Discrete Applied Mathematics 181 (2015) 78–89

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Sorting by Prefix Reversals and Prefix Transpositions

Zanoni Dias *, Ulisses Dias

Institute of Computing, University of Campinas, Campinas - SP, Brazil

ARTICLE INFO

Article history:
Received 4 October 2013
Received in revised form 28 August 2014
Accepted 2 September 2014
Available online 26 September 2014

Keywords:
Approximation algorithms
Genome rearrangement
Prefix reversals
Prefix transpositions

ABSTRACT

In this paper, we present a new algorithm for the Sorting by Prefix Reversals and Prefix Transpositions Problem. The previous approximation algorithm was bounded by factor 3, and here we present an asymptotic 2-approximation algorithm. We consider theoretical and practical aspects in our analysis, and we show that our method is better than other approaches in both cases.

1. Introduction

In the literature on genome rearrangement that focuses on mathematical models of genomes, chromosomes are usually represented as sequences of segments called syntenic blocks, which we assume to be shared by the genomes being compared. Let n be the total number of shared segments, we assign a unique number in the set $\{1, \ldots, n\}$ to each segment such that chromosomes can be regarded as permutations. We sort a given permutation by applying successive operations that transform it into another permutation where all elements are in ascending order. The main goal of sorting problems is to find the minimum number of such operations, which is called distance.

Reversals and transpositions are two operations that affect real genomes. They lead to the challenging classic problems in genome rearrangement field called Sorting by Reversals Problem and Sorting by Transpositions Problem.

Reversal

Reversals occur when a block of elements in the permutation is reversed. Caprara proved that the Sorting by Reversals Problem is NP-Hard [6]. Kececioglu and Sankoff [19] presented the first approximation algorithm with approximation factor 2. The factor was later improved to 1.75 by Bafna and Pevzner [1] and to 1.5 by Christie [9]. The best algorithm to date is the 1.375-approximation algorithm deviated by Berman, Hannenhalli and Karpinski [3].

Trans position

Transpositions occur when two adjacent blocks of elements exchange position. Bulteau, Fertin and Rusu proved that the Sorting by Transpositions Problem is NP-Hard [5]. Bafna and Pevzner [2] presented the first approximation algorithm with approximation factor 1.5. The factor was later improved to 1.375 by Elias and Hartman [14], which is the best approximation factor so far. From a practical viewpoint, Dias and Dias [10] and Dias et al. [11] presented heuristics that lead to the best results to date.

When reversals and transpositions act on blocks located in the beginning of the permutation, they are called prefix reversals and prefix transpositions, respectively. Since the Sorting by Reversals and the Sorting by Transpositions problems are very challenging, the study of variants like prefix transpositions and prefix reversals has been tried in order to shed light on the original problem [15].

* Corresponding author. Tel.: +55 1935215861; fax: +55 1935215847.
E-mail addresses: zanoni@ic.unicamp.br (Z. Dias), udias@ic.unicamp.br (U. Dias).
URL: http://www.ic.unicamp/~zanoni (Z. Dias).

http://dx.doi.org/10.1016/j.dam.2014.09.004
0166-218X/

Prefix operations also relate to interconnection network problems where processors are labeled as permutations of a given size n, and a communication link between two processors exists if a prefix reversal can transform one label into the other. The maximum communication delay between a pair of processors in the network is measured by the diameter of the graph, which is the greatest distance among all permutations of size n [21].

The so-called Pancake Flipping Problem proposed by Dweighter (pseudonym of J.E. Goodman) [13] is the sorting problem where the only allowed operations are prefix reversals. Therefore, the Pancake Flipping Problem is hereafter referred to as the Sorting by Prefix Reversals Problem. In 1979, Gates and Papadimitriou [17] proved that $\frac{5n+5}{3}$ movements are sufficient and $\frac{17n}{16}$ movements may be necessary to sort any permutation of n elements. In 1997, Heydari and Sudborough [18] showed that $\frac{15n}{14}$ movements may be required to sort a permutation of size n using only prefix reversals. In 2009, Chitturi et al. [7] proved that $\frac{18n}{11}$ movements are sufficient to sort any permutation of n elements. The best algorithm to date was presented in 2005 by Fischer and Ginzinger [16] with approximation factor 2. Recently, Bulteau, Fertin and Rusu [4] proved that this is an NP-complete problem.

