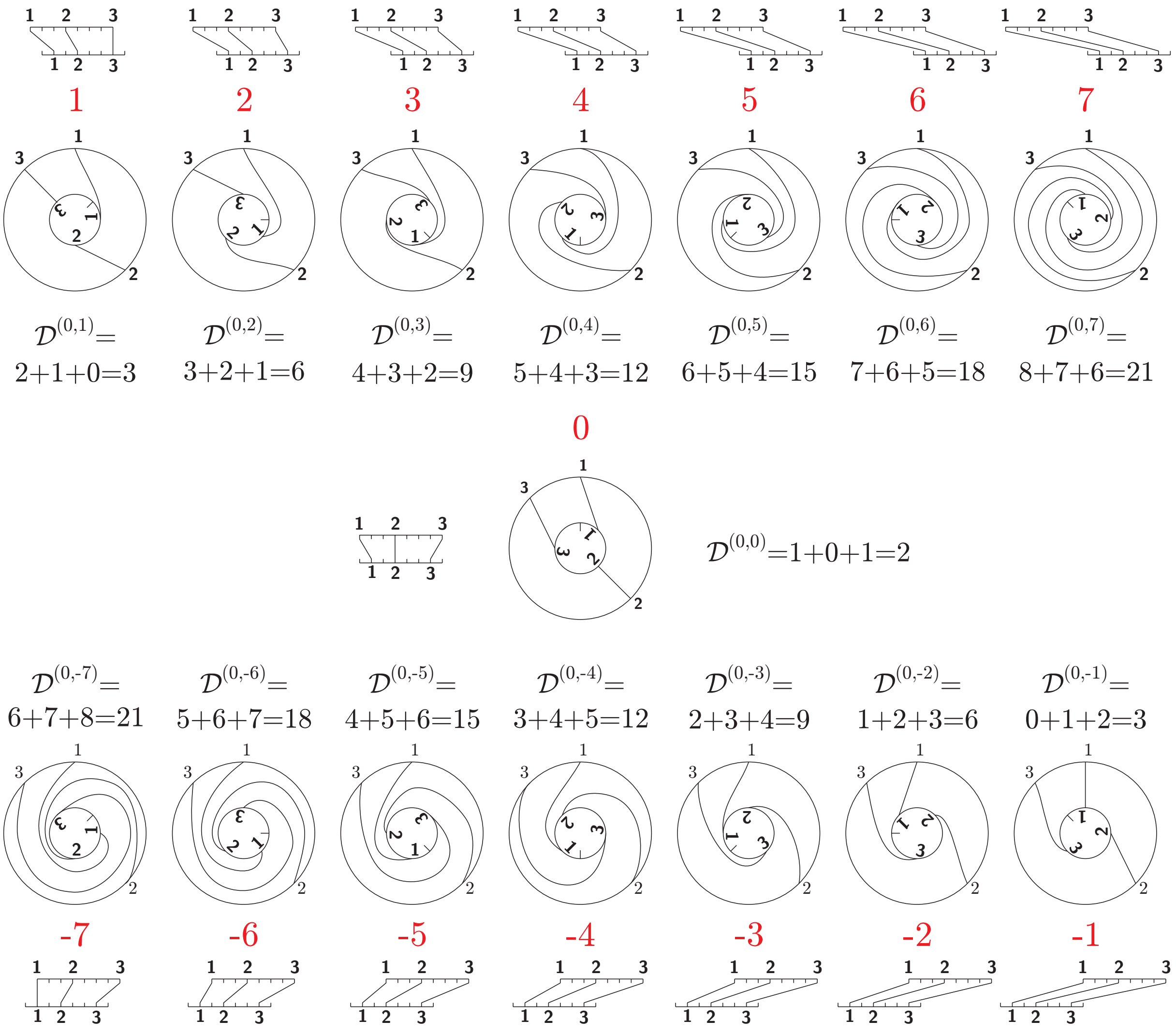
to be the *k-mapping* of the 1's in $X$ with those 1's in $Y$.

