Sorting by Prefix Transpositions

Zanoni Dias Vinicius Fortuna* João Meidanis

August 2, 2004

Abstract

A transposition is an operation that exchanges two consecutive, adjacent blocks in a permutation. A prefix transposition is a transposition that moves the first element in the permutation. In this work we present the pioneering results on the problem of sorting permutations with the minimum number of prefix transpositions. This problem is a variation of the transposition distance problem, related to genome rearrangements. We present approximation algorithms with performance ratios of 2 and 3. We also present an algorithm to sort the reverse permutation $R_n = [n, n-1, \ldots, 3, 2, 1]$ with $n - \lfloor \frac{n}{4} \rfloor$ prefix transpositions. We conjecture that this is the maximum prefix transposition distance among permutations of size n and present the results of some computational tests that support this. Finally, we propose an efficient algorithm that decides whether a given permutation can be sorted using just the number of transpositions indicated by the breakpoint lower bound.

1 Introduction

Sequence comparison is one of the most studied problems in computer science. Usually we are interested in finding the minimum number of local operations, such as insertions, deletions, and substitutions that transform a given sequence into another given sequence. This is the edit distance problem, described in many Computational Biology textbooks [21]. Several studies,

*Supported by FAPESP under grant 03/04578-5

however, have shown that global operations such as reversals and transpositions (also called rearrangement events) are more appropriate when we wish to compare the genomes of two species [20].

A new research area called Genome Rearrangements appeared in the last years to deal with problems such as, for instance, to find the minimum number of rearrangement events needed to transform one genome into another. In the context of Genome Rearrangements, a genome is represented by an n-tuple of genes (or gene clusters). When there are no repeated genes, this n-tuple is a permutation. We proceed with a brief overview of the literature related to the present work.

The best studied rearrangement event is the reversal. A reversal inverts a block of any size in a genome. Caprara [6] proved that finding the minimum number of reversals needed to transform one genome into another is an NP-Hard problem. Bafna and Pevzner [3] have presented an algorithm with approximation factor 2 for this problem. Later Berman, Hannenhalli and Karpinski [5] presented the best known algorithm for the problem, with factor 1.375.

Hannenhalli and Pevzner [13] have studied the reversal distance problem when the orientation of genes is known. In this case they proved that there is a polynomial algorithm for the problem. This algorithm has been refined successively until Tannier and Sagot [22] presented a subquadratic algorithm to sort a signed permutation with reversals. Bader, Moret and Yan [2] presented a linear-time algorithm if you want only the reversal distance between two signed permutations. Meidanis, Walter and Dias [19] have shown that all the reversal theory developed for linear genomes can be easily adapted to circular genomes.

Another interesting variation of this problem is the so-called *prefix reversal problem* or *pancake flipping problem* as it was originally called [9]. In this variation only reversals involving the first consecutive elements of a genome are permitted. Heydari and Sudborough [14] have proved that this problem is NP-Hard. Gates and Papadimitriou [12] and Heydari and Sudborough [15] have studied the diameter of prefix reversals.

The rearrangement event called transposition has the property of exchanging two adjacent blocks of any size in a genome. The transposition distance problem, that is, the problem of finding the minimum number of transpositions necessary to transform one genome into another, has been studied by Bafna and Pevzner [4], who presented some important results, including the best approximation algorithm for the problem at that time, which needs about $3n/4$ moves to sort any permutation, a lower bound of $\lfloor n/2 \rfloor$ for the transposition diameter for the symmetric group S_n, and an upper bound of $\lfloor n/2 \rfloor + 1$ for the transposition distance $d(R_n)$ of the reverse permutation R_n. Computational results proved that $d(R_n)$ equals the upper bound for $3 \leq n \leq 10$. They also conjectured that the transposition diameter for the symmetric group equals the transposition distance of the reverse permutation, which is in fact true for $n < 10$.

Later, Eriksson and colleagues [10] obtained an improved approximation algorithm that sorts any permutation with at most $\lfloor (2n-2)/3 \rfloor$ transpositions. They also give a counterexample to the conjecture for the transposition diameter finding permutations of size 13 and 15 with transposition distance greater than that of the reverse permutation. However they still believe that the conjecture is true for $n > 15$.

The transposition distance problem is still open: we do not know of any NP-Hardness proof, and there are no evidences that an exact polynomial algorithm exists.

Here we present the pioneering results on the variation of the transposition distance problem that we call prefix transposition distance, that is, the rearrangement distance problem where only transpositions affecting two consecutive blocks of the genome, with one of these blocks formed by the first consecutive elements of the genome are allowed. The present work is an improved extension of a previous work by Dias and Meidanis [8], with an improved algorithm and proof by Fortuna and Meidanis [11].

The paper is divided as follows. Initially, in Section 2, we define important concepts that will be used throughout. In Section 3 we present two approximation algorithms for the prefix transposition distance problem, with factors 3 and 2. In Section 4 we present an upper bound on the prefix transposition distance $d_p(R_n)$ of the reverse permutation, several results on the prefix transposition diameter, leading to the conjecture that $D_p(n) = n - \left\lfloor \frac{n}{4} \right\rfloor$, and tests with programs that we implemented to help validate our conjectures. We show in Section 5 an algorithm that verifies whether a given genome can be sorted using the minimum number of prefix transpositions according to the breakpoint lower bound (Lemma 3.8). Finally, in Section 6, we exhibit our conclusions and suggestions for future work.

2 Definitions

Here we introduce a number of basic concepts used in Genome Rearrangements. Notice that some definitions, for instance that of transposition, is different from the definition used in other areas.