The problem of sorting a permutation by prefix transpositions was posed by Dias and Meidanis [12]. They presented a 2-approximation algorithm, and provided lower and upper bounds of $\frac{n}{2}$ and $n-1$, respectively, for the number of prefix transposition that may be necessary to sort any permutation of size n. Chitturi and Sudborough [8] improved the upper bound to $n - \log_8 n$, and Labarre [20] improved the lower bound to $\lfloor \frac{3n}{4} \rfloor$.

Recently, Sharmin et al. [22] proposed the sorting problem where the only allowed operations are prefix reversals and prefix transpositions. They presented a 3-approximation algorithm to this problem. In this paper, we present an asymptotic 2-approximation algorithm to the same problem.

2. Basic definitions

Throughout this paper, we represent a genome with n conserved blocks as a permutation $\pi = (\pi_1\ \pi_2\ \dots\ \pi_n)$, $\pi_i \in \mathbb{N}$, $1 \le \pi_i \le n$, and $\pi_i \neq \pi_j$ for all $i \neq j$. Here, we consider a permutation as a bijective function in the set $\{1, 2, \dots, n\}$ such that $\pi(i) = \pi_i$.

Definition. The *composition* of two permutations π and σ is the permutation $\pi \cdot \sigma = (\pi_{\sigma(1)}\ \pi_{\sigma(2)}\ \dots\ \pi_{\sigma(n)})$.

We can see the composition as the relabeling of elements in π according to the elements in σ. Let ι be the identity permutation $\iota(i) = i$, we can easily verify that ι is a neutral element such that $\pi \cdot \iota = \iota \cdot \pi = \pi$.

Definition. We define the *inverse* of a permutation π as the permutation π^{-1} that returns $\pi \cdot \pi^{-1} = \pi^{-1} \cdot \pi = \iota$.

The inverse permutation is the function that returns $\pi^{-1}_{\pi(i)} = i$. In other words, it returns the position in π of each element π_i. index <-> value

Definition. A *reversal* $\rho_r(i, j)$ is the permutation $(1\ \dots\ i-1\ j\ j-1\ \dots\ i+1\ i\ j+1\ \dots\ n)$, $1 \le i < j \le n$. Applying a reversal to a permutation π reverses the order of $\pi[i..j]$, which is the same as the composition $\pi \cdot \rho_r(i, j) = (\pi_1\ \pi_2\ \dots\ \pi_{i-1}\ \pi_j\ \pi_{j-1}\ \dots\ \pi_{i+1}\ \pi_i\ \pi_{j+1}\ \dots\ \pi_{n-1}\ \pi_n)$.

Definition. A *prefix reversal* $\rho_{pr}(k)$ is a reversal $\rho_r(1, k)$ that reverses k elements in the beginning of the permutation.

Definition. A prefix reversal $\rho_{pr}(k)$ *acts* on element π_x if $x \le k$.

B A

Definition. A *transposition* $\rho_t(i, j, k)$ is the permutation $(1\ 2\ \dots\ i-1\ j\ j+1\ \dots\ k-1\ i\ i+1\ \dots\ j-1\ k\ \dots\ n)$, $1 \le i < j < k \le n+1$. Applying a transposition to a permutation π swaps the adjacent blocks $\pi[i..j-1]$ and $\pi[j..k-1]$, which is the same as the composition $\pi \cdot \rho_t(i, j, k) = (\pi_1\ \pi_2\ \dots\ \pi_{i-1}\ \pi_j\ \pi_{j+1}\ \dots\ \pi_{k-1}\ \pi_i\ \pi_{i+1}\ \dots\ \pi_{j-1}\ \pi_k\ \dots\ \pi_n)$.

A、B互為鄰居且相互交換

Definition. A *prefix transposition* $\rho_{pt}(j, k)$ is a transposition $\rho_t(1, j, k)$ that moves a block in the beginning of the permutation.

