

# The Distance Computation of the Non-overlapping Inversion and Transposition between Two Sequences

Tzu-Chiang Hsu, Chang-Biau Yang and Kuo-Tsung Tseng

**Abstract**—Due to the growing interest in large-scale mutation events, in 2016, Ta *et al.* introduced the mutation distance problem between two sequences of the same length  $n$  with two operations, non-overlapping inversions and transpositions. The time complexity of their algorithm for solving the problem is  $O(n^3)$ . In this paper, we propose an improved algorithm for solving the same problem with  $O(n^2)$  time. Our algorithm is based on the dynamic programming approach, and makes a strong connection between the non-overlapping transposition and the run structure.

**Index Terms**—Sequence Similarity, Non-overlapping, Inversion, Transposition, Repetition, Run Structure

## I. INTRODUCTION

DNA (deoxyribonucleic acid) is a couple of spiral strands, where each strand can be seen as a long sequence consisting of nucleotides a, t, c, g. Only two kinds of nucleotide combinations may appear. a and t can be combined together with two hydrogen bonds, and c and g with three hydrogen bonds. This fact leads to the *complementarity* of couples of strands. That is, for a nucleotide of a, t, c, or g in a strand, the corresponding position in the other strand is t, a, g, or c, respectively. Thus a strand of nucleotides can be determined by the other strand of the DNA, and then a double-strand DNA sequence can be represented with a single sequence of nucleotides due to its complementarity. For example, the DNA strand gattaca is the complementary strand of ctaatgt.

One of the most common ways to estimate the evolutionary relationships between different species in bioinformatics is the *edit distance* of two DNA sequences [16]. The most common mutation events for determining the edit distance of two DNA sequences are *point mutations* (or *single base modifications*) acting on a single nucleotide, including insertion, deletion, and substitution introduced by Levenshtein in 1966 [25]. The alignment problem is to compute the edit distance made by point mutations and it has been widely studied since 1974 [32, 38]. With the advance of bioinformatics, point mutations may not be sufficient to model gene evolution accurately. The *segmental mutations* acting on segments (mutations in several consecutive nucleotides) are called *genome rearrangements*

[9, 16], such as *reversal* (reversing a segment), *inversion* (reversing a segment and then substituting each nucleotide of the segment with its complement), *transposition* (exchanging two adjacent segments), and duplication. However, such genome rearrangement problems with unrestricted segmental mutations were proved to be NP-hard [7, 8]. Thus, some approximation algorithms were proposed [3, 4, 18, 22, 39] to deal with them.

Several algorithms for some special rearrangement problems were proposed recently. The alignment problem with unrestricted inversions does not have a known polynomial algorithm [37]. Schöniger and Waterman [33] solved the alignment problem with the restriction of non-overlapping inversions in  $O(n^6)$  time. Here, without loss of generality,  $n$  is assumed to be the longer length of the two aligned sequences. An improved algorithm with  $O(n^3)$  time was proposed by Vellozo *et al.* [37] in 2006.

The alignment problem with unrestricted transpositions was proved to be NP-hard [34]. The alignment problem with the restriction on non-overlapping transpositions, inversions and tandem duplications [24] was solved by Ledergerber and Dessimoz in  $O(n^4)$  time.

In 2016, Ta *et al.* [36] combined the aforementioned alignment problems with rearrangement events and permutations, and then defined the *non-overlapping inversion and transposition distance* problem. The problem is to determine the edit distance of two DNA sequences with the same length. The mutation events allowed in this variant alignment problem are non-overlapping inversions and transpositions, but without point mutations. The algorithm of Ta *et al.* for solving the problem requires  $O(n^3)$  time.

In this paper, we propose an improved algorithm to solve the non-overlapping inversion and transposition distance problem, based on the observation of relations between the non-overlapping transposition and the repetitions in sequences. The time complexity of our algorithm is  $O(n^2)$ , which is asymptotically  $n$  times faster than the algorithm of Ta *et al.*

The organization of this paper is as follows. Section II introduces the fundamentals of this paper. Section III describes the framework of our algorithm and the computation of equality and inversion matrices, which are needed in our algorithm. We then give a brute-force method to compute the transposition matrix and analyze its time complexity in Sections IV and V, respectively. Finally, Section VI derives the conclusion of our work.

Tzu-Chiang Hsu is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.

Chang-Biau Yang is with Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan. E-mail: cbyang@cse.nsysu.edu.tw. (Corresponding author)

Kuo-Tsung Tseng is with Department of Shipping and Transportation Management, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. E-mail: tsengkt@nkust.edu.tw.

## II. PRELIMINARIES

In this section, we first introduce the formal definition of the non-overlapping inversion and transposition distance problem, and then present the properties of repetitions and runs in a sequence, which are strongly relevant to the time complexity analysis of our algorithm.

### A. The Non-overlapping Inversion and Transposition Distance Problem

For a sequence  $A = a_1a_2 \cdots a_n$ , let  $A_{i..j}$  denote the substring of  $A$  from positions  $i$  to  $j$ . Two operations are available in our problem, defined as follows.

**Definition 1.** (inversion) *Given a sequence  $A$  of length  $n$ , the inversion operation  $\theta$  is defined as  $\theta_{i,j}(A) = A_{1..i-1}A_{i..j}^R A_{j+1..n}$ , where  $1 \leq i \leq j \leq n$ , and  $A_{i..j}^R$  is obtained from the reversely ordered  $A_{i..j}$ , then each character is replaced by its complement.*

**Definition 2.** (transposition [4]) *Given a sequence  $A$  of length  $n$ , the transposition operation  $\tau$  is defined as  $\tau_{i,j,k}(A) = A_{1..i-1}A_{j..k-1}A_{i..j-1}A_{k..n}$ , where  $1 \leq i < j < k \leq n+1$ . In other words,  $\tau_{i,j,k}$  divides  $A_{i..k-1}$  into two parts,  $A_{i..j-1}$  and  $A_{j..k-1}$ , then exchanges these two parts.*

The ranges of  $\theta_{i,j}$  and  $\tau_{i,j,k}$  are intervals  $[i, j]$  and  $[i, k-1]$ , respectively [36]. For example, suppose  $A = \text{tacgc}$ . Then  $\theta_{2,4}(A) = \text{tcgctc}$  and  $\tau_{2,4,6}(A) = \text{tgcac}$ . The ranges of  $\theta_{2,4}$  and  $\tau_{2,4,6}$  are  $[2, 4]$  and  $[2, 5]$ , respectively.