Definition 2.1 *An arbitrary genome formed by* n *genes will be represented as a permutation* $\pi = [\pi(1),\ \pi(2),\ \ldots,\ \pi(n)]$ *where each element of* π *represents a gene. The identity genome* ι_n *is defined as* $\iota_n = [1,\ 2,\ \ldots,\ n]$*.*

Definition 2.2 *A transposition* $\tau(x, y, z)$*, where* $1 \leq x < y < z \leq n+1$*, is a rearrangement event that transforms* π *into the genome* $\tau\pi = [\pi(1),\ \ldots,\ \pi(x-1), \pi(y),\ \ldots, \pi(z-1),\ \pi(x),\ \ldots,\ \pi(y-1), \pi(z),\ \ldots,\ \pi(n)]$*.*

Definition 2.3 *A prefix transposition* $\tau(1, x, y)$*, where* $1 < x < y \leq n+1$*, is a rearrangement event that transforms* π *into the genome* $\tau\pi = [\pi(x),\ \ldots,\ \pi(y-1), \pi(1),\ \ldots,\ \pi(x-1), \pi(y),\ \ldots,\ \pi(n)]$*.*

Definition 2.4 *Given two genomes* π *and* σ *we define the transposition distance* $d(\pi, \sigma)$ *between these two genomes as being the least number of transpositions needed to transform* π *into* σ*, that is, the smallest* r *such that there are transpositions* $\tau_1, \tau_2, \ldots \tau_r$ *with* $\tau_r \ldots \tau_2\tau_1\pi = \sigma$*. We call sorting distance by transpositions,* $d(\pi)$*, the transposition distance between the genomes* π *and* ι_n*, that is,* $d(\pi) = d(\pi, \iota_n)$*.*

Definition 2.5 *Given two genomes* π *and* σ *we define the prefix transposition distance* $d_p(\pi, \sigma)$ *between these two genomes as being the least number of prefix transpositions needed to transform* π *into* σ*, that is, the smallest* r *such that there are prefix transpositions* $\tau_1, \tau_2, \ldots \tau_r$ *with* $\tau_r \ldots \tau_2\tau_1\pi = \sigma$*. We call sorting distance by prefix transpositions,* $d_p(\pi)$*, the prefix transposition distance between genomes* π *and* ι_n*, that is,* $d_p(\pi) = d_p(\pi, \iota_n)$*.*

3 Approximation Algorithms

The first important observation is the following.

Lemma 3.1 *For any permutation* π*, we have* $d_p(\pi) \geq d(\pi)$*.*

Proof: This follows from the observation that every prefix transposition is a transposition. The converse is not always true. ∎

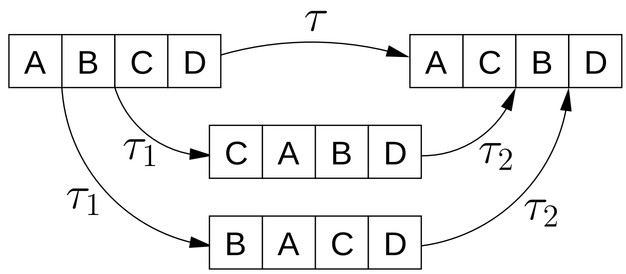

Figure 1: Two examples of how it is possible to obtain prefix transpositions τ_1 and τ_2 such that $\tau\pi = \tau_2\tau_1\pi$, for a given transposition $\tau = \tau(x, y, z)$, with $x \neq 1$.

3.1 Approximation Algorithm with Factor 3

Lemma 3.2 *For every transposition $\tau(x, y, z)$ with $x \neq 1$, there are prefix transpositions $\tau_1(1, r, s)$ and $\tau_2(1, t, u)$ such that $\tau_2\tau_1\pi = \tau\pi$.*

Proof: Indeed, it suffices to take $r = y$, $s = z$, $t = z-y+1$ and $u = z-y+x$, or, alternatively, $r = x$, $s = y$, $t = y - x + 1$ and $u = z$. Figure 1 shows how two prefix transpositions can simulate a transposition. ■

Lemma 3.3 *Any k-approximation algorithm for the transposition distance problem can be transformed into a $2k$-approximation algorithm for the prefix transposition distance problem.*

Proof: Immediate from Lemma 3.2. ■

Therefore it is easy to obtain an approximation algorithm with factor 3 for the prefix transposition distance problem using the approximation algorithm with factor $\frac{3}{2}$ for the transposition distance problem given by Bafna and Pevzner [4] and by Christie [7].

3.2 Approximation Algorithm with Factor 2

We need to define a few important concepts before proceeding. Notice that our definition of breakpoints is slightly different from the usual definitions of breakpoints for the general problem of sorting by transpositions, because the position 1 is always a breakpoint according to our definition.

Definition 3.4 *A breakpoint for the prefix transposition problem is a position i of a permutation π such that $\pi(i) - \pi(i-1) \neq 1$, and $2 \leq i \leq n$. By definition, position 1 (beginning of the permutation) is always considered a breakpoint. Position $n+1$ (end of the permutation) is considered a breakpoint when $\pi(n) \neq n$. We denote by $b_p(\pi)$ the number of breakpoints of permutation π.*

By the former definition $b_p(\pi) \geq 1$ for any permutation π and the only permutations with exactly one breakpoint are the identity permutations (ι_n, for all $n \geq 1$). It is easy to see that there are no permutation with exactly two breakpoints.

Definition 3.5 *A strip is a subsequence $\pi(i..j)$ of π ($i \leq j$) such that i and $j+1$ are breakpoints and there are no breakpoints between these positions.*

Definition 3.6 *Given a permutation π and a prefix transposition τ, we define $\Delta b_p(\pi, \tau)$ as the variation on the number of breakpoints due to operation τ, that is, $\Delta b_p(\pi, \tau) = b_p(\tau\pi) - b_p(\pi)$.*

The first important observation about breakpoints is the following.