We also redefine $y$ as $y \cdot (y + n)$ and denote

[1]We use operator $\oplus$ to indicate that all indices are viewed modulo $n$.

$$\mathcal{D}^{(k,r)} = \sum_{i=0}^{\ell-1} |y_{i+k} - x_i + r|, \text{ for } k \in [0..\ell-1] \tag{2}$$

to be the sum of the number of swap between the pairs in $\mathcal{M}^k$ and rotation $r$ of $Y$. (Here $|x|$ designates the absolute value of $x$.) The reason for using $y \cdot (y+n)$ instead of $y$ is because $M^k$ will always have $k$ mappings that are circular, for example, for $\mathcal{M}^1$ we map $x_2$ with $y_0$ but the number of swap needed to map $x_2$ with $y_0$ is $(y_0+n) - x_2$ instead of $y_0 - x_2$. Fig. 2 shows rotations $-7, -6, \ldots, 6, 7$ for mapping $\mathcal{M}^0$ of $X = [10010001]$ and $Y = [01010010]$. Note that $\mathcal{D}^{(k,n-r)} + \mathcal{D}^{(k,-r)} = \ell n$ for $0 < r < n$.



1
$\mathcal{D}^{(0,1)}$=2+1+0=3
2
$\mathcal{D}^{(0,2)}$=3+2+1=6
3
$\mathcal{D}^{(0,3)}$=4+3+2=9
4
$\mathcal{D}^{(0,4)}$=5+4+3=12
5
$\mathcal{D}^{(0,5)}$=6+5+4=15
6
$\mathcal{D}^{(0,6)}$=7+6+5=18
7
$\mathcal{D}^{(0,7)}$=8+7+6=21
0
$\mathcal{D}^{(0,0)}$=1+0+1=2
$\mathcal{D}^{(0,-7)}$=6+7+8=21
-7
$\mathcal{D}^{(0,-6)}$=5+6+7=18
-6
$\mathcal{D}^{(0,-5)}$=4+5+6=15
-5
$\mathcal{D}^{(0,-4)}$=3+4+5=12
-4
$\mathcal{D}^{(0,-3)}$=2+3+4=9
-3
$\mathcal{D}^{(0,-2)}$=1+2+3=6
-2
$\mathcal{D}^{(0,-1)}$=0+1+2=3
-1

Figure 2: Rotations $-7, -6, \ldots, 6, 7$ for mapping $\mathcal{M}^0$ of $X = [10010001]$ and $Y = [01010010]$ or $x = (0, 3, 7)$ and $y = (1, 3, 6)$. $X/Y$ corresponds to the outer/inner cyclic string.

## 3 Naive Solution

The naive approach is to examine each mapping and calculate for each possible rotation the sum of number of swap operations between each pair of mapped 1's. This approach costs $O(n\ell^2)$ time. This is because there are $\ell$ possible mappings and $n$ possible rotations per each mapping. Fig. 3 shows the main steps of the algorithm.

```
Algorithm 1 Naive Solution
Input: $x, y, n, \ell$
Output: $d_{min}, r_{min}, k_{min}$
1.  $y = y \cdot (y + n)$; $d_{min} = r_{min} = k_{min} = \infty$
2.  for $k = 0$ to $\ell - 1$ do
3.          for $r = 0$ to $n - 1$ do
4.                  $d = 0$
5.                  for $i = 0$ to $\ell - 1$ do
6.                      $d = d + |y_{i+k} - x_i + r|$
7.                  if $d < d_{min}$ then $d_{min} = d, r_{min} = r, k_{min} = k$
8.  return $(d_{min}, r_{min}, k_{min})$
```

Figure 3: Naive Algorithm.

