International Journal of Computational Methods
© World Scientific Publishing Company

NECKLACE SWAP PROBLEM FOR RHYTHMIC SIMILARITY MEASURES

YOAN JOSÉ PINZÓN ARDILA

Department of System Engineering and Industrial Engineering,
National University of Colombia, Bogota, Colombia
ypinzon@unal.edu.co

RAPHAËL CLIFFORD

Department of Computer Science, University of Bristol, Bristol, UK
clifford@cs.bris.ac.uk

COSTAS S. ILIOPOULOS

Department of Computer Science, King's College London, London, UK
csi@dcs.kcl.ac.uk

GAD M. LANDAU *

Department of Computer Science, Haifa University, Haifa 31905, Israel
landau@cs.haifa.ac.il
and
Department of Computer and Information Science, Polytechnic University,
Six MetroTech Center, Brooklyn, NY 11201-3840

MANAL MOHAMED

Department of Computer Science, King's College London, London, UK
manal@dcs.kcl.ac.uk

Received (Day Month Year)
Revised (Day Month Year)

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 to B, minimized over all rotations of B. We show that, given the compressed representation of A and B, this distance may be computed in $O(k^2)$.

Keywords: Repeated patterns; music retrieval; swap distance; cyclic strings; rhythmic/melodic similarity.

*Partially supported by the Israel Science Foundation grant 35/05.

1. Introduction

Mathematics and music theory have a long history of collaboration dating back to at least Pythagoras[a] [Godwin (1993)]. More recently the emphasis has been mainly on analysing string pattern matching problems that arise in music theory [Cambouropoulos *et al.* (2002); Clifford *et al.* (2004); Clifford and Iliopoulos (2004); Clifford *et al.* (2005); Crawford *et al.* (1998); Crochemore *et al.* (2003)].

A fundamental problem of both theoretical and practical importance in music information retrieval is that of comparing arbitrary pieces of music. Here we restrict our attention to rhythm similarity, *i.e.* to what extent is rhythm A similar to rhythm B? Long term goals of this research include content-based retrieval methods for large musical databases using such techniques as *query-by-humming* (QBH) [Ghias *et al.* (1995); Mo *et al.* (1999)] and finding music copyright infringements [Cronin (1998)].

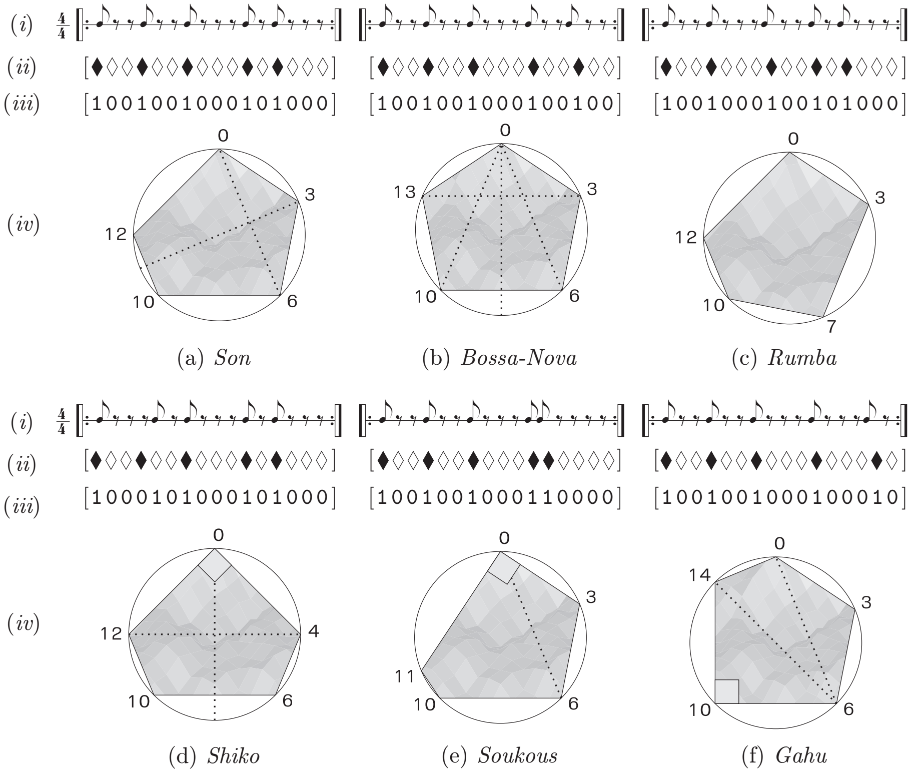
In geometry and other branches of mathematics, we often measure the similarity of two objects that are in the same class but not identical. For example, the relative similarity of two real numbers can be computed as the difference, or the square of their differences. The similarity of two functions over some period might be computed as the unsigned integral between them over this period. Thus, we can say that two pieces of music are similar if their melody or rhythm are similar.