Lemma 3.7 *Given a permutation π and a prefix transposition τ, we have that $-2 \leq \Delta b_p(\pi, \tau) \leq 2$.*

Proof: A transposition acts on three different points of a permutation so it can create at most three breakpoints. However a prefix transposition always acts on the first point, which is always a breakpoint by Definition 3.4. Therefore a prefix transposition can create at most two *new* breakpoints. Conversely, if a prefix transposition τ removes more than two breakpoints from a permutation π, its inverse τ^{-1} creates more than two breakpoints in $\tau\pi$, which contradicts the previous statement. That way we have $-2 \leq \Delta b_p(\pi, \tau) \leq 2$, for all permutations π and prefix transpositions τ. ■

Lemma 3.8 *For every permutation π, we have that $d_p(\pi) \geq \left\lceil \frac{b_p(\pi)-1}{2} \right\rceil$.*

Proof: To sort a permutation is equivalent to leave it with only one breakpoint. That means we need to remove $b_p(\pi) - 1$ breakpoints from a permutation π. Since we can remove at most two breakpoints per prefix transposition by Lemma 3.7, we need at least $\left\lceil \frac{b_p(\pi)-1}{2} \right\rceil$ steps to sort a permutation π by prefix transpositions. ■

Lemma 3.9 *Given a permutation $\pi \neq \iota_n$, where $n = |\pi|$, it is always possible to obtain a prefix transposition τ such that $\Delta b_p(\pi, \tau) \leq -1$.*

Proof: Let k be the last element of the first strip of π. If $k < n$, then there is a strip beginning with the element $k+1$, such that $\pi^{-1}(k) < \pi^{-1}(k+1)$ and $\tau = \tau(1, \pi^{-1}(k)+1, \pi^{-1}(k+1))$ suffices. If $k = n$, take $\tau = \tau(1, \pi^{-1}(k)+1, n+1)$. ■

Lemma 3.10 *Let π be a permutation with $b_p(\pi) = 3$, then $d_p(\pi) = 1$.*

Proof: By Lemma 3.9 we can remove at least one breakpoint from π. Removing one breakpoint from π would leave it with exactly two breakpoints, which cannot occur. Therefore, by Lemma 3.7, we can remove two breakpoints from π, leaving it with only one, that is, the identity permutation. ■

Lemma 3.11 *For every permutation π different from the identity we have $d_p(\pi) \leq b_p(\pi) - 2$.*

Proof: If π is not the identity we can leave it with three breakpoints with at most $b_p(\pi)-3$ prefix transpositions by Lemma 3.9. After that, by Lemma 3.10, we need only one more step to sort π. Therefore we can sort π with at most $b_p(\pi) - 3 + 1 = b_p(\pi) - 2$ prefix transpositions. ■

Theorem 3.12 *For every permutation π different from the identity we have $\left\lceil \frac{b_p(\pi)-1}{2} \right\rceil \leq d_p(\pi) \leq b_p(\pi) - 2$.*

Theorem 3.13 *Any algorithm that produces the prefix transpositions according to Lemmas 3.9 and 3.10 is an approximation algorithm with factor 2 for the prefix transposition distance problem.*

Another important point regarding genome rearrangements is the possibility of sorting a permutation without ever increasing the number of breakpoints. Christie [7] has proved that this is true for transposition events. The following lemma establishes the analogous result for prefix transpositions. Since its proof is also analogous to that presented by Christie, it will be omitted here.

Lemma 3.14 *Let π be an arbitrary permutation and $d_p(\pi) = k$ its prefix transposition distance. Then there exists an optimal sequence of prefix transpositions $\tau_1, \ldots, \tau_k$, such that $\tau_k \ldots \tau_1 \pi = \iota_n$, where $n = |\pi|$, and $\Delta b_p(\tau_{i-1} \ldots \tau_1 \pi, \tau_i) \leq 0$ for every $1 \leq i \leq k$.*

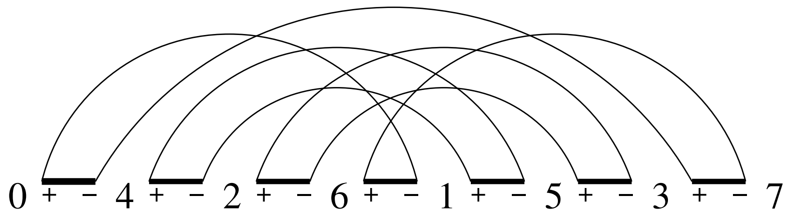

Figure 2: Cycle Diagram for $\pi = [4,\ 2,\ 6,\ 1,\ 5,\ 3]$.

3.3 The Cycle Diagram

Bafna and Pevzner [4] developed the "Cycle Diagram" (originally called "Cycle Graph"), a very useful tool in the study of transposition distance problems. This structure was defined also as the "Reality and Desire Diagram" by Meidanis, Walter and Dias [19], a definition that we reproduce below.

Definition 3.15 *The vertex sequence of the Cycle Diagram is constructed as follows: for every element $\pi(i)$ of permutation π create a pair $-\pi(i)$ and $+\pi(i)$, in this order. Add a vertex $+0$ in the beginning of the sequence and a vertex $-(n+1)$ in the end. The edges of the diagram are of two types: the "desire" (gray) edges and the "reality" (black) edges. The reality edges are drawn joining vertexes $+0$ and $-\pi(1)$, $+\pi(i)$ and $-\pi(i+1)$ (for $1 \leq i \leq (n-1)$), and finally $+\pi(n)$ and $-(n+1)$. The desire edges are drawn joining vertexes $+i$ and $-(i+1)$ (for $0 \leq i \leq n$).*

Once defined the diagram, we now need to define its cycles.