## 4 Better-than-Naive Solution

The problem is reduced to find, for each mapping $\mathcal{M}^k$, the rotation $r$ that minimizes Equation 2. This can be approximated by replacing the absolute values in (2) by squares so we get

$$\mathcal{D}'^{(k,r)} = \sum_{i=0}^{\ell-1} (y_{i+k} - x_i + r)^2, \text{ for } k \in [0..\ell - 1].$$

Lets say that $s_i^k = y_{i+k} - x_i$ so we get

$$\mathcal{D}'^{(k,r)} = \sum_{i=0}^{\ell-1} (s_i^k + r)^2,$$

which we can be rewritten as

$$\mathcal{D}'^{(k,r)} = \sum_{i=0}^{\ell-1} (s_i^k)^2 + 2r \sum_{i=0}^{\ell-1} s_i^k + \ell r^2. \tag{3}$$

Differentiating (3) and setting the result equal to zero, we obtain

$$\frac{\partial \mathcal{D}'^{(k,r)}}{\partial r} = 2 \sum_{i=0}^{\ell-1} s_i^k + 2\ell r = 0. \tag{4}$$

If we say that $\mathcal{S}^k = \sum_{i=0}^{\ell-1} s_i^k$ and solving $r$ from (4) it follows $r = -\mathcal{S}^k/\ell$. The corollary good news is that

$$\mathcal{S}^k = \mathcal{S}^{k-1} + n, \text{ for } k \in [2..\ell-1]. \tag{5}$$

So, if we let

$$f = \mathcal{S}^0 = \sum_{i=0}^{\ell-1} s_i^0 = s_0^0 + s_1^0 + \cdots + s_{\ell-1}^0 = y_0 + y_1 + \cdots + y_{\ell-1} - x_0 - x_1 - \cdots - x_{\ell-1},$$

then

$$r'_k = -\frac{f+kn}{\ell}, \text{ for } k \in [0..\ell-1], \tag{6}$$

will correspond to the rotation that will minimize the square of the swap distance for mapping $\mathcal{M}^k$. Thus, $\mathcal{D}'^{(k,r'_k)}$ is optimal for $\mathcal{M}^k$.

Now that we know how to find the best rotation $r'_k$ for each mapping, it is easy to compute the minimum swap distance for each such rotation and the best overall rotation will be given by

$$d^* = \min_{i=0}^{\ell-1}(\mathcal{D}'^{(i,r'_i)}).$$

The time complexity to compute $f$ is $\Theta(\ell)$ and each $r'_i$ for $i \in [0..\ell-1]$ can be compute in $\Theta(1)$ time using Equation 6. So the time required to compute all rotations $r'_0, r'_1, \ldots, r'_{\ell-1}$ is $\Theta(\ell)$. Once we have the rotations, the time to compute the swap distance for each mapping and each rotation $\mathcal{D}^{(k,r'_k)}$, for $k \in [0..\ell-1]$, is $\Theta(\ell)$ if done naively. Hence, the total time to find $d^*$ is $\Theta(\ell^2)$ or $\Theta(n+\ell^2)$ if we assume that sequences $x$ and $y$ are to be computed. Fig. 4 shows these ideas algorithmically.

**Algorithm 2** Better-than-Naive Solution

```
Input: $x, y, n, \ell$
Output: $d_{min}, r_{min}, k_{min}$
1.  $y = y \cdot (y+n)$; $f = 0$; $d_{min} = r_{min} = k_{min} = \infty$
2.  for $i = 0$ to $\ell - 1$ do
3.      $f = f + y_i - x_i$
4.  for $k = 0$ to $\ell - 1$ do
5.      $r = -(f+nk)/\ell$; $d = 0$
6.      for $i = 0$ to $\ell - 1$ do
7.          $d = d + |y_{i+k} - x_i + r|$
8.      if $d < d_{min}$ then $d_{min} = d, r_{min} = r, k_{min} = k$
9.  return $(d_{min}, r_{min}, k_{min})$
```

Figure 4: Better-than-Naive Algorithm.