**Definition 3.** (non-overlapping inversion and transposition distance problem [36]) *Given two sequences  $A$  and  $B$  of length  $n$ , the non-overlapping inversion and transposition distance problem is to determine the minimum number of inversions and transpositions, denoted as  $md(A, B)$ , required to transform  $A$  into  $B$ , where the ranges of every two operations are disjoint. If it is unable to transform  $A$  into  $B$ , the distance is treated as infinity.*

Let  $\Phi$  denote a set consisting of inversion and transposition operations, where the two ranges of every two operations are disjoint. Let  $\Phi(A)$  denote the resulting sequence after all operations in  $\Phi$  applied to  $A$ . Since the ranges of the operations in  $\Phi$  are not overlapped, the order of operations has nothing to do with the resulting sequence.  $md(A, B)$  can be expressed in Equation 1 [36].

$$md(A, B) = \begin{cases} \min\{|\Phi| \mid \Phi(A) = B\} \\ \infty, \end{cases} \quad \text{if there is no such } \Phi. \quad (1)$$

For example, let  $A = \text{tacgc}$  and  $B = \text{acgtc}$ . Two sets  $\Phi_1 = \{\theta_{1,1}, \theta_{2,4}\}$  and  $\Phi_2 = \{\tau_{1,2,5}\}$  satisfy that  $\Phi_1(A) = B$  and  $\Phi_2(A) = B$ . Therefore,  $md(A, B) = \min\{2, 1\} = 1$ .

### B. Repetitions and Runs

The detecting and locating repetitions in a sequence is a fundamental topic in combinatorics on words [2, 6, 10, 26]. There are many applications, such as pattern matching [17], text compression [40], and computational bioinformatics [27].

The *period*  $p$  of a sequence  $S$ , denoted by  $p(S)$ , is defined as the minimum positive integer satisfying that  $s_i = s_{i+p}$ , for all  $i, i+p \in [1, |S|]$ . If there is no such integer, then  $p(S) = |S|$ . The *exponent* of  $S$ , denoted by  $e(S)$ , is defined as  $\frac{|S|}{p(S)}$ . A sequence  $S$  is a *repetition* if  $e(S)$  is an integer greater than 1. Then,  $S$  can be represented as  $u^{e(S)} = \underbrace{uu \dots u}_{e(S)}$ , where  $u = S_{1..p(S)}$ .

For example,  $\text{atatat} = (\text{at})^3$  is a repetition with exponent 3 and period 2. But  $\text{atata}$  is not a repetition since its exponent is 2.5, not an integer. The *maximal repetition* is a repetition with maximal exponent, which cannot be extended to the left nor right with one period. For example, the maximal repetitions in  $A = \text{ttatatataaaat}$  are  $A_{1..2} = \text{tt} = \text{t}^2$ ,  $A_{9..11} = \text{aaa} = \text{a}^3$ ,  $A_{2..9} = \text{tatata} = (\text{ta})^4$ , and  $A_{3..8} = \text{atatat} = (\text{at})^3$ .

Several algorithms for positioning all occurrences of maximal repetitions in a sequence were proposed. The algorithm proposed by Crochemore [10] is based on the minimization algorithm of *finite state automata* [1], and the algorithm proposed by Apostolico and Preparata [2] is based on the *suffix tree* technique [31]. Main and Lorentz [30] also proposed an algorithm for positioning all occurrences of squares, where a square is a substring in the form  $uu$ . For example, let  $A = \text{atatatat}$ , there are squares  $A_{1..4} = A_{3..6} = A_{5..8} = (\text{at})(\text{at})$ ,  $A_{2..5} = A_{4..7} = (\text{ta})(\text{ta})$ , and  $A_{1..8} = (\text{atat})(\text{atat})$ . The time complexities of these three algorithms are all  $O(n \log n)$ , where  $n$  denotes the sequence length. Besides, these algorithms have been proved to be optimal [2, 10, 30].

Though the number of repetition occurrences is  $\Omega(n \log n)$  [2, 10, 30], the computation and representation of all repetition occurrences in a sequence can be done in linear time and space. The *run* structure, first introduced by Iliopoulos *et al.* [20] in 1997, represents repetitions in a compact way. Later in 1999, Kolpakov and Kucherov [23] showed that the number of runs is linear to the sequence length, and they proposed an algorithm for computing all runs in a sequence with linear time, based on the algorithm of Main [29]. To find the first occurrence position of all distinct runs in a sequence, the algorithm of Main [29] requires linear time, utilizing the concept of the sequence decomposition, called the *s-factorization*, proposed by Crochemore [11, 15].

Formally, a *run*  $\Gamma = A_{i..j}$  is a substring of  $A$ , where  $e(\Gamma) \geq 2$ ,  $p(A_{i-1..j}) > p(\Gamma)$  and  $p(A_{i..j+1}) > p(\Gamma)$ . That is,  $|\Gamma| \geq 2 \times p(\Gamma)$ , and the extension of  $\Gamma$  to the right or left by one more character yields a larger period. Any substring  $\Gamma'$  in a run  $\Gamma$  is a repetition if and only if  $|\Gamma'|$  is an integer multiple at least two times of  $p(\Gamma)$ .

For example, let  $A = \text{taaataataaataataataata}$ . All runs in  $A$  are illustrated in Figure 1. There are 6 runs of period 1 (a), which are  $A_{2..4}$ ,  $A_{6..7}$ ,  $A_{9..11}$ ,  $A_{13..14}$ ,  $A_{16..17}$ , and  $A_{19..20}$ . Two runs of period 3 (aat), which are  $A_{3..10}$  and  $A_{10..22}$ . And  $A_{6..14}$  and  $A_{1..17}$  are runs of periods 4 (aata) and 7 (taaataa), respectively. Each substring  $\Gamma'$  of the run  $A_{10..22}$  is a repetition if and only if  $|\Gamma'| \in \{6, 9, 12\}$ .



Fig. 1: All runs in  $A = \text{taaataataaataataataa}$  [13].

These repetitions are  $A_{10+x..15+x}$  with length 6 for  $x \in [0, 7]$ ,  $A_{10+x..18+x}$  with length 9 for  $x \in [0, 4]$ ,  $A_{10..21}$  and  $A_{11..22}$  with length 12.

The work of Kolpakov and Kucherov [23] is a breakthrough that all repetition occurrences can be positioned efficiently. The maximal number of runs in a sequence of length  $n$  is proved to be bounded between  $0.944n$  [35] and  $1.029n$  [12]. Furthermore, Kolpakov and Kucherov [23] proved that the sum of exponents of runs in a sequence of length  $n$  is  $O(n)$ . The precise bound was claimed to be less than  $4.1n$  by Crochemore *et al.* [14] in 2012, and a tighter bound  $3n$  was given by Bannai *et al.* [5] in 2014.