Definition 3.16 *The size of a cycle in the Cycle Diagram is defined as the number of reality edges that compose the cycle. We call $c(\pi)$ the number of cycles in the Cycle Diagram of π. Similarly, $c_{odd}(\pi)$ is the number of odd cycles in the Cycle Diagram of π.*

Figure 2 shows a complete example of a Cycle Diagram. The diagram is composed by two cycles of size 2 and a cycle of size 3. Note that the only permutations of n genes with $n+1$ cycles are the identity permutations ($\pi = \iota_n$, for every $n \geq 1$).

Definition 3.17 *Given an arbitrary permutation π and a transposition τ, we define $\Delta c(\pi, \tau)$ ($\Delta c_{odd}(\pi, \tau)$) as the variation in number of cycles (odd cycles) caused by operation τ, that is, $\Delta c(\pi, \tau) = c(\tau\pi) - c(\pi)$ ($\Delta c_{odd}(\pi, \tau)$ $= c_{odd}(\tau\pi) - c_{odd}(\pi)$).*

The results in the next section were proved by Bafna and Pevzner [4].

3.3.1 Results for transpositions

Lemma 3.18 *Given an arbitrary permutation* π *and a transposition* τ*, we have* $\Delta c_{odd}(\pi,\tau) = \{-2, 0, 2\}$.

Lemma 3.19 *For every permutation* π*, we have* $d(\pi) \geq \frac{n - c_{odd}(\pi)}{2}$.

Lemma 3.20 *Given an arbitrary permutation* $\pi \neq \iota_n$*, where* $n = |\pi|$*, it is always possible to obtain either a transposition* τ *such that* $\Delta c_{odd}(\pi,\tau) = 2$ *or three transpositions* τ_1*,* τ_2 *and* τ_3 *such that* $\Delta c_{odd}(\pi,\tau_1) = 0$*,* $\Delta c_{odd}(\tau_1\pi,\tau_2) = 2$ *and* $\Delta c_{odd}(\tau_2\tau_1\pi,\tau_3) = 2$.

Lemma 3.21 *For every permutation* π*, we have* $d(\pi) \leq \frac{3}{4}(n - c_{odd}(\pi))$.

Proof: Immediate by Lemma 3.20. ■

Theorem 3.22 *For every permutation* π*, we have* $\frac{n - c_{odd}(\pi)}{2} \leq d(\pi) \leq \frac{3}{4}(n - c_{odd}(\pi))$.

Theorem 3.23 *Any algorithm that produces the transpositions indicated by Lemma 3.20 is an algorithm of approximation with factor* $\frac{3}{2}$ *for the transposition distance problem.*

3.3.2 Results for prefix transpositions

In this section we will prove that it is not possible to obtain an approximation algorithm for prefix transpositions with factor better than 2 using the theory developed for general transpositions.

During our study, it was important to characterize some permutations in a more formal way. We will do that by regarding those permutations as sequences and defining a concatenation operator.

Definition 3.24 *The sequence* $a \odot b$ *will be the concatenation of the sequence* a *with the sequence* b*. We also define a generalized concatenation operator such that* $\bigodot_{j=a}^{b} f(j)$ *is the concatenation of the sequences* $f(j)$ *with* j *ranging from* a *to* b.

Example: The following equalities illustrate the use of the operators:

$$[1,2,3] \odot [4,5,6] = [1,2,3,4,5,6]$$

$$\bigodot_{j=0}^{4} [1+3j, 2+3j] = [1,2,4,5,7,8,10,11,13,14]$$

Definition 3.25 *Let M_k be the family of permutations defined as follows: $M_k = \bigodot_{j=0}^{k-1}[1+3j,\ 3+3j,\ 2+3j]$. The permutation M_k is formed by one cycle of size 1 and k cycles of size 3.*

Lemma 3.26 *Consider the permutation M_k, for some $k \geq 1$. In this case we have $d_p(M_k) \geq 2k$ and it is not possible to obtain a prefix transposition τ such that $\Delta c_{odd}(M_k, \tau) = 2$ nor three prefix transpositions τ_1, τ_2 and τ_3 such that $\Delta c_{odd}(M_k, \tau_1) = 0$, $\Delta c_{odd}(\tau_1 M_k, \tau_2) = 2$ and $\Delta c_{odd}(\tau_2 \tau_1 M_k, \tau_3) = 2$.*

Proof: Each one of the cycles of size 3 is eliminated only when a transposition acts on its three reality edges. Since initially no cycle intercepts any other cycle and no 3-cycle can be immediately destroyed, at least two movements are necessary per cycle. Hence $d_p(M_k) \geq 2k$. ■

Lemma 3.27 *Given a permutation M_k, with $k \geq 1$, we have $d_p(M_k) \leq 2k$.*

Proof: Immediate by Algorithm 1. A step-by-step execution of this algorithm on permutation M_2 can be seen in Figure 3. ■

Algorithm 1 Algorithm to sort M_k.