In [Toussaint (2002)], several rhythm representations were discussed as follows: Rhythms are usually notated for musicians using the standard western music notation (*see* Figure 1(a–f)(*i*)). A more popular way of representation is called the Box Notation Method [b] intended for percussionists that do not read music (*see* Figure 1(a–f)(*ii*)). The box notation method is convenient for simple-to-notate rhythms like bell and clave patterns, where a common variant of this method is simply to use one symbol for the note (*e.g.* ⧫) and another for the rest (*e.g.* ◊). A rhythm may also be 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[c]: [1001001000101000]. An even better representation for such cyclic rhythms is obtained by imagining a clock with 16 hours marked on its face instead of the usual 12. Let us think that the hour and the minute hands have been broken off so that only the second-hand remains. Now set the clock ticking starting at noon (16 O'clock) and let it strike a bell at the 3, 6, 10 and 12 positions for a total of five strikes per clock cycle. These times are marked with a bell in Figure 2. Thereof, a common geometric representation of rhythms is obtained by connecting consecutive note locations with edges to form a convex polygon inscribed in our imaginary clock (*see* Figure 1(a–f)(*iv*)). A cyclic rhythm may be represented succinctly as a sequence of note positions; for example, $(0, 3, 6, 10, 12)$ is the *compressed representation* of [1001001000101000].

[a] In the 5th century BC, Pythagoreas was quoted to have said, "*There is geometry in the humming of strings. There is music in the spacing of the spheres*".

[b] Also known as TUBS (Time Unit Box System).

[c] This rhythm can also be thought as a point in a 16-dimensional space (the hypercube).



(a) *Son* (b) *Bossa-Nova* (c) *Rumba*

(d) *Shiko* (e) *Soukous* (f) *Gahu*

Fig. 1: Six fundamental 4/4 time *clave* rhythms. Each rhythm is depicted using (*i*) a standard western notation system, (*ii*) a box notation, (*iii*) a binary representation and (*iv*) a common geometric representation using convex polygons. The dotted lines indicate the base of an isoceles triangle or an axis of mirror symmetry. This figure is from [Toussaint (2002)].

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 proposed a distance measure termed the swap distance [Toussaint (2002); Toussaint (2004A); Toussaint (2004B)]. 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. For non circular binary strings, [Jiang (2008A)] proposed a linear-time algorithm for Hamming distance with shifts, which generalises both Hamming distance and swap distance.

In [Toussaint (2002)], the swap distance measure of dissimilarity was shown to be more appropriate than several other measures of rhythm similarity including the

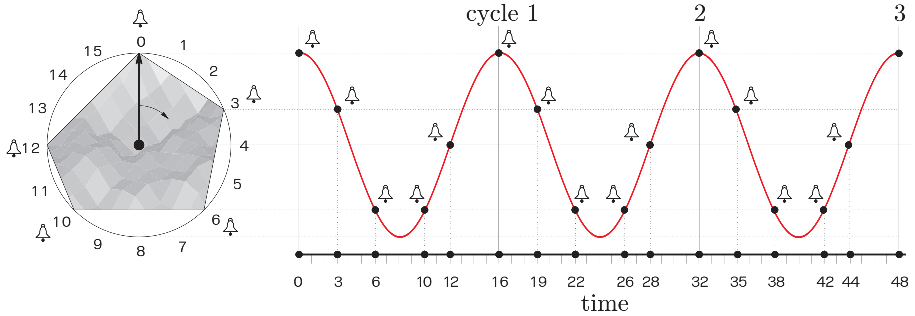


Fig. 2: 16-hours cycle clock representation of the clave *son* rhythm. The end of one cycle is the same spatial position as the beginning of the next.

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.

Toussaint stated that the swap distance between two rhythms represented as cyclic binary sequences may be computed in $O(n^2)$ time [Toussaint (2004A)]. He left as an open problem the possibility of improving this computation time. In this paper we aim to find an efficient algorithm to measure the swap distance between two cyclic binary sequences. 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 to B, minimized over all rotations of B. We show that this distance may be efficiently and elegantly computed in $O(k^2)$ time, assuming that the two necklaces are given using their compressed representations. Note that an additional $O(n)$ time is needed to compute the compressed representations if such representations are not given. Our algorithm is considered to be more efficient than Toussaint's when k is $o(n)$.

A closely related problems have been studied in [Bremner *et al.* (2006)] where several necklace alignment problems were defined and several $o(n^2)$-time convolution-based algorithm were proposed. Very recently, Jiang studied a different rhythmic similarity measure based on the sum of the distances along a circle [Jiang (2008B)].

The outline of the paper is as follows: Some preliminaries are described in Section 2. An $O(k^3)$-time algorithm is presented in Section 3 followed by a more efficient $O(k^2)$-time algorithm in Section 4. Conclusion and further discussion are drawn in Section 5.