The time complexity analysis of our algorithm is based on the properties of the run structure.

### III. THE MAIN CONCEPT OF OUR ALGORITHM

Though the algorithm of Ta *et al.* [36] seems complicated, by observing the data dependency, we can simplify their algorithm by the dynamic programming formula as follows.

$$\text{Dist}(j) = \begin{cases} \min_{1 \leq i \leq j} \{ \text{Dist}(i-1) + \Psi(i, j) \}, & \text{if } 1 \leq j \leq n, \\ 0, & \text{if } j = 0. \end{cases} \quad (2)$$

In the above formula,  $\text{Dist}(j)$  denotes the distance of  $A_{1..j}$  and  $B_{1..j}$  with non-overlapping inversions and transpositions.  $\Psi(i, j)$  is the minimum cost that transforms  $A_{i..j}$  into  $B_{i..j}$  with an inversion or a transposition on range  $[i, j]$ , or  $\Psi(i, j) = 0$  if  $A_{i..j} = B_{i..j}$  (0 operation). If none of the above conditions is satisfied, we set  $\Psi(i, j) = \infty$ .

For example, suppose that  $A = \text{ttctttaagt}$  and  $B = \text{ttaagtctt}$ .  $A_{2..8}$  can be transformed into  $B_{2..8}$  by transposing  $A_{2..4} = \text{tct}$  and  $A_{5..8} = \text{taag}$ . Thus, we have  $\Psi(2, 8) = 1$ .  $A_{3..5} = \text{ctt}$  can be transformed into  $B_{3..5} = \text{aag}$  by an inversion, thus  $\Psi(3, 5) = 1$ .  $A_{1..2}$  can be transformed into  $B_{1..2}$  by transposing  $A_{1..1}$  and  $A_{2..2}$ , but  $A_{1..2} = B_{1..2}$ . Thus  $\Psi(1, 2) = 0$  since no operation is needed.  $A_{1..3}$  cannot be transformed into  $B_{1..3}$  by an inversion or a transposition, thus  $\Psi(1, 3) = \infty$ .

In the algorithm of Ta *et al.*,  $\Psi(i, j)$  was calculated by a linear scan, which requires  $O(j - i + 1) = O(n)$  time. The linear scan method is the bottleneck of the algorithm. The main improvement of our algorithm is to build the whole matrix  $\Psi$  in  $O(n^2)$  time, and then each  $\Psi(i, j)$  can be queried in constant time.

In the preprocessing phase, to compute  $\Psi$  efficiently, our algorithm prepares three auxiliary matrices, *equality matrix*

( $EQ$ ), *inversion matrix* ( $IV$ ), and *transposition matrix* ( $TP$ ). We give the definitions of these matrices in Definition 4.

**Definition 4.** ( $EQ$ ,  $IV$ , and  $TP$ ) Let  $A$  and  $B$  be two sequences of the same length  $n$ , and  $1 \leq i \leq j \leq n$ .  $EQ(i, j) = 0$  if and only if  $A_{i..j} = B_{i..j}$ ;  $IV(i, j) = 1$  if and only if  $A_{i..j}^R = B_{i..j}$ ;  $TP(i, j) = 1$  if and only if  $A_{i..j}$  can be transformed into  $B_{i..j}$  by a transposition operation on the whole  $A_{i..j}$  (There exists  $k \in [i+1, j]$  such that  $(\tau_{i,k,j+1}(A))_{i..j} = B_{i..j}$ ). Otherwise,  $EQ(i, j) = \infty$ ,  $IV(i, j) = \infty$ , or  $TP(i, j) = \infty$ .

For example, suppose that  $A = \text{ttctttaagt}$  and  $B = \text{ttaagtctt}$ .  $TP(2, 8) = 1$  since  $A_{2..8} = \text{tctttaag}$  can be transformed into  $B_{2..8} = \text{taagtctt}$  by transposing  $A_{2..4} = \text{tct}$  and  $A_{5..8} = \text{taag}$ .  $IV(3, 5) = 1$  since  $A_{3..5} = \text{ctt}$  can be transformed into  $B_{3..5} = \text{aag}$  by an inversion.  $EQ(1, 2) = 0$  since  $A_{1..2} = B_{1..2} = \text{tt}$ .

In Definition 4, note that  $TP(i, j) = 1$  if  $A_{i..j} = B_{i..j}$ . It can be viewed as the special case that the empty substring at the last position of  $A_{i..j}$  is moved to the front of itself, and the resulting is  $B_{i..j}$ . However, the traditional transposition is defined as an exchange of two adjacent nonempty substrings. Here, we relax the restriction for allowing empty substrings. It can be treated as a generalized transposition.

$EQ(i, j)$ ,  $IV(i, j)$  and  $TP(i, j)$  are the minimum cost of a single equality, inversion and transposition operation, respectively, on range  $[i, j]$ . Thus, we have

$$\Psi(i, j) = \min\{EQ(i, j), IV(i, j), TP(i, j)\}. \quad (3)$$

Our algorithm for calculating the distance of the non-overlapping inversion and transposition between two sequences is presented in Algorithm 1.

**Algorithm 1** Computation of the non-overlapping inversion and transposition distance

- 1: Compute  $EQ$ ,  $IV$ , and  $TP$
- 2:  $\Psi(i, j) := \min\{EQ(i, j), IV(i, j), TP(i, j)\}$ , for  $1 \leq i \leq j \leq n$
- 3:  $\text{Dist}(0) := 0$
- 4:  $\text{Dist}(j) := \min_{1 \leq i \leq j} \{ \text{Dist}(i-1) + \Psi(i, j) \}$ , for  $1 \leq j \leq n$
- 5: **return**  $\text{Dist}(n)$

Figure 2 shows the contents of  $\Psi$ ,  $EQ$ ,  $IV$ ,  $TP$ , and  $\text{Dist}$  for two sequences  $A = \text{ttctttaagt}$  and  $B = \text{ttaagtctt}$ . Each empty spaces in the matrices represents  $\infty$ . The solution  $\text{Dist}(9) = 1$ , since it can be obtained from  $\text{Dist}(1) + \Psi(2, 8) + \Psi(9, 9) = EQ(1, 1) + TP(2, 8) + EQ(9, 9) = 0 + 1 + 0 = 1$ , or  $\text{Dist}(2) + \Psi(3, 9) = EQ(1, 2) + TP(3, 9) = 0 + 1 = 1$ .