Require: $\pi = M_k$, with $k \geq 1$
Ensure: π sorted
  for $i \leftarrow 1$ **to** k **do**
    $\pi \leftarrow \tau(1, 3(i-1)+2, 3(i-1)+3)\pi$
    $\pi \leftarrow \tau(1, 2, 3(i-1)+4)\pi$

Theorem 3.28 *Given a permutation M_k, for some $k \geq 1$, we have $d_p(M_k) = 2k$.*

Proof: Immediate by Lemmas 3.26 and 3.27. ■

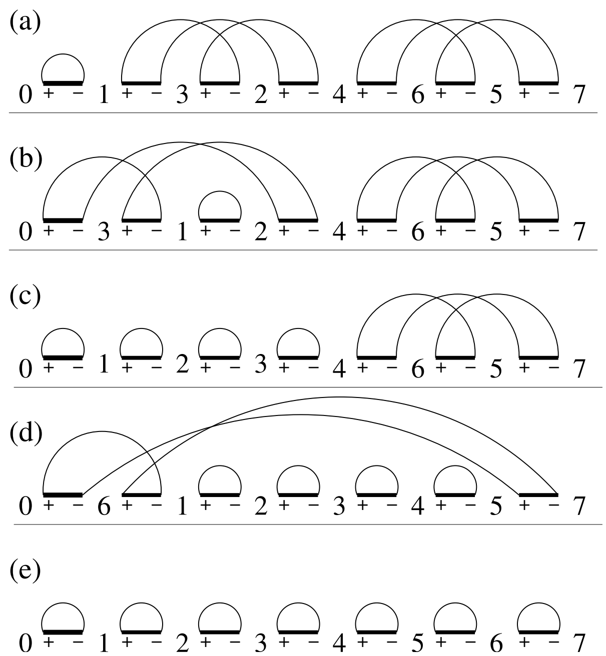

Figure 3: (a) Permutation $M_2 = [1,\ 3,\ 2,\ 4,\ 6,\ 5]$. (b)-(e) Sorting M_2.

Theorem 3.29 *No approximation algorithm for the prefix transposition distance problem based on the Cycle Diagram and using the lower bound of Lemma 3.19 can have an approximation factor less than* 2.

Proof: It suffices to notice that the lower bound of Lemma 3.19 gives $d_p(M_k) \geq k$, while we know that in fact $d_p(M_k) = 2k$ by Theorem 3.28. ■

4 The Diameter of Prefix Transpositions

We call rearrangement diameter the largest rearrangement distance between two permutations of a certain size n. We Denote by $D_p(n)$ the diameter of prefix transpositions and by $D(n)$ the diameter of transpositions. Bafna and Pevzner [4] proved the following result.

Theorem 4.1 *The diameter of transpositions for permutations of size* n *is such that* $\frac{n}{2} \leq D(n) \leq \frac{3n}{4}$.

We can present a similar result for the prefix transposition distance problem.

Theorem 4.2 *The diameter of prefix transpositions for permutations of size n is such that* $\frac{n}{2} \leq D_p(n) \leq n-1$.

Proof: To begin with note that $D_p(n) \geq D(n)$, since $d_p(\pi) \geq d(\pi)$ for any permutation π (Lemma 3.1). We can then use the result of Aigner and West [1] that says that the diameter for the rearrangement distance problem that considers only insertion of the first element, that is, transpositions of the form $\tau(1, 2, x)$, is $n-1$. ■

Definition 4.3 *Let R_n be the family of permutations defined as follows: $R_n = \bigodot_{j=0}^{n-1}[n-j] = [n,\ n-1,\ \ldots,\ 2,\ 1]$. Permutation R_n is called* reverse permutation *of size n.*

Lemma 4.4 *For $n \geq 1$, we have $d_p(R_{n+1}) \geq d_p(R_n)$.*

Proof: It is easy to see that any series of prefix transpositions that sorts R_{n+1} will also sort R_n, provided we adapt the movements that include the element $n+1$. ■

The following result was proved independently by Christie [7] and Meidanis, Walter and Dias [18].

Theorem 4.5 *For $n \geq 3$, we have $d(R_n) = \left\lfloor \frac{n}{2} \right\rfloor + 1$.*

When dealing with prefix transpositions, we could state, based solely on Theorem 3.12, that $\left\lceil \frac{n}{2} \right\rceil \leq d_p(R_n) \leq n-1$. However, a stronger statement holds, as witnessed by Algorithm 2 of page 13.

4.1 Algorithm to sort R_n

Algorithm 2 sorts the reverse permutation R_n with $n - \left\lfloor \frac{n}{4} \right\rfloor$ prefix transpositions. We will prove the correctness this algorithm by splitting it into several phases and proving the transitions between them, working on the characterization of the permutation in each step of the process.

Phase 0 is responsible for reducing the problem of sorting R_n for any n to the problem of sorting R_n for n multiple of 4. What it does is nothing else than to place the $n \bmod 4$ extra elements, one at a time, at their final positions at the end of the permutation. This way they can be left out from this point on and thus we can consider n as being multiple of 4.

Algorithm 2 Algorithm to sort R_n.

Require: $\pi = R_n$, with $n \geq 4$
Ensure: π sorted
  $\triangleright$ Phase 0: Reduces to n multiple of 4
  for $i \leftarrow 0$ **to** $n \bmod 4 - 1$ **do**
    $\pi \leftarrow \tau(1, 2, n+1-i)\pi$
  $n \leftarrow n - (n \bmod 4)$

  $\triangleright$ Phase 1 – At this point $\pi = R_n$, n multiple of 4
  for $i \leftarrow 0$ **to** $n/4 - 2$ **do**
    $\pi \leftarrow \tau(1, 5, n-2i)\pi$
  $\pi \leftarrow \tau(1, 3, n/2+2)\pi$

  $\triangleright$ Phase 2 – At this point $\pi = F_1(n)$
  for $i \leftarrow 0$ **to** $n/4 - 1$ **do**
    $\pi \leftarrow \tau(1, 3, n+1-4i)\pi$