2. Preliminaries

Let $X[0..n-1]$ be a necklace (circular string) of length n over $\Sigma = \{0, 1\}$. By $X[i]$ we denote the bit in X at position i, $0 \leq i < n$. We also denote by k, the number of 1's in X. Let $x = (x_0, x_1, \ldots, x_{k-1})$ be the *compressed* representation of X, such that $X[x_i] = 1$ for $0 \leq i < k$. For some integer r, let $x^{\langle r \rangle}$ be the r-*inverted-rotation* of x such that $x_i^{\langle r \rangle} = x_{i \circlearrowright r}$ for $0 \leq i < k$ and $i \circlearrowright r = \mathrm{mod}(i + r, k)$. If $X = [\texttt{10000100010001001}]$, for example, $x = (0, 5, 9, 13, 16)$, $x^{\langle 1 \rangle} = (5, 9, 13, 16, 0)$, and $x^{\langle 3 \rangle} = (13, 16, 0, 5, 9)$.

We define a *mapping* π:$\{1, \ldots, k\} \rightarrow \{1, \ldots, k\}$ such that π is a bijective (both onto and 1-1) function. We define the *non-crossing* mappings $\pi^0, \ldots, \pi^{k-1}$ as follows

$$\pi^h(i) = (i + h) \bmod k, \text{ for } 0 \leq i, h < k. \tag{1}$$

For $n = 5$, Figure 3 graphically illustrates the π^h mappings for $0 \leq h < 5$. All of these mappings are non-crossing , i.e. they have the property that their arrows never cross. We will show in Lemma 4.4 that only non-crossing mappings should be considered.

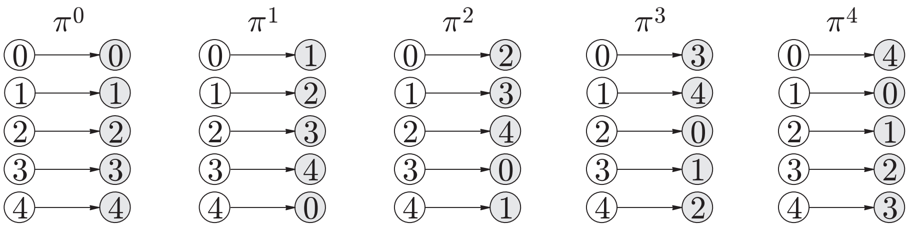


Fig. 3: π^h-mappings for $0 \leq h < 5$.

We define the *median* of a sorted sequence $x = (x_0, \ldots, x_{k-1})$ as follows:

$$x_{\mathrm{med}} = \begin{cases} x_{(k-1)/2} \ , & k \text{ odd}, \\ x_{\lfloor (k-1)/2 \rfloor}, & k \text{ even}. \end{cases} \tag{2}$$

Note that, when k is even, there are actually two medians, occurring at $\lfloor (k+1)/2 \rfloor$ (*lower median*) and $\lceil (k+1)/2 \rceil$ (*upper median*). For simplicity, Eq. 2 considers the lower median as "the median" when k is even.

Let $x = (x_0, x_1, ..., x_{k-1})$ and $y = (y_0, y_1, ..., y_{k-1})$ be two compressed representations of X and Y, respectively. Then the *Manhattan distance* $L_1(x, y)$ is defined as follows:

$$L_1(x, y) = \sum_{i=0}^{k-1} |y_i - x_i|. \tag{3}$$

Definition 2.1. *Given X and Y, two necklaces both of length n and same number of 1's, the* MINIMUM NECKLACE SWAP PROBLEM *is to find the cyclic swap distance* *between X and Y,* i.e., *the minimum number of swaps needed to convert X to 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.*

Throughout the paper we assume that the two necklaces are given using their compressed representations together with the length n. If this is not the case then an additional $O(n)$ time is needed to compute the compressed representations.

3. An $O(k^3)$-Time Algorithm

The naive approach is to examine each mapping and calculate for each possible rotation the sum of the swap operations needed for each pair of mapped 1's. This approach costs $O(nk^2)$ time. This is because k non-crossing mappings should be considered (*cf.* Lemma 4.4) and for each mapping, there are n rotations (circular shift) need to be examined. The question is: Do we really need to examine all possible n circular shifts for each mapping? Lemma 4.3 suggests that only k circular shifts need to be checked for each mapping. This gives a total cost of $O(k^3)$ time. The algorithm works as follows:

Let u and v be the *rest-interval* sequences for x and y, *resp.*, defined as follows

$$u_i = \begin{cases} x_{i+1} - x_i & , \text{ if } \ 0 \leq i < k-1 \\ n - x_{k-1} + x_0, & \text{if } \ i = k-1 \end{cases} \tag{4}$$

and

$$v_i = \begin{cases} y_{i+1} - y_i & , \text{ if } \ 0 \leq i < k-1 \\ n - y_{k-1} + y_0, & \text{if } \ i = k-1 \end{cases} \tag{5}$$

Using this representation we can now compute the following sequences

$$x_i^{[h]} = \begin{cases} u_i^{\langle h \rangle} & , \text{ if } \ i = 0 \\ (x_{i-1}^{[h]} + u_i^{\langle h \rangle}) \bmod n, & \text{if } \ 0 < i < k \end{cases} \tag{6}$$

and

$$y_i^{[h]} = \begin{cases} v_i^{\langle h \rangle} & , \text{ if } \ i = 0 \\ (y_{i-1}^{[h]} + v_i^{\langle h \rangle}) \bmod n, & \text{if } \ 0 < i < k \end{cases} \tag{7}$$

with the characteristic that for $x^{[i]}$ and $y^{[j]}, 0 \leq i, j < k$, the 1-bit in X at position i coincides with the 1-bit in Y at position j.