Note that the value $d^*$ is an approximated cyclic swap distance. Our experiments show that in very few cases $d^*$ was not the optimal cyclic swap distance. However, it indicates correctly the optimal mapping $\mathcal{M}^*$. Additionally, we were also able to prove [4] that if $r_k$ was the optimal exact rotation for mapping $\mathcal{M}^k$, then at least one of the values $|y_{i+k} - x_i + r_k|$ is equal to 0, for $0 \le i \le \ell - 1$. Thus, the exact cyclic swap distance may be calculated using first Algorithm 4 to calculate the optimal mapping. Then, if none of the values $|y_{i+k} - x_i + r|$ (Line 8) is equal to 0, then additional $O(k^2)$ is needed to calculate the optimal cyclic swap distance. Clearly, this is not going to effect the overall running time.

Before continuing our discussion, we show why Equation 5 is correct. First, we illustrate the formula using the example in Fig. 5. Here, $f = \mathcal{S}^0 = s_0^0 + s_1^0 + s_2^0 = 1-2-3 = -4$, $\mathcal{S}^1 = s_0^1 + s_1^1 + s_2^1 = 3+0+3 = 6$ and $\mathcal{S}^2 = s_0^2 + s_1^2 + s_2^2 = 5+6+5 = 16$. However, we do not need to compute $s_0^1, s_1^1, s_2^1, s_0^2, s_1^2$ and $s_2^2$ in order to compute $\mathcal{S}^1$ and $\mathcal{S}^2$, instead, we apply Equation 5 and find that $\mathcal{S}^1 = \mathcal{S}^0 + n = -4 + 10 = 6$ and $\mathcal{S}^2 = \mathcal{S}^0 + 2n = -4 + 20 = 16$.