Definition. A prefix transposition $\rho_{pt}(j, k)$ *acts* on element π_x if $x \le k-1$. 前後多插入一個元素，為了要建立Breakpoint

Consider the extended permutation that can be obtained from π by inserting two new elements: $\pi_0 = 0$ and $\pi_{n+1} = n+1$. The extended permutation is still denoted as π. Below, we present two definitions of breakpoint for a permutation π.

Definition. A pair of elements π_i and π_{i+1}, with $1 \le i \le n$, is a *prefix reversal breakpoint* if $|\pi_{i+1} - \pi_i| \neq 1$. The number of prefix reversal breakpoints in a permutation π is denoted by $b_{pr}(\pi)$. b() is the number of breakpoint

Definition. Prefix reversal breakpoints divide a permutation into *strips*, which are maximal intervals with no prefix reversal breakpoints. In addition, the elements $\pi_0 = 0$ and $\pi_{n+1} = n+1$ do not belong to any strip.

strip為沒有breakpoint對最大區間

Definition. A pair of elements π_i and π_{i+1}, with $1 \leq i \leq n$, is a *prefix transposition breakpoint* if $\pi_{i+1} - \pi_i \neq 1$. The number of prefix transposition breakpoints in a permutation π is denoted by $b_{pt}(\pi)$.

對方向敏感

Observe that by definition the pair $[\pi_0, \pi_1]$ is not a breakpoint.

The only permutation with no breakpoint (neither prefix reversal breakpoint nor prefix transposition breakpoint) is the identity permutation $\iota = (1\ 2\ \ldots\ n)$, such that $b_{pr}(\iota) = b_{pt}(\iota) = 0$. 沒有breakpoint的排序即排序完成

In this paper, we mention several distance problems related to prefix operations. Those problems are defined as follows.

/xi/

Definition. Let ξ be a set of rearrangement events that can be applied to π, the *distance* $d_\xi(\pi)$ is the minimum number t of operations $\xi_1, \xi_2, \ldots, \xi_t \in \xi$ such that $\pi \cdot \xi_1 \cdot \xi_2 \cdot \ldots \cdot \xi_t = \iota$. ξ為可以在排序中操作的事件

Another important definition relates to the number of operations that may be necessary to sort any permutation of size n.

Sn為n個元素的所有排列可能

Definition. Let S_n be the group of all permutations that have the same size n, and ξ be the set of rearrangement events that can be applied to permutations in S_n. The greatest distance for all permutations in S_n using events in ξ is said to be the *diameter* of S_n in regard to ξ, and we denote it by $D_\xi(n) = \max\{d_\xi(\pi) | \pi \in S_n\}$所有Sn中的排列使用ξ中事件的最大距離（事件數）

Hereafter, we shall use *pr*, *pt* and *prpt* to represent rearrangement models such that we allow only prefix reversals, prefix transpositions and both, respectively.

Our main result is an asymptotic 2-approximation algorithm to the Sorting by Prefix Reversals and Prefix Transpositions Problem. In short, our algorithm relies on four scenarios that remove prefix reversal breakpoints and on one step that is used when the first element in the permutation is $\pi_1 = 1$. This step does not remove prefix reversal breakpoints, but we guarantee that it will be used at most twice during the execution of our algorithm.

Next sections are organized as follows. Section 3 further describes the literature on the Sorting by Prefix Reversals and Prefix Transpositions Problem. Section 4 introduces our algorithm and gives a formal proof for the approximation factor. Section 5 presents a comparative analysis of the algorithms that provide valid solutions for the problem we are dealing with. Section 6 concludes this work.

3. Previous algorithms

In this section, we discuss the theoretical approximation aspects of three previous algorithms that provide valid sequences to the Sorting by Prefix Reversals and Prefix Transpositions Problem.

Definition. We denote as $\Delta_{b_{pr}}(\pi, \rho_\xi) = b_{pr}(\pi \cdot \rho_\xi) - b_{pr}(\pi)$ the change in number of prefix reversal breakpoints due to operation ρ_ξ, $\xi \in \{pr, pt, prpt\}$.