Example 3.1. *Let $x = (1, 6, 9, 12, 13)$ and $y = (0, 3, 4, 10, 16)$. To understand the meaning of u and v we first show the representations of x and y as bit strings X and Y in Table 1. Note that the 1's are numbered from 0 to $k-1$, for example the 1st and 2nd bit in X are located at positions 1 and 6. So, the rest-interval sequence*

$u = (5, 3, 3, 1, 5)$ *corresponds to the intervals between consecutive strokes (1's). In other words* u *stores the gaps between the 1's in* X*. In the same way, we compute* $v = (3, 1, 6, 6, 1)$.

Table 1: Illustration of $x^{[3]}$ and $y^{[1]}$ computation for $x = (1, 6, 9, 12, 13)$ and $y = (0, 3, 4, 10, 16)$, $u = (5, 3, 3, 1, 5)$ and $v = (3, 1, 6, 6, 1)$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X	0	1^0	0	0	0	0	1^1	0	0	1^2	0	0	1^3	1^4	0	0	0
Y	1^0	0	0	1^1	1^2	0	0	0	0	0	1^3	0	0	0	0	0	1^4
$X^{[3]}$	1^3	1^4	0	0	0	0	1^0	0	0	0	0	1^1	0	0	1^2	0	0
$Y^{[1]}$	1^1	1^2	0	0	0	0	0	1^3	0	0	0	0	0	1^4	1^0	0	0

Now let's understand the meaning of, lets say, $x^{[3]}$. *To compute* $x^{[3]}$*, according to Eq. 6, we need to calculate* $u^{\langle 3\rangle}$*, thus the 3*rd-inverted-rotation *of* u*. Since* $u = (5, 3, 3, 1, 5)$*, then* $u^{\langle 3\rangle} = (1, 5, 5, 3, 3)$ *(*i.e. u *was rotated to the left by 3 positions). Using* $u^{\langle 3\rangle}$ *and Eq. 6 we get the accumulated rotated intervals sequence* $x^{[3]} = (1, 6, 11, 14, 0)$. *Notice that sequence* $x^{[3]}$ *corresponds to a rotation of* X *such that its 4th 1-bit is set to position 0; this is equivalent to shifting* X *to the left by 12 positions. Table 1 also shows the representation of* $Y^{[1]}$ *which has the property of having the 2nd 1-bit in* Y *is set to position 0. Note that if we compute* $L_1(x^{[3]}, y^{[1]})$ *we compute the swap distance for* π^3 *(*cf. *Figure 3).*

The minimum necklace swap problem is equivalent to calculating

$$s^* = \min_{0 \le i,j < k} L_1(x^{[i]}, y^{[j]}). \tag{8}$$

Example 3.2. *For* $x = (1, 6, 9, 12, 13)$ *and* $y = (0, 3, 4, 10, 16)$ *Table 2 shows the computation of* $L_1(x^{[i]}, y^{[j]})$ *for* $0 \le i, j < 5$*. The value 3 was the minimum swap distance given by, for example, (3,4,9,14,0) and (3,4,10,16,0). This is also the number of swaps needed to match* [10011000010000100] *and* [10011000001000001].

Table 2: Computation of $L_1(x^{[i]}, y^{[j]})$ for $0 \le i, j < 5$. $x = (1, 6, 9, 12, 13)$, $y = (0, 3, 4, 10, 16)$, $u = (5, 3, 3, 1, 5)$ and $v = (3, 1, 6, 6, 1)$.

$u^{\langle i\rangle} \to x^{[i]}$	5,3,3,1,5 →	3,3,1,5,5 →	3,1,5,5,3 →	1,5,5,3,3 →	5,5,3,3,1 →
$v^{\langle j\rangle} \to y^{[j]}$	5,8,11,12,0	3,6,7,12,0	3,4,9,14,0	1,6,11,14,0	5,10,13,16,0
3,1,6,6,1 → 3,4,10,16,0	11	9	$\underline{3}$	7	11
1,6,6,1,3 → 1,7,13,14,0	9	11	9	$\underline{3}$	9
6,6,1,3,1 → 6,12,13,16,0	11	19	17	15	$\underline{3}$
6,1,3,1,6 → 6,7,10,11,0	4	8	10	10	12
1,3,1,6,6 → 1,4,5,11,0	15	7	9	11	23

Algorithm 1

```
Input: $x, y, k$
Output: $s^*$
1.  $\triangleright$ compute $u$ and $v$ using Eq. 4 and Eq. 5, resp.
2.  $s^* = \infty$
3.  for $i = 0$ to $k-1$ do
4.      for $j = 0$ to $k-1$ do
5.          $\triangleright$ compute $x^{[i]}$ and $y^{[j]}$ using Eq. 6 and Eq. 7, resp.
6.          if $s^* > L_1(x^{[i]}, y^{[j]})$ then $s^* = L_1(x^{[i]}, y^{[j]})$
7.  return $s^*$
```

Fig. 4: Algorithm 1.