It is clear that the time required for computing line 4 in Algorithm 1 is  $O(n^2)$ . Hence, if the matrices  $EQ$ ,  $IV$ , and  $TP$  can be computed in  $O(n^2)$  time, then the time complexity of Algorithm 1 is  $O(n^2)$ .

The computation of the equality matrix  $EQ$  is presented in Procedure 1. It is obvious that it can be done in  $O(n^2)$  time, since these maximal intervals are pairwise disjoint.

| $\Psi$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|--------|---|---|---|---|---|---|---|---|---|
| t      | 0 | 0 |   |   |   |   |   |   |   |
| t      |   | 0 |   |   |   |   |   | 1 |   |
| c      |   |   |   | 1 |   |   |   |   | 1 |
| t      |   |   |   |   | 1 |   |   |   |   |
| t      |   |   |   |   |   |   |   |   |   |
| a      |   |   |   |   |   | 1 |   |   |   |
| a      |   |   |   |   |   |   |   | 1 |   |
| g      |   |   |   |   |   |   |   |   |   |
| t      |   |   |   |   |   |   |   |   | 0 |

| $EQ$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|---|---|---|---|---|---|---|---|---|
| t    | 0 | 0 |   |   |   |   |   |   |   |
| t    |   | 0 |   |   |   |   |   |   |   |
| c    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| g    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |

| $Dist$ | 1 | 2 | 3        | 4        | 5 | 6 | 7        | 8 | 9 |
|--------|---|---|----------|----------|---|---|----------|---|---|
|        | 0 | 0 | $\infty$ | $\infty$ | 1 | 2 | $\infty$ | 1 | 1 |

| $IV$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|---|---|---|---|---|---|---|---|---|
| t    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |
| c    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| g    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |

| $TP$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|---|---|---|---|---|---|---|---|---|
| t    | 1 | 1 |   |   |   |   |   |   |   |
| t    |   | 1 |   |   |   |   |   | 1 |   |
| c    |   |   |   |   |   |   |   |   | 1 |
| t    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| a    |   |   |   |   |   |   |   |   |   |
| g    |   |   |   |   |   |   |   |   |   |
| t    |   |   |   |   |   |   |   |   | 1 |

Fig. 2: The contents of  $\Psi$ ,  $EQ$ ,  $IV$ ,  $TP$ , and  $Dist$  for  $A = \text{ttctttaagt}$  and  $B = \text{ttaagtctt}$ , where each empty space in the matrices is  $\infty$ .

#### Procedure 1 Computation of the equality matrix $EQ$

- 1:  $EQ(i, j) := \infty$ , for  $1 \leq i \leq j \leq n$
- 2: **for each** maximal interval  $[i, j]$  with  $A_{i..j} = B_{i..j}$  **do**
- 3:  $EQ(i', j') := 0$ , for  $i \leq i' \leq j' \leq j$

The computation of the inversion matrix  $IV$  is presented in Procedure 2. The main idea is to extend the range from one center position (for odd length) or two center positions (for even length) to the left and right. Obviously, the computation of  $IV$  requires  $O(n^2)$  time.

#### Procedure 2 Computation of the inversion matrix $IV$

- 1:  $IV(i, j) := \infty$ , for  $1 \leq i \leq j \leq n$
- 2: **for**  $i := 1 \rightarrow n$  **do** // Check inversion of odd length
- 3: **for**  $\delta := 0 \rightarrow \min\{i-1, n-i\}$  **do** // Extend from  $i$  to the left and right
- 4: **if**  $\overline{a_{i-\delta}} = b_{i+\delta}$  and  $\overline{a_{i+\delta}} = b_{i-\delta}$  **then**
- 5:  $IV(i-\delta, i+\delta) := 1$
- 6: **else**
- 7: **break**
- 8: **for**  $i := 1 \rightarrow n$  **do** // Check inversion of even length
- 9: **for**  $\delta := 0 \rightarrow \min\{i-1, n-(i+1)\}$  **do** // Extend from  $i$  and  $i+1$  to the left and right
- 10: **if**  $\overline{a_{i-\delta}} = b_{(i+1)+\delta}$  and  $\overline{a_{(i+1)+\delta}} = b_{i-\delta}$  **then**
- 11:  $IV(i-\delta, (i+1)+\delta) := 1$
- 12: **else**
- 13: **break**

## IV. TRANSPOSITION COMPUTATION WITH A BRUTE-FORCE METHOD

To compute the transposition matrix  $TP$  efficiently is not so easy, thus we first present a brute-force method for an overview. An improved method will be given in the next section.

**Definition 5.** (cyclic shift) A cyclic shift operation  $\sigma$  on a sequence  $A = a_1a_2 \dots a_n$  is defined as Equation 4 (defined by Maes [28]).

$$\sigma^k(A) = \begin{cases} A, & \text{if } k = 0 \\ a_2a_3 \dots a_na_1, & \text{if } k = 1 \\ \sigma^{k-1}(\sigma(A)), & \text{if } k > 1, \end{cases} \quad (4)$$

where  $\sigma^k$  moves the first  $k$  characters to the end of the sequence. We further generalize the definition for a negative exponent  $k$  without violation of the original meaning as follows.

$$\sigma^k(A) = \begin{cases} \sigma^{k+1}(\sigma^{-1}(A)), & \text{if } k < -1 \\ a_na_1a_2 \dots a_{n-1}, & \text{if } k = -1 \\ A, & \text{if } k = 0 \\ a_2a_3 \dots a_na_1, & \text{if } k = 1 \\ \sigma^{k-1}(\sigma(A)), & \text{if } k > 1. \end{cases} \quad (5)$$

Note that the operation  $\sigma$  is commutative and associative. With Equation 5,  $TP$  can be redefined in Definition 6.

**Definition 6.** ( $TP$ ) Given two sequences  $A$  and  $B$  of the same length  $n$ , for  $1 \leq i \leq j \leq n$ ,

$$TP(i, j) = \begin{cases} 1, & \text{if } \exists k \in [0, j-i] \ni \sigma^k(A_{i..j}) = B_{i..j} \\ \infty, & \text{otherwise.} \end{cases} \quad (6)$$

**Definition 7.** ( $\lambda$ ) Given two sequences  $A$  and  $B$  of the same length  $n$ ,  $\lambda(i, j)$  represents the maximal equality extension length from  $a_i$  and  $b_j$ , i.e.  $\lambda(i, j) = \min\{\ell \mid a_{i+\ell} \neq b_{j+\ell}, \ell \geq 0\}$ , for  $i, j \in [1, n]$ .