  $\triangleright$ Phase 3 – At this point $\pi = F_2(n)$
  for $i \leftarrow 0$ **to** $\lfloor n/8 \rfloor - 1$ **do**
    $\pi \leftarrow \tau(1, n-4-4i, n-4i)\pi$
  if $n \bmod 8 = 0$ **then**
    $\pi \leftarrow \tau(1, 3, n/2+2)\pi$
  else
    $\pi \leftarrow \tau(1, n/2, n/2+2)\pi$

  $\triangleright$ Phase 4 – At this point $\pi = F_3(n)$
  for $i \leftarrow 0$ **to** $\lceil n/8 \rceil - 2$ **do**
    $\pi \leftarrow \tau(1, 5, 4\lfloor n/8 \rfloor + 6 + 4i)\pi$
  $\triangleright$ At this point $\pi = \iota_n$

Example: Phase 0 does, for $n = 6$, the following transformation:

$$[6,\ 5,\ 4,\ 3,\ 2,\ 1] \rightarrow [4,\ 3,\ 2,\ 1,\ 5,\ 6].$$

After this transformation, the problem reduces to sorting R_n for $n = 4$.

We will define some special families of permutations F_1, F_2 and F_3 that will be proved as the resulting permutations of phases 1, 2, and 3, respectively.

Definition 4.6 *Permutation $F_1(n)$ is given by the following formula:*

$$F_1(n) = [2] \odot \bigodot_{j=0}^{n/4-2} [n-4j,\ n-1-4j] \odot [4,\ 3] \odot \bigodot_{j=0}^{n/4-2} [6+4j,\ 5+4j] \odot [1].$$

Definition 4.7 *Permutation $F_2(n)$ is given by the following formula:*

$$F_2(n) = [3] \odot \bigodot_{j=0}^{n/4-2} [6+4j,\ 7+4j,\ 4+4j,\ 5+4j] \odot [1,\ 2,\ n].$$

Definition 4.8 *Permutation $F_3(n)$ is given by the following formula:*

$$\begin{aligned} F_3(n) \quad = \quad & \bigodot_{j=0}^{\lceil n/8 \rceil -2} \bigodot_{k=0}^{3} [10 - n \bmod 8 + 8j + k] \odot [1,\ 2,\ 3,\ 4,\ 5] \odot \\ & \bigodot_{j=0}^{\lfloor n/8 \rfloor -2} \bigodot_{k=0}^{3} [6 + n \bmod 8 + 8j + k] \odot [n-2,\ n-1,\ n]. \end{aligned}$$

The following lemmas will have their proofs omitted, as they are a bit lengthy and can be found in the publicly available technical report by Fortuna and Meidanis [11].

Lemma 4.9 Phase 1 *changes R_n, $n \geq 4$, into permutation $F_1(n)$.*

Lemma 4.10 Phase 2 *changes $F_1(n)$, $n \geq 4$, into permutation $F_2(n)$.*

Lemma 4.11 Phase 3 *changes $F_2(n)$, $n \geq 8$, into permutation $F_3(n)$.*

Lemma 4.12 Phase 4 *changes $F_3(n)$, $n \geq 8$, into the identity permutation ι_n.*

Theorem 4.13 *For $n \geq 4$, Algorithm 2 sorts R_n with $n - \lfloor n/4 \rfloor$ transpositions.*

Proof: The claim that the algorithm sorts R_n, for $n \geq 8$, is a direct consequence of the application of lemmas 4.9, 4.10, 4.11, 4.12 in sequence. For $n = 4$ it is enough to run the algorithm and verify that it sorts indeed.

Now that we know Algorithm 2 in fact sorts R_n, it remains to count the number of transpositions it takes. Let $n' = n - (n \bmod 4)$. Since n' is multiple of 4, $\left\lfloor \frac{n'}{8} \right\rfloor = \frac{n' - n' \bmod 8}{8}$ and $\left\lceil \frac{n'}{8} \right\rceil = \frac{n' + n' \bmod 8}{8}$. The total number of transpositions is then

$$\begin{aligned} n \bmod 4 + \frac{n'}{4} + \frac{n'}{4} + (\left\lfloor \frac{n'}{8} \right\rfloor + 1) + (\left\lceil \frac{n'}{8} \right\rceil - 1) &= \frac{3n'}{4} + n \bmod 4 \\ &= \frac{4n - (n - n \bmod 4)}{4} \\ &= n - \left\lfloor \frac{n}{4} \right\rfloor, \end{aligned}$$

which concludes our proof. ■

Theorem 4.14 *For $n \geq 1$, $d_p(R_n) \leq n - \left\lfloor \frac{n}{4} \right\rfloor$*

Proof: For $n \geq 4$, this upper bound is guaranteed by Theorem 4.13. For smaller values of n we use Theorem 4.2. ■

Christie [7] and Meidanis, Walter and Dias [18] have proposed the following conjecture.

Conjecture 4.15 *The transposition diameter $D(n)$, for $n \geq 3$, is given by $D(n) = d(R_n) = \left\lfloor \frac{n}{2} \right\rfloor + 1$.*

For $n \leq 12$, Eriksson and colleagues [10] proved that this conjecture is true. For $n = 13$ and $n = 15$ they found counterexamples that were more difficult to sort than the reverse permutation, proving $D(13) = D(14) = 8$ and $D(15) = 9$. However they still believe that the conjecture is true for $n > 15$, since their computer experiments suggested that the patterns that are especially difficult to sort for $n = 13$ and $n = 15$ cease to be difficult for larger n.

Like the general problem of sorting by transpositions, we also believe that the following statement is true for the analogous problem of sorting by prefix transpositions.