後 – 前

Definition. We denote as $\Delta_{b_{pt}}(\pi, \rho_\xi) = b_{pt}(\pi \cdot \rho_\xi) - b_{pt}(\pi)$ the change in number of prefix transposition breakpoints due to operation $\rho_\xi, \xi \in \{pr, pt, prpt\}$.

Lemma 1. $\Delta_{b_{pr}}(\pi, \rho_{pr}) \in \{-1, 0, 1\}$ [16]. ρpr只會新增或刪除一個pr

Proof. The prefix reversal $\rho_{pr}(k)$ splits the pairs $[\pi_0, \pi_1]$ and $[\pi_k, \pi_{k+1}]$. Since there is no prefix reversal breakpoint at indices 0 and 1 (by definition), only at indices k, $k+1$ the prefix reversal can create or remove one prefix reversal breakpoint. □

Lemma 2. $b_{pr}(\pi) \leq d_{pr}(\pi) \leq 2b_{pr}(\pi)$ [16].

Proof. It is straightforward from Lemma 1 that $d_{pr}(\pi) \geq b_{pr}(\pi)$ [16]. In addition, Fischer and Ginzinger [16] presented a 2-approximation algorithm for the Sorting by Prefix Reversals Problem that finds a sequence that sorts π with at most $2b_{pr}(\pi)$ prefix reversals. □

Lemma 3. $\Delta_{b_{pt}}(\pi, \rho_{pt}), \Delta_{b_{pr}}(\pi, \rho_{pt}) \in \{-2, -1, 0, 1, 2\}$ [12,22].

Proof. The prefix transposition $\rho_{pt}(j, k)$ splits the pairs $[\pi_0, \pi_1]$, $[\pi_{j-1}, \pi_j]$ and $[\pi_{k-1}, \pi_k]$. Since there is no breakpoint at indices 0 and 1 (by definition), only at indices $j-1, j$ and $k-1, k$ the prefix transposition can create or remove either prefix reversal breakpoints or prefix transposition breakpoints. □

Lemma 4. $\frac{b_{pt}(\pi)}{2} \leq d_{pt}(\pi) \leq b_{pt}(\pi) - 1$ [12].

Proof. It is straightforward from Lemma 3 that $d_{pt}(\pi) \geq \frac{b_{pt}(\pi)}{2}$. In addition, Dias and Meidanis [12] presented a 2-approximation algorithm for the Sorting by Prefix Transpositions Problem that finds a sequence that sorts π with at most $b_{pt}(\pi) - 1$ prefix transpositions. □

Lemma 5. $d_{prpt}(\pi) \geq \frac{b_{pr}(\pi)}{2}$ [22].

Proof. Straightforward from Lemmas 1 and 3. □

Lemma 6. $\Delta_{b_{pt}}(\pi, \rho_{pr}) \in \{-n, -(n-1), \ldots, -1, 0, 1, \ldots, n-1, n\}$. pr能夠增加或減少pt的幅度較大，因為pt對方向敏感

Proof. The prefix reversal $\rho_{pr}(k)$ splits the pairs $[\pi_k, \pi_{k+1}]$, which implies that a prefix transposition breakpoint can be created or removed at that position. In addition, prefix transposition breakpoints can be created or removed anywhere in the range $\pi[1..k]$. Let us assume that $\pi_{a+1} = \pi_a + 1$ for some $a < k$, we can see that $[\pi_a, \pi_{a+1}]$ is not a prefix transposition breakpoint. After applying $\rho_{pr}(k)$, the prefix transposition breakpoint $[\pi_{a+1}, \pi_a]$ will be created. Analogously, one can remove a prefix transpositions breakpoint if the pair $[\pi_{a'+1}, \pi_{a'}]$ such that $\pi_{a'+1} = \pi_{a'} + 1$ exists for some $a' < k$. □

Lemma 7. $d_{prpt}(\pi) \geq \frac{b_{pt}(\pi)}{n}$.

Proof. Straightforward from Lemmas 3 and 6. □

We cited previously an algorithm for the Sorting by Prefix Reversals Problem presented by Fischer and Ginzinger [16], and an algorithm for the Sorting by Prefix Transpositions Problem presented by Dias and Meidanis [12]. Since the problem we are dealing with in this paper accepts both prefix reversals and prefix transpositions, those algorithms provide valid sequences to sort any input permutation. The next lemmas show us what we can expect about the approximation guarantee provided by both algorithms.