Figure 4 shows the main steps of the algorithm. Line 5 can be computed in $O(k)$ time. Hence, Algorithm 1 runs in $O(k^3)$ time. By using the "high/low- frequency" technique [Abrahamson (1987)], Indyk *et. al.* proposed an algorithm to calculate all values $L_1(x^{[i]}, y^{[j]}), 0 \leq i, j < k$ in $O(k^{(\omega+3)/2})$ time, where $O(k^\omega)$ is the running time required to multiply two matrices of size $k \times k$ (see [Indyk *et al.* (2004)]). In the following section, we will show that we do not need to calculate all values $L_1(x^{[i]}, y^{[j]})$ in order to find s^*.

4. An $O(k^2)$-Time Algorithm

In this section we present the main algorithm for solving the minimum necklace swap problem. We show that only one column in Table 2 needs to be considered in order to compute the minimum swap distance, hence giving an overall time complexity of $O(k^2)$. Our strategy is to consider each non-crossing mapping π^h, for $0 \leq h < k$, in turn and to find for each mapping the rotation that minimises the swap distance. The algorithm works as follows:

If we define the *residual* sequence $c^{[h]}$ as

$$c_i^{[h]} = y_i^{[h]} - x_i^{[0]}, \text{ for } 0 \leq i, h < k, \tag{9}$$

then the minimum necklace swap problem is equivalent to calculating

$$s^* = \min_{0 \leq h < k} \sum_{i=0}^{k-1} |\delta_i^{[h]}|, \tag{10}$$

where

$$\delta_i^{[h]} = c_i^{[h]} - c_{\text{med}}^{[h]}, \text{ for } 0 \leq i, h < k, \tag{11}$$

and $c_{\text{med}}^{[h]}$ is the median of $c^{[h]}$ as defined in Eq. 2.

Algorithm 2

```
Input: $x, y, k$
Output: $s^*$
1.  ▷ compute $u$ and $v$ using Eq. 4 and Eq. 5, resp.
2.  ▷ compute $x^{[0]}$ using Eq. 6.
3.  $s^* = \infty$
4.  for $h = 0$ to $k-1$ do
5.      ▷ compute $y^{[h]}$ using Eq. 7
6.      ▷ compute $c^{[h]}$ using $y^{[h]}$, $x^{[0]}$ and Eq. 9.
7.      $c^{[h]}_{\text{med}} = \text{median}(c^{[h]})$
8.      ▷ compute $\delta^{[h]}$ using $c^{[h]}$, $c^{[h]}_{\text{med}}$ and Eq. 11.
9.      $d = 0$
10.     for $i = 0$ to $k-1$ do
11.         $d = d + |\delta^{[h]}_i|$
12.     if $s^* > d$ then $s^* = d$
13. return $s^*$
```

Fig. 5: Algorithm 2.

Figure 5 shows the main steps of the algorithm. Lines 1,2,5-6,8 can be computed in $O(k)$. Line 7 can be computed in $O(k)$ using [Reiser (1978)], therefore Algorithm 2 runs in $O(k^2)$.

Before proving the correctness of Algorithm 2, we give an example.

Example 4.1. *For* $x = (0, 5, 8, 11, 12)$ *and* $y = (0, 3, 4, 10, 16)$*, Table 3 shows the computation of* $c^{[h]}, c^{[h]}_{med}, \delta^{[h]}$*, and* $\sum_{i=0}^{k-1} |\delta^{[h]}_i|$ *for* $0 \leq h < k$*. The value 3 was the minimum swap distance between (5,8,11,12,0) and (6,7,10,11,0)*$^{\langle -1 \rangle}$*. This is also the number of swaps needed to match* [`10000100100110000`] *and* [`01000001100110000`]*. This example is also fully illustrated in Figure 6.*

Table 3: Computation of $c^{[h]}, c^{[h]}_{\text{med}}, \delta^{[h]}$, and $\sum_{i=0}^{k-1} |\delta_i^{[h]}|$ for $0 \leq h < k$. $x = (0, 5, 8, 11, 12)$, $y = (0, 3, 4, 10, 16)$, $u = (5, 3, 3, 1, 5)$, $v = (3, 1, 6, 6, 1)$.