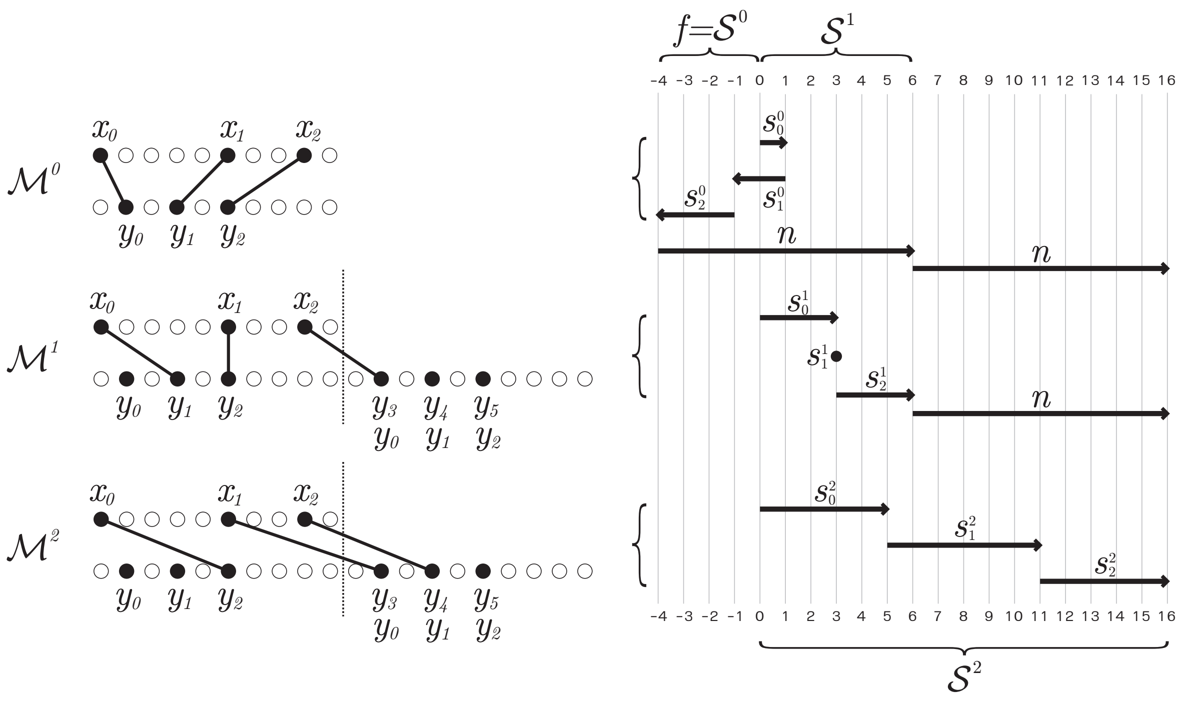


Figure 5: Illustration to demonstrate the correctness of Equation 5 for $X = [1000000100000101]$ and $Y = [0001000000101010]$ or $x = (0, 5, 8)$ and $y = (1, 3, 5)$. Note that $\mathcal{S}^1 = \mathcal{S}^0 + n$ and $\mathcal{S}^2 = \mathcal{S}^1 + n = \mathcal{S}^0 + 2n$.

More formally we know that for $\mathcal{M}^k$

$$\mathcal{S}^k = \sum_{i=0}^{\ell-1} s_i^k = \sum_{i=0}^{\ell-1} (y_{i+k} - x_i) = \sum_{i=0}^{\ell-1} y_{i+k} - \sum_{i=0}^{\ell-1} x_i.$$

Note that

$$\begin{aligned}\sum_{i=0}^{\ell-1} y_{i+k} &= y_{0+k} + y_{1+k} + \dots + y_{\ell-2+k} + y_{\ell-1+k} \\ &= y_{1+(k-1)} + y_{2+(k-1)} + \dots + y_{\ell-1+(k-1)} + y_{k-1} + n \\ &= y_{0+(k-1)} + y_{1+(k-1)} + y_{2+(k-1)} + \dots + y_{\ell-1+(k-1)} + n \\ &= \sum_{i=0}^{\ell-1} y_{i+(k-1)} + n.\end{aligned}$$

Therefore, Equation 5 is correct.

**Example.** Lets assume we want to solve the cyclic swap problem for the cyclic binary strings $X = [1000000100000101]$ and $Y = [0001000000101010]$. Then $n = 16$, $\ell = 4$, $x = (0, 7, 13, 15)$ and $y = (3, 10, 12, 14)$. Recall we double $y$ by adding $(y + n)$ to be able to compute the initial mappings, so $y$ will be $(3, 10, 12, 14, 19, 26, 28, 30)$. (See Fig. 6 for a visualization of the input data.)

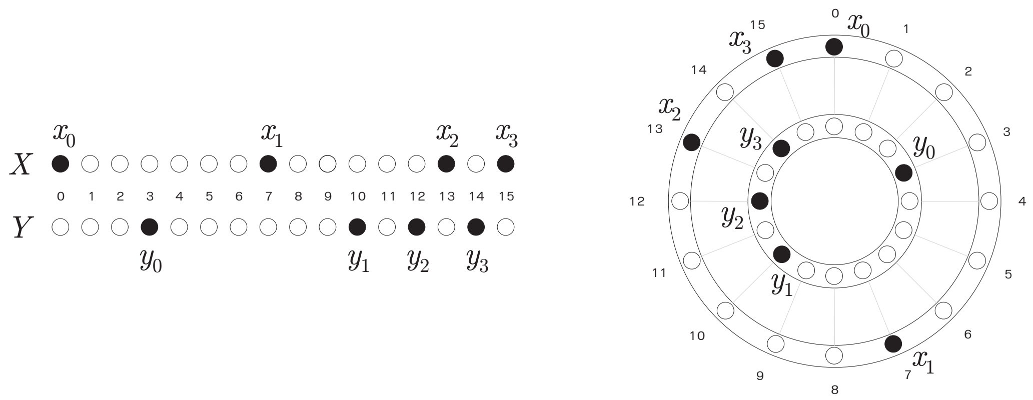


Figure 6: Example for $x = (0, 7, 13, 15)$ and $y = (3, 10, 12, 14)$. The right-hand side depicts the strings circularly.