Lemma 8. *The algorithm presented by Fischer and Ginzinger [16] for the Sorting by Prefix Reversals Problem is a 4-approximation algorithm for the Sorting by Prefix Reversals and Prefix Transpositions Problem.*

Proof. The algorithm presented by Fischer and Ginzinger finds a sequence that sorts π with at most $2b_{pr}(\pi)$ prefix reversals. Because we know from Lemma 5 that $d_{prpt}(\pi) \geq \frac{b_{pr}(\pi)}{2}$, the stated approximation guarantee follows. □

Lemma 9. *The algorithm presented by Dias and Meidanis [12] for the Sorting by Prefix Transpositions Problem is an unbounded algorithm for the Sorting by Prefix Reversals and Prefix Transpositions Problem.*

Proof. The algorithm presented by Dias and Meidanis finds a sequence that sorts π with at most $b_{pt}(\pi) - 1$ prefix transpositions. We know from Lemma 7 that $d_{prpt}(\pi) \geq \frac{b_{pt}(\pi)}{n}$, so the best we could do is to guarantee an $O(n)$ approximation ratio. □

Recently, Sharmin et al. [22] presented a 3-approximation algorithm to the problem of sorting permutations by prefix reversals and prefix transpositions. They showed that it is possible to remove 2 prefix reversal breakpoints in three operations. Therefore, they use at most $\frac{3b_{pr}(\pi)}{2}$ operations to sort an arbitrary permutation π. Because we know from Lemma 5 that $d_{prpt}(\pi) \geq \frac{b_{pr}(\pi)}{2}$, the stated approximation guarantee follows.

In Section 4, we present a new algorithm with a lower approximation ratio. Our algorithm has an asymptotic approximation factor 2. The three algorithms cited in this section were used in a practical analysis against our algorithm in Section 5.

4. Algorithm for sorting by prefix reversals and prefix transpositions

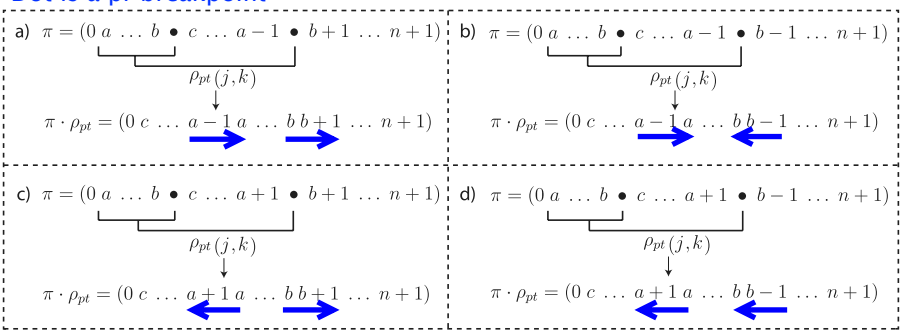
First, we will assume $\pi_1 \neq 1$ and explain in Section 4.1 that our algorithm always finds a movement that removes at least one breakpoint if that condition is satisfied. In Section 4.2, we deal with $\pi_1 = 1$, we do not remove any prefix reversal breakpoint in this case, but we guarantee that it occurs at most twice during the execution of our algorithm. That will lead to the guaranteed asymptotic approximation factor.

4.1. Removing prefix reversal breakpoints $\pi 1 \neq 1$

This section deals with permutations π such that $\pi_1 \neq 1$. Our algorithm firstly tries to remove two prefix reversal breakpoints, which is the best we can do according to Lemma 3 by using prefix transpositions. Observe that this goal cannot be achieved by prefix reversals according to Lemma 1. That said, the prefix transposition that removes 2 breakpoints should cause the following effects. 使用ρpt移除pr breakpoint效果最好