For example, suppose that  $A = \text{atattattattat}$  and  $B = \text{tattattattata}$ . The contents of  $\lambda$  is shown in the left part of Figure 3.  $\lambda(2, 4) = 9$  indicates that exactly 9 consecutive characters of  $A$  and  $B$  are equal, that is,  $A_{2..10} = B_{4..12}$ , but  $A_{2..11} \neq B_{4..13}$ .  $\lambda(6, 2) = 8$  means  $A_{6..13} = B_{2..9}$ . In addition,  $\lambda(2, 4) = 9 \geq 4 = 6 - 2$  and  $\lambda(6, 2) = 8 \geq 2 = 4 - 2$ , we get  $A_{2..5} = B_{4..7}$  and  $A_{6..7} = B_{2..3}$ . Thus,  $A_{2..7}$  can be transformed into  $B_{2..7}$  by transposing  $A_{2..5}$  and  $A_{6..7}$ . Accordingly,  $TP(2, 7) = 1$ . Similarly,  $TP(2, 4) = 1$  since  $A_{2..4}$  can be transformed into  $B_{2..4}$  by transposing  $A_{2..2}$  and  $A_{3..4}$ .

Now, we can formulate the  $\lambda$  matrix to get the  $TP$  matrix. See Figure 4 for illustration. If  $A_{i..i+d-1} = B_{(i+r)..(i+r)+d-1}$  and  $A_{(i+d)..(i+d)+r-1} = B_{i..i+r-1}$ , then  $A_{i..i+r+d-1}$  can be transformed into  $B_{i..i+r+d-1}$  by transposing  $A_{i..i+d-1}$  and  $A_{i+d..i+r+d-1}$ . That is,  $TP(i, i+r+d-1) = 1$ . Obviously, we should have  $\lambda(i, i+r) \geq d$  and  $\lambda(i+d, i) \geq r$ , which derives Theorem 1.

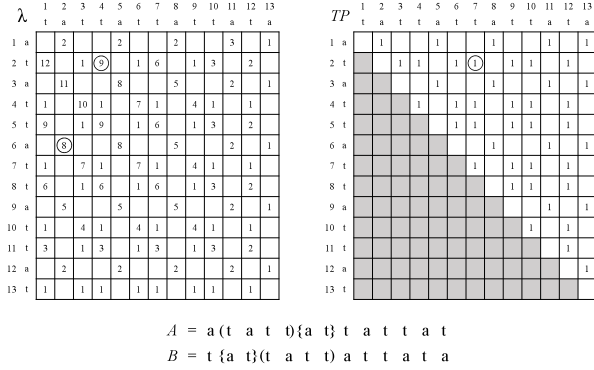


Fig. 3: The contents of  $\lambda$  and  $TP$  for  $A = a t a t t a t t a t t a t$  and  $B = t a t t a t t a t t a t a$ , where each empty space in  $\lambda$  and  $TP$  is 0 and  $\infty$ , respectively.

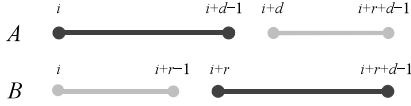


Fig. 4: A transposition operation on range  $[i, i + r + d - 1]$ .

**Theorem 1.** Let  $A$  and  $B$  be two sequences of the same length  $n$ , and  $i, r, d$  be non-negative integers, where  $i \in [1, n]$  and at least one of  $r$  and  $d$  is not zero. The following conditions are equivalent.

- 1)  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$ .
- 2)  $\sigma^d(A_{i..i+r+d-1}) = \sigma^{-r}(A_{i..i+r+d-1}) = B_{i..i+r+d-1}$ .
- 3)  $TP(i, i + r + d - 1) = 1$ .

*Proof.* For the case that both  $d$  and  $r$  are not zero ( $d \geq 1$  and  $r \geq 1$ ), if  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$ , then  $A_{i..i+(d-1)} = B_{i+r..i+r+(d-1)}$  and  $A_{i+d..i+d+(r-1)} = B_{i..i+(r-1)}$ , thus  $\sigma^d(A_{i..i+r+d-1}) = \sigma^{-r}(A_{i..i+r+d-1}) = B_{i..i+r+d-1}$ . If one of  $r$  and  $d$  is zero, then  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$  derives that  $A_{i..i+r+d-1} = B_{i..i+r+d-1}$ , thus  $\sigma^d(A_{i..i+r+d-1}) = \sigma^{-r}(A_{i..i+r+d-1}) = B_{i..i+r+d-1}$ . Both cases yield that  $TP(i, i + r + d - 1) = 1$  by Definition 6.

If  $TP(i, j) = 1$ , then there exists  $k \in [0, j - i]$  such that  $\sigma^k(A_{i..j}) = B_{i..j}$ . For the case that  $k \neq 0$ ,  $A_{i+k..j} = B_{i..j-k}$  and  $A_{i..i+k-1} = B_{j-k+1..j}$ , thus  $\lambda(i, i + r) \geq d$  and  $\lambda(i + d, i) \geq r$ , for  $r = j - i - k + 1 \geq 0$  and  $d = k > 0$ . If  $k = 0$ , then  $A_{i..j} = B_{i..j}$ , thus  $\lambda(i, i + r) \geq d$  and  $\lambda(i + d, i) \geq r$  hold for  $r = 0$  and  $d = j - i + 1$ .  $\square$

According to Theorem 1, Procedure 3 exhaustively examines all combinations of  $i, r$  and  $d$ , thus the transposition matrix  $TP$  is correctly computed. Since the computation of  $\lambda$  can be done in  $O(n^2)$  time, Procedure 3 requires  $O(n^3)$  time.

---

**Procedure 3** Computation of the transposition matrix  $TP$  with a brute-force method

---

- 1:  $TP(i, j) := \infty$ , for  $1 \leq i \leq j \leq n$
  - 2: Compute  $\lambda$
  - 3: **for**  $i := 1 \rightarrow n$  **do**
  - 4:     **for**  $d := 0 \rightarrow n - i$  **do**
  - 5:         **for**  $r := 0 \rightarrow n - i$  **do**
  - 6:             **if**  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$  and  
               ( $d > 0$  or  $r > 0$ ) **then**
  - 7:                  $TP(i, i + r + d - 1) := 1$
- 

Figure 3 shows an example for computing  $TP$  from  $\lambda$  with  $A = a t a t t a t t a t t a t$  and  $B = t a t t a t t a t t a t a$ . In the figure,  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$  for  $\langle i, r, d \rangle = \langle 2, 2, 4 \rangle$ . It is equivalent to that  $\lambda(2, 4) = 9 \geq 4$  and  $\lambda(6, 2) = 8 \geq 2$ . Thus,  $TP(2, 7) = 1$ . For  $\langle i, r, d \rangle = \langle 2, 2, 1 \rangle$ , we have  $\lambda(2, 4) = 9 \geq 1$  and  $\lambda(3, 2) = 11 \geq 2$ . Thus,  $TP(2, 4) = 1$ .