The first step is to compute $f$ as follows

$$\begin{aligned} f &= \sum_{i=0}^{3} s_i^0 = s_0^0 + s_1^0 + s_2^0 + s_3^0 \\ &= y_0 + y_1 + y_2 + y_3 - x_0 - x_1 - x_2 - x_3 \\ &= 3 + 10 + 12 + 14 - 0 - 7 - 13 - 15 \\ &= 4.\end{aligned}$$

Now we compute the best rotation for each mapping applying Equation 6 and get

$$r'_0 = -\frac{4}{4} = -1, \quad r'_1 = -\frac{4+16}{4} = -5, \quad r'_2 = -\frac{4+32}{4} = -9, \quad r'_3 = -\frac{4+48}{4} = -13.$$

The next step is to find the swap distance for each mapping using Equation 2 and the $r$'s we found in the previous step

$$\begin{aligned}
\mathcal{D}^{(0,r'_0)} &= |y_0 - x_0 + r_0| + |y_1 - x_1 + r_0| + |y_2 - x_2 + r_0| + |y_3 - x_3 + r_0| \\
&= |3 - 0 - 1| + |10 - 7 - 1| + |12 - 13 - 1| + |14 - 15 - 1| \\
&= 2 + 2 + 2 + 2 = 8.
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}^{(1,r'_1)} &= |y_1 - x_0 + r_1| + |y_2 - x_1 + r_1| + |y_3 - x_2 + r_1| + |y_4 - x_3 + r_1| \\
&= |10 - 0 - 5| + |12 - 7 - 5| + |14 - 13 - 5| + |19 - 15 - 5| \\
&= 5 + 0 + 4 + 1 = 10.
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}^{(2,r'_2)} &= |y_2 - x_0 + r_2| + |y_3 - x_1 + r_2| + |y_4 - x_2 + r_2| + |y_5 - x_3 + r_2| \\
&= |12 - 0 - 9| + |14 - 7 - 9| + |19 - 13 - 9| + |26 - 15 - 9| \\
&= 3 + 2 + 3 + 2 = 10.
\end{aligned}$$

$$\begin{aligned}
\mathcal{D}^{(3,r'_3)} &= |y_3 - x_0 + r_3| + |y_4 - x_1 + r_3| + |y_5 - x_2 + r_3| + |y_6 - x_3 + r_3| \\
&= |14 - 0 - 13| + |19 - 7 - 13| + |26 - 13 - 13| + |28 - 15 - 13| \\
&= 1 + 1 + 0 + 0 = 2.
\end{aligned}$$

Fig. 7 shows the best rotations for each mapping. We conclude that mapping $\mathcal{M}^3$ with $Y$ rotated by -13 (+3) gives the best swap distance $\mathcal{D}^{(3,-13)} = \mathcal{D}^{(3,+3)} = 2$.

## 5 Conclusions

We have presented a new algorithm that solve the problem of cyclic swap distance between two $n$-bit (cyclic) binary strings in $O(n + \ell^2)$ where $\ell$ is the number of 1's (same) in both strings.