h	$v^{\langle h \rangle} \rightarrow y^{[h]}$	$x^{[0]}$	$c^{[h]}$	$c^{[h]}_{\text{med}}$	$\delta^{[h]}$	$\sum_{i=0}^{k-1} \lvert\delta_i^{[h]}\rvert$
0	3,1,6,6,1 → 3,4,10,16,0	5,8,11,12,0	-2,-4,-1,4,0	-1	-1,-3,0,5,1	10
1	1,6,6,1,3 → 1,7,13,14,0	5,8,11,12,0	-4,-1,2,2,0	0	-4,-1,2,2,0	9
2	6,6,1,3,1 → 6,12,13,16,0	5,8,11,12,0	1,4,2,4,0	2	-1,2,0,2,-2	7
3	6,1,3,1,6 → 6,7,10,11,0	5,8,11,12,0	1,-1,-1,-1,0	-1	2,0,0,0,1	$\underline{3}$
4	1,3,1,6,6 → 1,4,5,11,0	5,8,11,12,0	-4,-4,-6,-1,0	-4	0,0,-2,3,4	9

Lemma 4.2. *For a given mapping π and two cyclic bit-strings X and Y, the rotation θ that minimises the swap distance between X and Y can be found in $O(k)$ time using Algorithm 2.*

Proof. Let's consider the mapping π^0, then we are trying to find the rotation θ that minimises the swap distance between X and Y under π^0. According to Eq. 1, π^0 pairs the 1's in X with the 1's in Y as follows:

$$(0 \rightarrow 0), (1 \rightarrow 1), \ldots, (k \rightarrow k).$$

Since the positions of the 1's in X and Y are stored in $x^{[0]}$ and $y^{[0]}$, *resp.*, vector $c^{[0]}$ stores the number of swaps needed to make each of the 1-bit in Y coincides with its corresponding 1-bit in X, but at the same time, since the signs are kept, it also "stores" the direction in which the swaps are to be done. Figure 7 illustrates this fact.

Now we seek to find the value θ such that

$$\sum_{i=0}^{k-1} |c_i^{[0]} - \theta| \tag{12}$$

is minimum. To minimise (12), θ needs to be chosen as the median of $c^{[0]}$ which can be calculated in $O(k)$ time using a linear time selection algorithm [Reiser (1978)]. Once θ has been computed, Eq. 12 can be calculated in linear time. Hence, Eq. 10 produces the minimum swap distance. □

Lemma 4.3. *In order to find the global minimum, the only rotations θ that need to be consider are those where two 1's coincide.*