Conjecture 4.16 *The diameter of prefix transpositions $D_p(n)$, for $n \geq 4$, is given by $D_p(n) = d_p(R_n) = n - \left\lfloor \frac{n}{4} \right\rfloor$.*

4.2 Tests

The tests that will be presented in this section were performed in a Digital Alpha Server GS140 computer, with 10 Alpha 21264 EV6 processors of 524MHz and 64-bit word length, with 8 GB of physical memory and running the OSF1 version 4.0 operating system. All programs were written in C++ and compiled with g++ using compilation directive "-O3". Our programs use just one processor and during the tests the machine was always executing other processes as well. The measured times are the times effectively spent by the programs (user + system time) and not the total time of execution (real time).

We implemented two "branch and bound" algorithms to compute the exact distance of prefix transpositions. The first version considers all possible prefix transpositions, while the second version considers only prefix transpositions that do not create new breakpoints, according to Lemma 3.14. Using these programs it was possible to obtain directly the prefix transposition distance for all reverse permutations R_n with $n \leq 15$ and indirectly for R_{16}. Table 1 and Figure 4 show result summaries.

Lastly we implemented two programs to verify the conjectures proposed in Section 4. The two programs are based in the same strategy. We built a graph as follows: we created a vertex for each of the $n!$ permutations with n elements and an edge for each pair of permutations that differ by a rearrangement event. In this graph we search for the permutations that posses the largest distance from the identity permutation. This strategy can be implemented in linear time on the graph size. With this method we could certify in slightly over 20 hours that both conjectures are true for permutations with $n \leq 11$ elements. Unfortunately 30 GB of physical memory are need to build the graph for $n = 12$, what made the test of our conjectures for $n \geq 12$ impossible.

5 Permutations that Satisfy the Breakpoint Lower-Bound

Kececioglu and Sankoff [17] conjectured that to determine whether a permutation can be sorted using the minimum number of reversals indicated by the breakpoint lower bound for reversals was an NP-Hard problem, just like the general problem of sorting by reversals. Irving and Christie [16] and Tran [23]

n	$d_p(R_n)$	Time without optimization (seconds)	Time with optimization (seconds)
02	01	0	0
03	02	0	0
04	03	0	0
05	04	0	0
06	05	0	0
07	06	0	0
08	06	4	2
09	07	9	3
10	08	59	22
11	09	1011	373
12	09	8872	2607
13	10	16294	4305
14	11	118463	45168
15	12	2771374	1081631
16	12*	750 days *	300 days *

Table 1: distance of prefix transposition for reverse permutations with 16 or less elements. The times in column "without optimization" refer to the "branch and bound" algorithm that considers all prefix transpositions possible, while the column "with optimization" presents the results of the implementation that considers only prefix transpositions that do not create new breakpoints, according to Lemma 3.14. We could not compute $d_p(R_{16})$ directly using any of the two implementations; instead we present an estimate of the time necessary for each algorithm to compute correctly the distance. Note also that it is possible to infer the distance $d_p(R_{16})$ from Theorem 4.14 and Lemma 4.4.

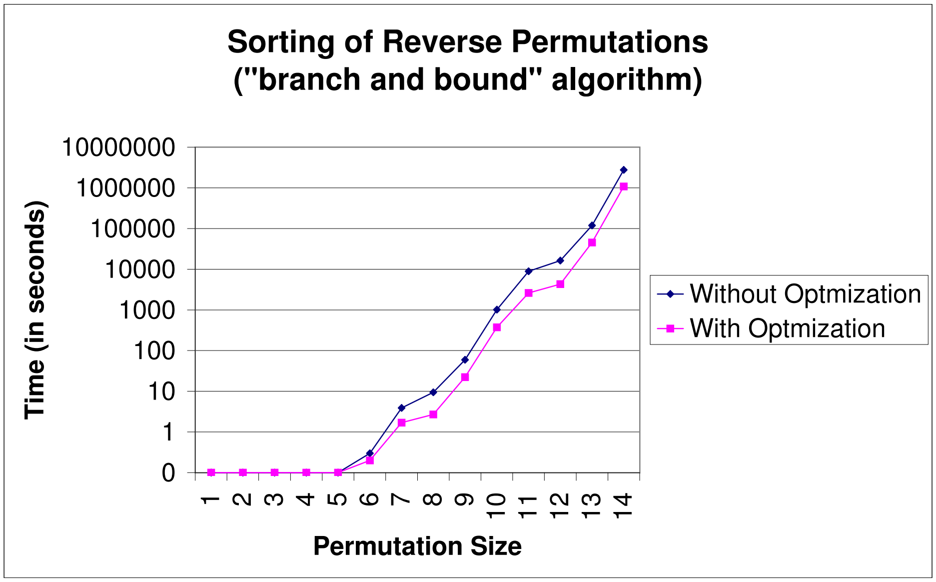

Figure 4: Results for the "branch and bound" algorithm. Total approximate time of 47 days nonstop processing, with about 34 days for the version not optimized and 13 days for the optimized version.

independently proved that this conjecture is false, exhibiting a polynomial algorithm for the problem.

In the case of prefix transpositions we know from Lemma 3.8 that for every permutation π we have $d_p(\pi) \geq \lceil (b_p(\pi) - 1)/2 \rceil$. However, given a permutation π, is it possible to determine whether $d_p(\pi) = (b_p(\pi) - 1)/2$? The following results prove that the answer is yes.