In Procedure 3, since at least one of  $d$  and  $r$  is not zero, and the roles of  $d$  and  $r$  are symmetric, we can simply set the starting values of  $d$  and  $r$  to 0 and 1, respectively. Furthermore, we replace the loop of  $r$  by the linked list described in Definition 8, and then it can be traversed linearly. With slight modification, we rewrite the computation of  $TP$  in Procedure 4.

**Definition 8.**  $\mathcal{L}_i$  is a singly linked list, for  $i \in [1, n]$ , where  $\mathcal{L}_i$  is initialized to a list of integers from 1 to  $n - i$  in increasing order.  $\mathcal{L}_i.first$  and  $\mathcal{L}_i.last$  denote the first and last elements of  $\mathcal{L}_i$ , respectively.

---

**Procedure 4** Linked implementation for transposition computation from Procedure 3

---

- 1:  $TP(i, j) := \infty$ , for  $1 \leq i \leq j \leq n$
  - 2: Compute  $\lambda$
  - 3: **for**  $i := 1 \rightarrow n$  **do**
  - 4:     Construct linked list  $\mathcal{L}_i$     // integers from 1 to  $n - i$
  - 5:     **for**  $d := 0 \rightarrow n - i$  **do**
  - 6:         **for**  $r := \mathcal{L}_i.first \rightarrow \mathcal{L}_i.last$  **do**
  - 7:             **if**  $\min\{\lambda(i, i + r) - d, \lambda(i + d, i) - r\} \geq 0$  **then**
  - 8:                  $TP(i, i + r + d - 1) := 1$
  - 9:             **else if**  $r > \lambda(i + d, i)$  **then**    // also true for  
   larger  $r$
  - 10:                 **break**
  - 11:             **else**    //  $d > \lambda(i, i + r)$ . Also true for larger  $d$
  - 12:                 Remove  $r$  from  $\mathcal{L}_i$     // need not check this  
   for larger  $d$
- 

In Procedure 4, if the condition in line 7 is not satisfied, then it should be either  $r > \lambda(i + d, i)$  or  $d > \lambda(i, i + r)$ , which are the conditions in lines 9 and 11, respectively. Since  $r$  is increasing in the loop, once  $r > \lambda(i + d, i)$  happens, then every element in  $\mathcal{L}_i$  following  $r$  is greater than  $\lambda(i + d, i)$ , thus we can just skip the loop (done in line 10). Similarly, since  $d$  is increasing,  $d > \lambda(i, i + r)$  holds for every incremented  $d$ ,

we need not check this  $r$  any longer and then we can remove  $r$  from  $\mathcal{L}_i$ .

## V. AN ANALYSIS OF THE BRUTE-FORCE METHOD

Let  $\text{cnt}(i, j)$  denote the count that  $TP(i, j) := 1$  is executed in line 8 of Procedure 4. Note that one pair  $(i, j)$  may be counted multiple times if  $TP(i, j)$  is repeatedly set to 1. From Theorem 1 and with the constraint that  $r > 0$  in Procedure 4, we derive that  $\text{cnt}(i, j) = |\{k \in [0, j - i] \mid \sigma^k(A_{i..j}) = B_{i..j}\}|$ .

There are some properties for cases that  $\text{cnt}(i, j) \geq 2$ . We define the *paired sequence* to abstractly treat two sequences with the same length as one sequence.

**Definition 9.** (paired sequence) Let  $A = a_1a_2 \dots a_n$  and  $B = b_1b_2 \dots b_n$  be two sequences. The paired sequence  $W = w_1w_2 \dots w_n$  is generated by sequences  $A$  and  $B$ , where  $w_i = \langle a_i, b_i \rangle$ .  $w_i = w_j$  if and only if both  $a_i = a_j$  and  $b_i = b_j$  hold.

For example, let  $A$  and  $B$  be two sequences of the same length, and  $W$  be the paired sequence generated by  $A$  and  $B$ , where  $A_{i..j} = \text{tattattat}$  and  $B_{i..j} = \text{attattatt}$ .  $\sigma^k(A_{i..j}) = B_{i..j}$  only for  $k \in \{1, 4, 7\}$ , then  $\text{cnt}(i, j) = |\{1, 4, 7\}| = 3$ . We discover that the necessary condition for  $\text{cnt}(i, j) = 3$  is that exponent  $e(W_{i..j}) = 3$ . That is, both  $A_{i..j}$  and  $B_{i..j}$  are repetitions of exponent 3. More precisely, if  $\text{cnt}(i, j) \geq 2$ , then  $W_{i..j}$  is a repetition and  $\text{cnt}(i, j) = e(W_{i..j})$ . The result is presented in Theorem 3.

To discover  $\text{cnt}(i, j)$ , we have the following lemma for discussing the possible shifts of one transposition operation.

**Lemma 1.** Let  $S_1$  and  $S_2$  be two sequences of the same length  $m$ , and  $\sigma^k(S_1) = S_2$ . Then,  $G = \{g \mid \sigma^{k+g}(S_1) = S_2, g \in [0, m-1]\}$  is a group under addition modulo  $m$ .

*Proof.* Since  $\sigma^k(S_1) = S_2$ , we have  $S_1 = \sigma^{-k}(S_2)$ . Then,  $G$  can be written as  $G = \{g \mid \sigma^{k+g}(\sigma^{-k}(S_2)) = S_2, g \in [0, m-1]\}$ . Therefore,  $G = \{g \mid \sigma^g(S_2) = S_2, g \in [0, m-1]\}$ . The following properties show that  $G$  is a group, where the operator  $\oplus$  denotes the addition modulo  $m$ .