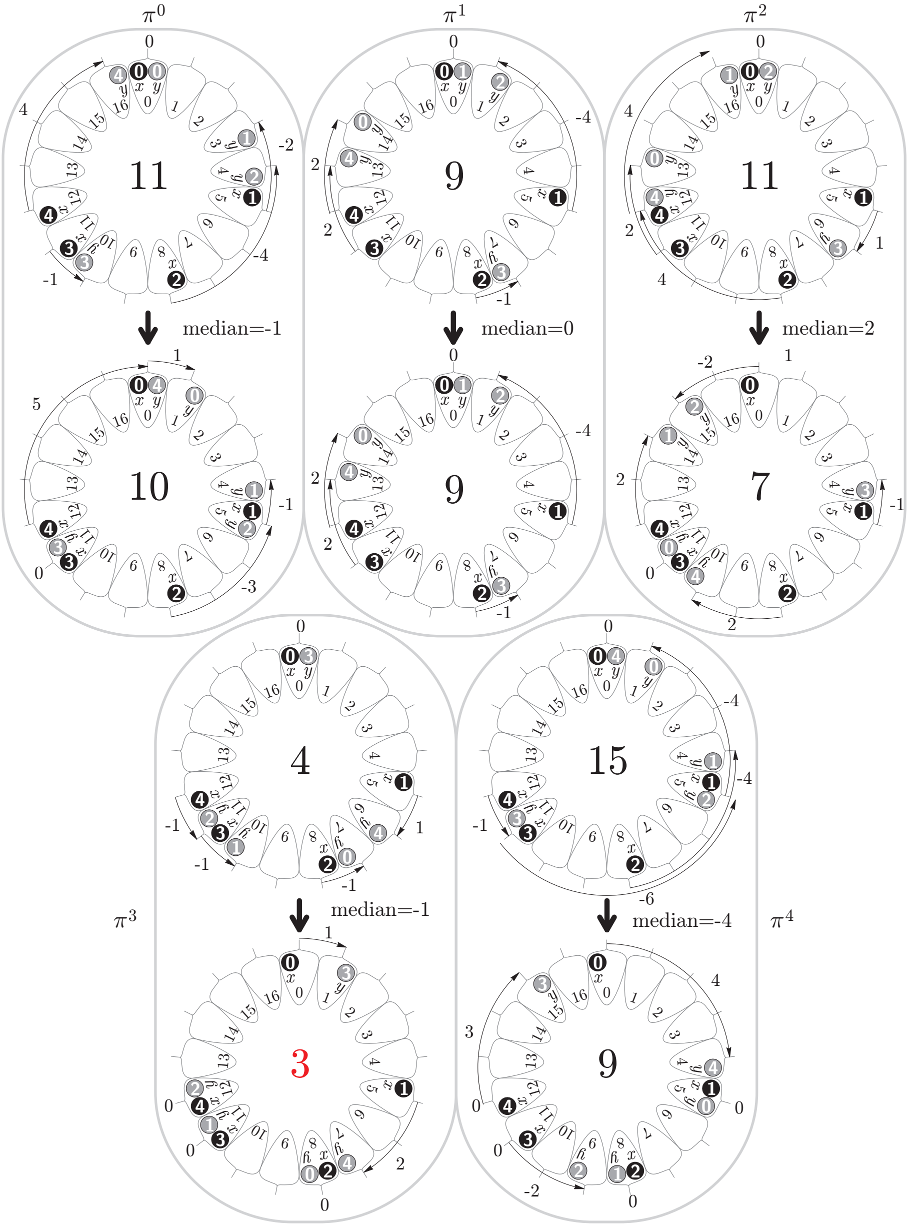


Fig. 6: Computation of the minimum cyclic swap distance for X = [`10000100100110000`] and Y = [`10011000001000001`]. $x = (0, 5, 8, 11, 12)$, $y = (0, 3, 4, 10, 16)$, $u = (5, 3, 3, 1, 5)$, $v = (3, 1, 6, 6, 1)$.

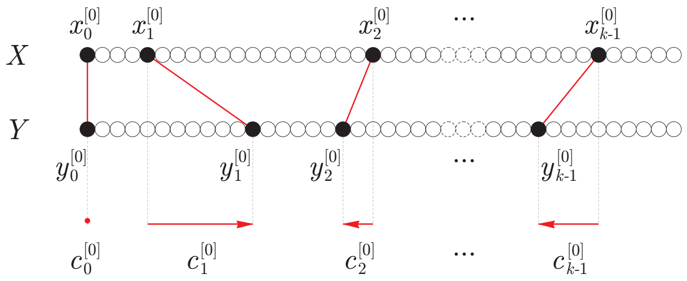


Fig. 7: Illustration of $c^{[0]}$ computation.

Proof. This follows from Lemma 4.2. For a given mapping h and residual sequence $c^{[h]}$; $c_{\text{med}}^{[h]}$ is a value in $c^{[h]}$ therefore, there must be at least one $\delta_i^{[h]} = 0$, for $0 \le i < k$. □

Lemma 4.4. *Consider cyclic string x and a linearised string y'. A minimum mapping for any shift is non-crossings.*

Proof. Consider any linearisation of string y and let y' be the concatenation of y to itself to make a string of length $2n$. Define π':$\{1, \ldots, n\} \rightarrow \{1, \ldots, 2n\}$ to be a mapping between the positions of the 1's in x and the 1's in $y's$ such that $\pi'(1) \le \pi'(i)$, for all i, and $\max_{i,j} \pi'(j) - \pi'(i) \le n$. It is clear that there is an 1-1 correspondence between mappings of this type and those defined in Section 2. We further extend the definition so that $\pi'^h(1)$ is equal to the position of the hth 1 in y'.

We say that a mapping, π', is *non-crossing* if for every i, j such that $i < j$ then $\pi'(i) < \pi'(j)$. Given a mapping π'^h (and a particular rotation of x), the swap distance with respect to π'^h is simply $\Sigma|\pi'^h(i) - i|$. We define a *minimum mapping* for a particular value h to be any mapping that minimises the swap distance.

The proof is by contradiction. Consider a mapping π' that both minimises the swap distance for the alignment and also is not a non-crossing. It follows that there exist i, j such that $j > i$ and $\pi'(i) > \pi'(j)$. Now consider a new mapping π'^*, identical to π' except that $\pi'^*(i) = \pi'(j)$ and $\pi'^*(j) = \pi'(i)$. Now $|\pi'^*(i) - i| < |\pi'(i) - i|$ and $|\pi'^*(j) - j| < |\pi'(j) - j|$ so the swap distance of π'^* is less than the swap distance of π'. This completes the proof. □

We can now prove the main theorem.

Theorem 4.5. *Algorithm 2 solves problem 1 in $O(k^2)$ time.*

Proof. Every set of swaps has a corresponding mapping from which its swap distance can be calculated. By Lemma 4.4, we need only to consider non-crossing mappings. By Lemma 4.2, Algorithm 2 finds the minimum swap distance for every mapping and as so by simply iterating over all non-crossing mappings the necklace problem is solved.

The running time of each iteration is determined by the time taken by Algorithm 2 which Lemma 4.2 shows to be $O(k)$. There are k iterations, giving an overall time complexity of $O(k^2)$. □

5. Conclusion and Further Discussion

We have presented a new algorithm that solves the cyclic swap distance problem for two n-bit (cyclic) binary strings in $O(k^2)$ where k is the number of 1's (same) in both strings. We have also shown that the swap distance calculated by our algorithm is optimal. It should be mentioned here that the ℓ_1 necklace alignment problem that has been recently studied by [Bremner *et al.* (2006)] are not exactly the same as the cyclic swap distance problem. Hence, the proposed convolution-based $o(k^2)$-time algorithm does not improve our result. In [Bremner *et al.* (2006)], the goal of the ℓ_1 necklace alignment problem is defined as finding the mapping s and the shift c that minimises $\sum_{i=0}^{k-1} |x_i - y_{i+s \mod k} - c|$, where x and y are the compressed representations of two necklaces. Similar to our result c is chosen to be the median over all $x_i - y_{i+s \mod n}$'s, for all possible non-crossing mapping $s \in \{0, 1, ..., k\}$. Thus, for two necklaces $X = 11100000$, $Y = 01100001$ represented as $x = 0, 1, 2$ and $y = 1, 2, 7$, the ℓ_1 distance is 4 ($s = 0$ and $c = 1$) while the cyclic swap distance calculated by our algorithm is 1.