The fact that a $n$-bit binary string could be thought as a point in a $n$-dimensional space (the hypercube) suggests the strong links between the cyclic swap problem and the problem of calculating the closest pairs problem in high dimension. The later problem is a fundamental and well-studied problem in computational geometry. For dimension $d = n$ (which is the case here), Shamos and Bently [5] conjectured that the problem can be solved in $O(n^2 \log n)$ time, where $n$ is the number of points all in $\mathcal{R}^d$. Recently, a better-than-naive-solution has been presented with running time of $O(n^{(\omega+3)/2})$, where $O(n^\omega)$ is the running time of matrix multiplication [13]. Several approximate algorithms were designed for the high-dimensional closest pair problem; see [12] for a survey of such algorithms. We plan to use some of these ideas, plus the fact that the swap distance problem is equivalent to calculate the $L_1$ distance [3], to further improve the time complexity of the presented algorithm, where for two strings $X$, $Y$, $L_1(X, Y)$ is defined as follows:

$$L_1(X, Y) = \sum_{i=0}^{n-1} |Y[i] - X[i]|.$$

The question of whether the swap distance can be calculated in more efficient time is left as an open problem.

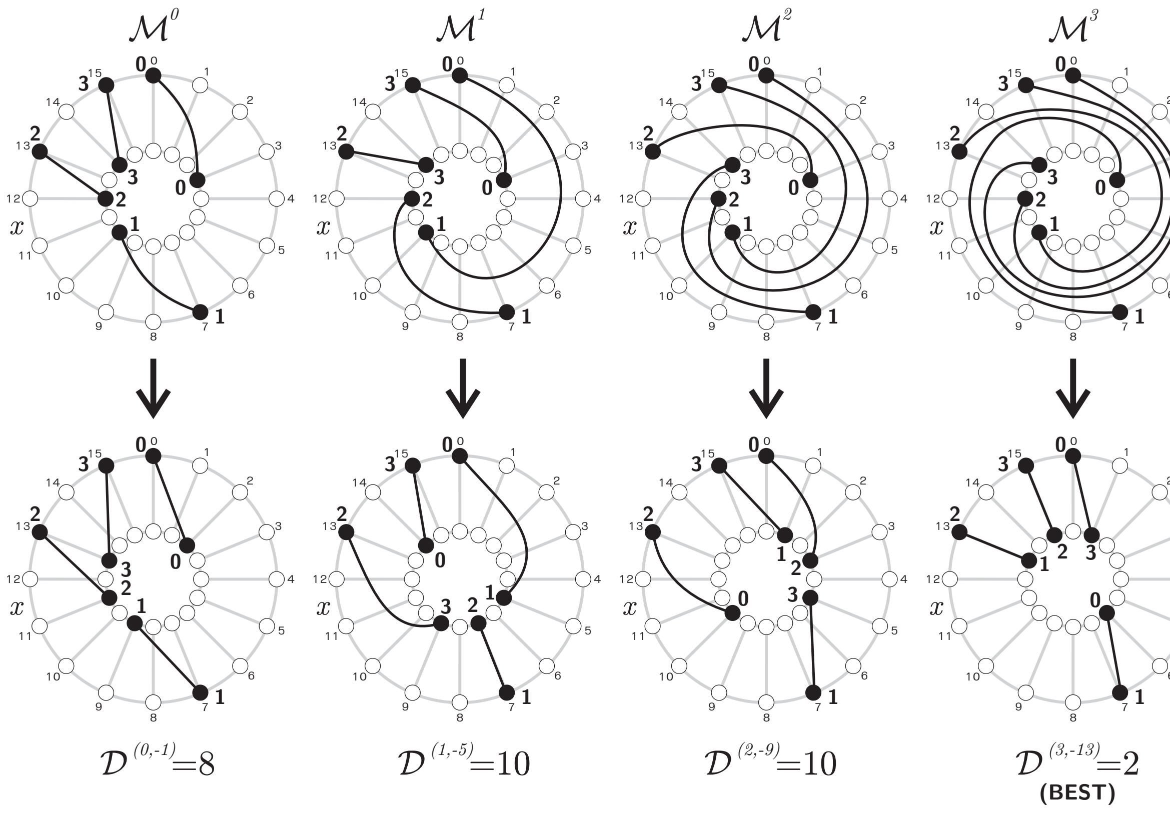


Figure 7: Computation of the minimum cyclic swap distance for $X = [1000000100000101]$ and $Y = [0001000000101010]$ or $x = (0, 7, 13, 15)$ and $y = (3, 10, 12, 14)$. The inner circle represents string $X$ while the outer circle represents string $Y$. The first row, shows the initial mappings corresponding to Equation 1. The second row shows the result after shifting the original mappings by the rotations computed using Equation 6, thus mapping $\mathcal{M}^0$ is shifted by $r'_0 = -1$ (a negative value means that the shift is performed anti-clockwise), $\mathcal{M}^1$ by -5, $\mathcal{M}^2$ by -9, and $\mathcal{M}^3$ by -13. For each new rotation the corresponding swap distance was calculated using Equation 2 and it can be seen that the best result is given by $\mathcal{M}^3$ with an anti-clockwise rotation of 13.

## References

[1] M. D. Adams, J. M. Kelley, J. D. Gocayne, M. Dubnick, M. H. Polymeropoulos, H. Xiao, C. R. Merril, A. Wu, B. Olde, R. F. Moreno, et al. Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, 252(5013):1651–1656, 1991.

[2] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland, New York, 1983.