- 1) (Closure) For all  $g_1, g_2 \in G$ ,  $\sigma^{g_1 \oplus g_2}(S_2) = \sigma^{g_1}(\sigma^{g_2}(S_2)) = S_2$ , then  $g_1 \oplus g_2 \in G$ .
- 2) (Associativity) For all  $g_1, g_2, g_3 \in G$ ,  $\sigma^{(g_1 \oplus g_2) \oplus g_3}(S_2) = \sigma^{g_1 \oplus (g_2 \oplus g_3)}(S_2) = S_2$ .
- 3) (Identity) For all  $g \in G$ , there is  $0 \in G$  such that  $0 \oplus g = g \oplus 0 = g$ . 0 is the identity.
- 4) (Inverse) For all  $g \in G$ , there is  $m - g \in [0, m-1]$  and  $\sigma^{m-g}(S_2) = S_2$ , therefore  $m - g \in G$ . Since  $g \oplus (m - g) = 0$ ,  $m - g$  is the inverse of  $g$ .

□

**Theorem 2.** (fundamental theorem of cyclic groups [21]) Any subgroup of a cyclic group  $\langle a \rangle$  is cyclic. If  $\langle a \rangle$  is finite of order  $r$ , then the order of every subgroup is a divisor of  $r$ , and for every positive divisor  $q$  of  $r$ , there is only one subgroup  $\langle a^{\frac{r}{q}} \rangle$  of order  $q$  (Each element of  $\langle a^{\frac{r}{q}} \rangle$  is a multiple of  $\frac{r}{q}$ ).

For example, let  $\mathbb{Z}_{12} = \{0, 1, 2, \dots, 11\}$ , which is the group of integers modulo 12. The subgroups of  $\mathbb{Z}_{12}$  are  $\{0, 1, 2, \dots, 11\}$  of order 12,  $\{0, 2, 4, \dots, 10\}$  of order 6,  $\{0, 3, 6, 9\}$  of order 4,  $\{0, 4, 8\}$  of order 3, and  $\{0\}$  of order 1. The order of each subgroup is a divisor of 12, and each subgroup with a specific order is unique.

**Corollary 1.** Let  $S_1$  and  $S_2$  be two sequences of the same length  $m$ ,  $\sigma^k(S_1) = S_2$ , and  $G = \{g \mid \sigma^{k+g}(S_1) = S_2, g \in [0, m-1]\}$ . Then  $G = \{0, \frac{m}{D}, \frac{2m}{D}, \dots, \frac{(D-1)m}{D}\}$  is the subgroup of  $\mathbb{Z}_m$ , where  $D = |G|$  and  $\mathbb{Z}_m$  is the group of integers modulo  $m$ .

Corollary 1 can be derived from Lemma 1 and Theorem 2. For example, suppose that  $S_1$  and  $S_2$  are two sequences of length 12, where  $\sigma^2(S_1) = S_2$ . If there are 3 distinct  $k \in [0, 11]$  satisfying that  $\sigma^k(S_1) = S_2$ , then  $G = \{0, 4, 8\}$  and thus these values of  $k$  are 2, 6, and 10.

**Theorem 3.** Let  $A$  and  $B$  be two sequences of the same length  $n$ , and  $W$  be the paired sequence generated by  $A$  and  $B$ . If  $\text{cnt}(i, j) \geq 2$ , then  $W_{i..j}$  is a repetition of exponent  $\text{cnt}(i, j)$ , that is  $\text{cnt}(i, j) = e(W_{i..j})$ , for  $1 \leq i \leq j \leq n$ .

*Proof.* According to Corollary 1, if  $\text{cnt}(i, j) \geq 2$ , then  $\sigma^{k+g}(A_{i..j}) = \sigma^k(A_{i..j}) = B_{i..j}$ , for  $g \in \{0, \frac{m}{D}, \frac{2m}{D}, \dots, \frac{(D-1)m}{D}\}$ , where  $m = j - i + 1$  and  $D = \text{cnt}(i, j)$ . Thus, both  $A_{i..j}$  and  $B_{i..j}$  are repetitions of exponent  $\text{cnt}(i, j)$ . Consequently,  $W_{i..j}$  is a repetition of exponent  $\text{cnt}(i, j)$ . □

From Theorem 3, the size of  $\text{cnt}(i, j)$  can be represented as follows, where  $W$  is the paired sequence generated by  $A$  and  $B$ .

$$\text{cnt}(i, j) = \begin{cases} 0, & \text{if } TP(i, j) = \infty \\ 1, & \text{if } TP(i, j) = 1 \text{ and } W_{i..j} \\ & \text{is not a repetition} \\ e(W_{i..j}), & \text{if } TP(i, j) = 1 \text{ and } W_{i..j} \\ & \text{is a repetition.} \end{cases} \quad (7)$$

Based on Equation 7, we can further analyze the time complexity of our method more precisely. And we can obtain the following theorem, which is not proved here due to the page limitation. Its proof can be found in the Master's thesis of Hsu [19].

**Theorem 4.** The non-overlapping inversion and transposition distance problem can be solved in  $O(n^2)$  time.

## VI. CONCLUSION

In this paper, we propose an algorithm to solve the non-overlapping inversion and transposition distance problem in quadratic time. It is a great improvement. The key steps of our algorithm are summarized as follows.

- 1) Calculate the minimum cost of each substring (interval) in the preprocessing phase, instead of a linear scan for each substring.

- 2) Utilizing the increment property, make the time complexity of our algorithm depend solely on the number of possible exchanges of two adjacent substrings.
- 3) Prove that the times of a substring being calculated is equal to the exponent of the substring, and show that a substring is counted more than once only if the substring is a repetition.
- 4) Consider only the leftmost substrings to reduce computation, then connect the time complexity analysis of our algorithm with the theorem by Kolpakov and Kucherov [23].

With the above key steps, our algorithm demonstrates the strong relevance of non-overlapping transpositions and the run structure, and then the computation of the auxiliary matrix  $TP$  can be done in quadratic time.

Our algorithm can further retrieve a sequence of operations by adding pointers to trace back in linear time. And any arbitrary scoring scheme can be applied for arbitrary substrings by adjusting the elements in  $\Psi$ .