Natural extensions to the cyclic swap distance problem could be:
(1) To consider unequal number of 1's among the input strings; this is known as the necklace Hamming distance with shift problem; proposed also in [Jiang (2008A)].
(2) To allow scaling i.e. a possible increase/decrease in the rest intervals of one of the input strings by a given constant.

Acknowledgment

The authors would like to express their gratitude to the referees for their knowledgeable and constructive comments that much improved the paper, and for Michael Bender for drawing the cyclic swap distance problem to our attention.

References

Abrahamson, K. (1987). Generalized string matching. *SIAM J. Comput.*, **16(6)**:1039–1051.

Ardila, Y. J. Pinzón, Clifford, R. and Mohamed, M. (2005). Necklace swap problem for rhythmic similarity measures. In *Proceedings of the International Symposium on String Processing and Information Retrieval*, M. Consens and G. Navarro, editors, *Lecture Notes in Computer Science*, Springer-Verlag, **3772**:235–246.

Ardila, Y. J. Pinzón, Iliopoulos, C. S., Landau, G. M. and Mohamed, M. (2005). Approximation algorithm for the cyclic swap problem. In J. Holub and M. Simnek, editors. In *Proceedings of the Prague Stringology Conference*, Czech Technical University, Prague, Czech Republic, pages 189–199.

Bremner, D., Chan, T. M., Erik, D. D., Erickson, J., Hurtado, F., Iacono, J., Langerman, S. and Taslakian, P. (2006). Necklaces, Convolutions, and X + Y. In *Proceedings of the 14th Annual European Symposium on Algorithms*, Zrich, Switzerland, pages 160–171.

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

Clifford, P. , Clifford, R. and Iliopoulos, C. S. (2005). Faster algorithms for δ,γ-matching and related problems. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, pages 68–78.

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

Clifford, R. and Iliopoulos, C. S. (2004). Approximate string matching for music analysis. *Soft Computing*, **8(9)**:597–603.

Crawford, T. , Iliopoulos, C. S. and Raman, R. (1998). String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, **11**:71–100.

Crochemore, M., Iliopoulos, C. S., Navarro, G. and Pinzon, Y. (2003). A bit-parallel suffix automaton approach for (δ,γ)-matching in music retrieval. In *Proceedings of the International Symposium on String Processing and Information Retrieval*, Springer-Verlag, pages 211–223.

Cronin, C. (1998). Concepts of melodic similarity in music-copyright infringement suits. In W.B. Hewlett and E. Selfridge-Field, editors, *Melodic Similarity: Concepts, Procedures and Applications*, MIT Press, Cambridge, Massachusetts, 1998.

Godwin, J. (1993). The Harmony of the spheres: A sourcebook of the pythagorean tradition in music. Inner Traditions Intl. Ltd.

Ghias, A. , Logan, J. , Chamberlin, D. and Smith, B. C. (1995). Query by humming: Musical information retrieval in an audio database. *ACM Multimedia*, pages 231-236.

Indyk, P., Lewenstein, M., Lipsky, O. and Porat, E. (2004). Closest pair problems in very high dimensions. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 782–792.

Jiang, M. (2008). A Linear-time algorithm for Hamming distance with shifts. *Theory of Computing Systems*, to appear.

Jiang, M. (2008). On the sum of distances along a circle. *Discrete Mathematics.* **308**:2038–2045.

Mo, J-S., Han, C. H. and Kim, Y-S. (1999). A melody-based similarity computation algorithm for musical information. In *Workshop on Knowledge and Data Engineering Exchange*, page 114.

Reiser, A. (1978). A linear selection algorithm for sets of elements with weights. *Inf. Process. Lett.*, **7(3)**:159–162.

Toussaint, G. T. (2002). A mathematical analysis of African, Brazilian, and Cuban clave rhythms. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, Townson University, Towson, MD, pages 157–168.

Toussaint, G. T. (2004). Computational geometric aspects of musical rhythm. In *Abstracts of the 14th Annual Fall Workshop on Computational Geometry*, Massachussetts Institute of Technology, pages 47–48.

Toussaint, G. T. (2004). A comparison of rhythmic similarity measures. In *Proceedings of the International Conference on Music Information Retrieval*, Barcelona, Spain, pages 242– 245. A longer version also appeared in: *School of Computer Science*, McGill University, Technical Report SOCS-TR-2004.6, August 2004.