[3] A. Amir, O. Lipsky, E. Porat and J. Umanski. Approximate Matching in the $L_1$ Metric. *In Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM'05)*, pp. 91–103, 2005.

[4] Y. J. Pinzón Ardila, R. Clifford and M. Mohamed. Necklace Swap Problem for Rhythmic Similarity Measures. Submitted for publication.

[5] J. L. Bentley and M. I. Shamos. Divide-and-Conquer in Multidimensional Space. *STOC*, pp. 220–230, 1976.

[6] H. Bunke and U. Buehler. Applications of approximate string matching to 2D shape recognition. *Pattern Recognition*, 26(12):1797—1812, 1993.

[7] E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, L. Mouchard, and Y. J. Pinzon. Computing approximate repetitions in musical sequences. *International Journal of Computer Mathematics*, 79(11):1135–1148, 2002.

[8] R. Clifford, T. Crawford, C. Iliopoulos, and D. Meredith. Problems in computational musicology. In C. S. Iliopoulos and Thierry Lecroq, editors, *String Algorithmics*, NATO Book series, King's College Publications, 2004.

[9] T. Crawford, C. S. Iliopoulos, R. Raman, String Matching Techniques for Musical Similarity and Melodic Recognition. *Computing in Musicology*, 11:71–100, 1998.

[10] M. Crochemore, C. S. Iliopoulos, G. Navarro, and Y. Pinzon. *A bit-parallel suffix automaton approach for ($\delta$,$\gamma$)-matching in music retrieval.* In M. A. Nascimento, Edleno S. de Moura, and A. L. Oliveira, editors, 10th International Symposium on String Processing and Information Retrieval, SPIRE 2003, Springer-Verlag, pp. 211–223, 2003.

[11] J. Godwin, *The Harmony of the Spheres: A Sourcebook of the Pythagorean Tradition in Music*, Inner Traditions Intl. Ltd, 1993.

[12] P. Indyk. Nearest neighbors in high-dimensional spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39. CRC Press, 2rd edition, 2004.

[13] P. Indyk, M. Lewenstein, O. Lipsky, E. Porat. Closest Pair Problems in Very High Dimensions. *ICALP* pp. 782–792, 2004.

[14] G. T. Toussaint, *Computational geometric aspects of musical rhythm*, Abstracts of the 14th Annual Fall Workshop on Computational Geometry, Massachussetts Institute of Technology, November 19-20, pp. 47–48, 2004.

[15] G. T. Toussaint, *A comparison of rhythmic similarity measures*, Proceedings of ISMIR 2004: 5th International Conference on Music Information Retrieval, Universitat Pompeu Fabra, Barcelona, Spain, October 10-14, 2004, pp. 242–245. A longer version also appeared in: School of Computer Science, McGill University, Technical Report SOCS-TR-2004.6, 2004.