#### REFERENCES

- [1] A. V. Aho and J. E. Hopcroft, *The Design and Analysis of Computer Algorithms*. Boston, Massachusetts, USA: Addison-Wesley Longman, 1974.
- [2] A. Apostolico and F. P. Preparata, "Optimal off-line detection of repetitions in a string," *Theoretical Computer Science*, Vol. 22, No. 3, pp. 297–315, 1983.
- [3] V. Bafna and P. A. Pevzner, "Genome rearrangements and sorting by reversals," *SIAM Journal on Computing*, Vol. 25, No. 2, pp. 272–289, 1996.
- [4] V. Bafna and P. A. Pevzner, "Sorting by transpositions," Vol. 11, No. 2, pp. 224–240, 1998.
- [5] H. Bannai, S. Inenaga, Y. Nakashima, M. Takeda, K. Tsuruta, *et al.*, "The "runs" theorem," *arXiv preprint arXiv:1406.0263*, 2014.
- [6] J. Berstel, A. Lauve, C. Reutenauer, and F. V. Saliola, *Combinatorics on Words: Christoffel Words and Repetitions in Words*. Providence, Rhode Island, USA: American Mathematical Society, 2009.
- [7] L. Bulteau, G. Fertin, and I. Rusu, "Sorting by transpositions is difficult," Vol. 26, No. 3, pp. 1148–1180, 2012.
- [8] A. Caprara, "Sorting by reversals is difficult," *Proceedings of the first annual international conference on Computational molecular biology*, Santa Fe, New Mexico, USA, pp. 75–83, ACM, 1997.
- [9] D. A. Christie, *Genome rearrangement problems*. PhD thesis, University of Glasgow, 1998.
- [10] M. Crochemore, "An optimal algorithm for computing the repetitions in a word," *Information Processing Letters*, Vol. 12, No. 5, pp. 244–250, 1981.
- [11] M. Crochemore, "Linear searching for a square in a word," *Bulletin-European Association for Theoretical Computer Science*, Vol. 24, No. 1, pp. 66–72, 1984.
- [12] M. Crochemore, L. Ilie, and L. Tinta, "The "runs" conjecture," *Theoretical Computer Science*, Vol. 412, No. 27, pp. 2931–2941, 2011.
- [13] M. Crochemore, C. S. Iliopoulos, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń, "Extracting powers and periods in a word from its runs structure," *Theoretical Computer Science*, Vol. 521, pp. 29–41, 2014.
- [14] M. Crochemore, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń, "On the maximal sum of exponents of runs in a string," *Journal of Discrete Algorithms*, Vol. 14, pp. 29–36, 2012.
- [15] M. Crochemore and W. Rytter, *Text Algorithms*. Singapore: World Scientific, 1994.
- [16] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Viallette, *Combinatorics of Genome Rearrangements*. Cambridge, Massachusetts, USA: MIT Press, 2009.
- [17] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge, England: Cambridge University Press, 1997.
- [18] L. S. Heath and J. P. C. Vergara, "Sorting by bounded block-moves," *Discrete Applied Mathematics*, Vol. 88, No. 1, pp. 181–206, 1998.
- [19] T.-C. Hsu, *An Algorithm for Computing the Distance of the Non-overlapping Inversion and Transposition*. Kaohsiung, Taiwan: Master's Thesis, National Sun Yat-sen University, 2017.
- [20] C. S. Iliopoulos, D. Moore, and W. F. Smyth, "A characterization of the squares in a fibonacci string," *Theoretical Computer Science*, Vol. 172, No. 1, pp. 281–291, 1997.
- [21] N. Jacobson, *Basic Algebra I*. New York, New York, USA: W.H. Freeman, 1985.
- [22] J. Kececioglu and D. Sankoff, "Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement," *Algorithmica*, Vol. 13, No. 1-2, pp. 180–210, 1995.
- [23] R. Kolpakov and G. Kucherov, "Finding maximal repetitions in a word in linear time," *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, New York, New York, USA, pp. 596–604, IEEE Computer Society, 1999.
- [24] C. Ledergerber and C. Dessimoz, "Alignments with non-overlapping moves, inversions and tandem duplications in  $O(n^4)$  time," *Journal of Combinatorial Optimization*, Vol. 16, No. 3, pp. 263–278, 2008.
- [25] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversal," *Soviet Physics Doklady*, Vol. 10, No. 8, pp. 707–710, 1966.
- [26] M. Lothaire, *Combinatorics on Words*. Cambridge, England: Cambridge University Press, 1997.
- [27] M. E. MacDonald, C. M. Ambrose, M. P. Duyao, R. H. Myers, C. Lin, L. Srinidhi, G. Barnes, S. A. Taylor, M. James, N. Groot, *et al.*, "A novel gene containing a trinucleotide repeat that is expanded and unstable on huntington's disease chromosomes," *Cell*, Vol. 72, No. 6, pp. 971–983, 1993.
- [28] M. Maes, "On a cyclic string-to-string correction problem," *Information Processing Letters*, Vol. 35, No. 2, pp. 73–78, 1990.

- [29] M. G. Main, “Detecting leftmost maximal periodicities,” *Discrete Applied Mathematics*, Vol. 25, pp. 145–153, 1989.
- [30] M. G. Main and R. J. Lorentz, “An  $O(n \log n)$  algorithm for finding all repetitions in a string,” *Journal of Algorithms*, Vol. 5, No. 3, pp. 422–432, 1984.
- [31] E. M. McCreight, “A space-economical suffix tree construction algorithm,” *Journal of the ACM*, Vol. 23, No. 2, pp. 262–272, 1976.
- [32] E. W. Myers, “An  $O(ND)$  difference algorithm and its variations,” *Algorithmica*, Vol. 1, No. 1-4, pp. 251–266, 1986.
- [33] M. Schöniger and M. S. Waterman, “A local algorithm for DNA sequence alignment with inversions,” *Bulletin of Mathematical Biology*, Vol. 54, pp. 521–536, 1992.
- [34] D. Shapira and J. A. Storer, “Edit distance with move operations,” *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, Fukuoka, Japan, pp. 85–98, Springer, 2002.
- [35] J. Simpson, “Modified padovan words and the maximum number of runs in a word,” *Australasian Journal of Combinatorics*, Vol. 46, pp. 129–145, 2010.
- [36] T. T. Ta, C. Y. Lin, and C. L. Lu, “An efficient algorithm for computing non-overlapping inversion and transposition distance,” *Information Processing Letters*, Vol. 116, pp. 744–749, 2016.
- [37] A. F. Vellozo, C. E. Alves, and A. P. do Lago, “Alignment with non-overlapping inversions in  $O(n^3)$ -time,” *Proceedings of the International Workshop on Algorithms in Bioinformatics*, Zurich, Switzerland, pp. 186–196, Springer, 2006.
- [38] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” *Journal of the ACM*, Vol. 21, No. 1, pp. 168–173, 1974.
- [39] G. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan, “The chromosome inversion problem,” *Journal of Theoretical Biology*, Vol. 99, No. 1, pp. 1–7, 1982.
- [40] I. Witten, A. Moffat, and T. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Burlington, Massachusetts, USA: Morgan Kaufmann, 1999.