Lemma 5.1 *Let π be an arbitrary permutation. Then there exists at most one prefix transposition τ such that $\Delta b_p(\pi, \tau) = -2$.*

Proof: Suppose that π and $\tau(1, x, y)$ are such that $\Delta b_p(\pi, \tau) = -2$. In this case we have $\pi = [\pi(1), \ldots, \pi(x-1), \pi(x), \ldots, \pi(y-1), \pi(y), \ldots]$ and $\tau\pi = [\pi(x), \ldots, \pi(y-1), \pi(1), \ldots, \pi(x-1), \pi(y), \ldots]$, where $\pi(x-1) \neq \pi(x) - 1$, $\pi(y-1) \neq \pi(y) - 1$, $\pi(y-1) = \pi(1) - 1$ and $\pi(x-1) = \pi(y) - 1$. Finally, note that $\pi(1)$ determines uniquely the index y, and y determines uniquely the index x. ■

Theorem 5.2 *Let π be an arbitrary permutation. Then it is possible to determine in polynomial time whether $d_p(\pi) = \frac{b_p(\pi)-1}{2}$.*

Proof: Immediate by Algorithm 3, that has complexity $O(n^2)$. ■

Given an integer k, is it always possible to find a permutation π such that there is a series of k prefix transpositions $\tau_1, \ldots, \tau_k$ with $\Delta b_p(\tau_{i-1}\tau_{i-2}\ldots\tau_1\pi, \tau_i) = -2$, for $1 \leq i \leq k$? Once again the answer is affirmative.

Algorithm 3 Verifying whether π has distance $d_p(\pi) = \frac{b_p(\pi)-1}{2}$.

```
n ← |π|
while π ≠ ι_n do
    y ← π^{-1}(π(1) − 1) + 1
    x ← π^{-1}(π(y) − 1) + 1
    ▷ Verifies whether there exists a movement that removes two break-
        points
    if x < y then
        π ← τ(1, x, y)π
    else
        return FALSE
return TRUE
```

Definition 5.3 *Let B_k be the family of permutations defined as follows: $B_k = \bigodot_{i=0}^{k-1}[k+1+i,\ k-i]$. Permutation B_k possesses $2k+1$ breakpoints.*

Lemma 5.4 *For every integer k it is possible to obtain a series of k prefix transpositions $\tau_1, \tau_2, \ldots, \tau_k$ that sort B_k such that $\Delta b_p(\tau_{i-1}\tau_{i-2}\ldots\tau_1\pi, \tau_i) = -2$, for $1 \leq i \leq k$.*

Proof: Immediate from Algorithm 4, that can be implemented in linear time. ■

Algorithm 4 The algorithm that sorts B_k.

Require: $\pi = B_k$, with $k \geq 1$

Ensure: π sorted

```
for i ← 1 to k do
    π ← τ(1, 2i, 2i + 1)π
```

---

6 Conclusions

We introduced in this work a new problem of Genome Rearrangement that we called distance of prefix transpositions. We showed a number of results for this problem, including two approximation algorithms (the best of them with factor 2), a proof that any permutation can be sorted without "cutting strips", a conjecture on the prefix transposition diameter stating that $D_p(n) = n - \left\lfloor \frac{n}{4} \right\rfloor$, and an algorithm for determining whether a permutation can be sorted using a series of prefix transpositions removing two breakpoints per step. The study of such permutations may give us a new insight about the problem, which remains open.

References

[1] M. Aigner and D. B. West. Sorting by insertion of leading element. *Journal of Combinatorial Theory*, 45:306–309, 1987.

[2] D. A. Bader, B. M. E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. In *Proceedings of the Seventh Workshop on Algorithms and Data Structures (WADS'01)*. Springer Verlag, 2001.

[3] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[4] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, May 1998.

[5] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th European Symposium on Algorithms (ESA'2002)*, Lecture Notes in Computer Science, Rome, Italy, September 2002. Springer.

[6] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology - (RECOMB'97)*, pages 75–83, New York, USA, January 1997. ACM Press.

[7] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, Glasgow University, 1998.

[8] Z. Dias and J. Meidanis. Sorting by prefix transpositions. In A. H. F. Laender and A. L. Oliveira, editors, *Proceedings of the String Processing and Information Retrieval (SPIRE'2002)*, number 2476 in Lecture Notes in Computer Science, pages 65–76, Lisboa, Portugal, September 2002. Springer-Verlag, Berlin.

[9] H. Dweighter. *American Mathematical Monthly*, volume 82, page 1010. The Mathematical Association of America, 1975.

[10] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wästlund. Sorting a bridge hand. *Discrete Mathematics*, 241:289–300, 2001.

[11] V. J. Fortuna and J. Meidanis. Sorting the reverse permutation by prefix transpositions. Technical Report IC-04-04, Institute of Computing - University of Campinas, April 2004.

[12] W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979.

[13] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, January 1999.

[14] M. H. Heydari and I. H. Sudborough. Sorting by prefix reversals is NP-complete. To be submitted.

[15] M. H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25:67–94, 1997.

[16] R. W. Irving and D. A. Christie. Sorting by reversals: on a conjecture of Kececioglu and Sankoff. Technical Report TR-95-12, Department of Computing Science, University of Glasgow, May 1995.

[17] J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13:180–210, January 1995.

[18] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Transposition distance between a permutation and its reverse. In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing (WSP'97)*, pages 70–79, Valparaiso, Chile, 1997. Carleton University Press.

[19] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. Technical Report IC-00-23, Institute of Computing - University of Campinas, December 2000.

[20] J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 27:87–97, 1988.

[21] J. C. Setubal and J. Meidanis. *Introduction to Computional Molecular Biology*. PWS Publishing Company, 1997.

[22] E. Tannier and M. Sagot. Sorting by reversals in subquadratic time. Technical Report 5097, INRIA, Institut National de Recherche en Informatique et en Automatique, January 2004.

[23] N. Q. Tran. An easy case of sorting by reversals. *Journal of Computational Biology*, 5(4):741–746, 1998.