1. Let $\pi_1 = a$, the operation $\rho_{pt}(j, k)$ should place a just after one of the elements $a-1$ or $a+1$ in $\pi \cdot \rho_{pt}(j, k)$. In this case, the element $a-1$ or $a+1$ must be at position $k-1$, we also need to assure that the pair $[\pi_{k-1}, \pi_k]$ is a prefix reversal breakpoint in order to be able to split it using $\rho_{pt}(j, k)$.
2. Let $\pi_{j-1} = b$, the operation $\rho_{pt}(j, k)$ should place b just before one of the elements $b-1$ or $b+1$ in $\pi \cdot \rho_{pt}(j, k)$. In this case, the element $b-1$ or $b+1$ must be at position k, we also need to assure that the pair $[\pi_{j-1}, \pi_j]$ is a prefix reversal breakpoint in order to be able to split it using $\rho_{pt}(j, k)$.

Dot is a pr breakpoint



4種可能 **Fig. 1.** Scenarios where we can remove two prefix reversal breakpoints. In addition, Algorithm 1 forces $c \neq 1$.

ascending strip

descending strip

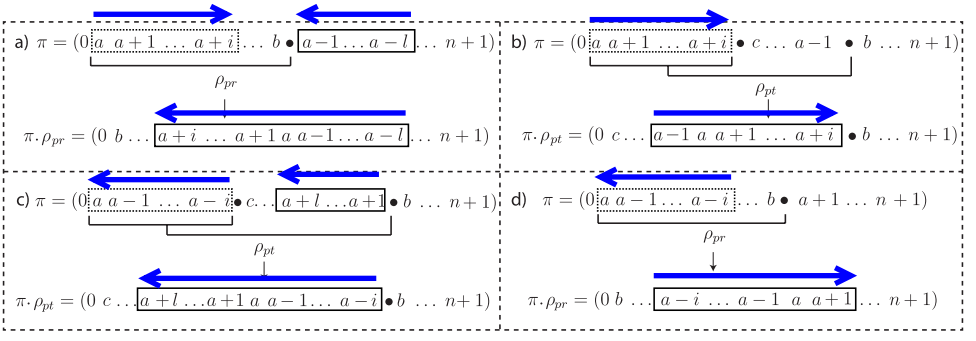


Fig. 2. Remove one prefix reversal breakpoint. Dotted boxes represent strips that have one or more elements. Solid boxes represent strips that have at least two elements.

We add a final constraint that the prefix transposition should not bring the element 1 to the beginning of the permutation. Therefore, let $\pi_j = c$, we have to guarantee that $c \neq 1$. ρpt前後，其π1≠1

Fig. 1 illustrates the scenarios where $\rho_{pt}(j, k)$ removes two breakpoints. In some cases, more than one scenario is possible for an input permutation π. When that happens, we arrange them in an order of priority where the scenario in Fig. 1(a) is better than the one in Fig. 1(b), which is in turn better than Fig. 1(c), which is again better than Fig. 1(d). a>b = c>d

The implementation is straightforward and requires $O(1)$ time. Lines 9–22 in Algorithm 1 show our implementation. Let $\rho_{pt}(j, k)$ be the prefix transposition, variable *kab* receives the value of k in Fig. 1(a) and (b). Likewise, *kcd* receives the value of k in Fig. 1(c) and (d). Variables *ja*, *jb*, *jc*, and *jd* receive the values of j in each scenario a, b, c and d, respectively.

The following lemma presents what our algorithm does when we cannot remove two prefix reversal breakpoints.

Lemma 10. *If $\pi_1 \neq 1$, then there is at least one prefix transposition $\rho_{pt}(j, k)$ such that $\Delta_{b_{pr}}(\pi, \rho_{pt}) \leq -1$.*

Proof. Let $\pi_1 = a$, we can always find an operation that places a close to $a - 1$ or $a + 1$. We will divide our proof in four cases, depending on whether a is part of an ascending or descending strip. If a is part of a unitary strip, then at least two of the cases below are possible.

If a is part of an ascending strip of length $i + 1$ $\langle a, a + 1, \ldots, a + i\rangle$, then there is a prefix reversal breakpoint whose elements are $a - 1$ and another arbitrary element b.

Case a: If there is a prefix reversal breakpoint $[b, a - 1]$, then we can apply the prefix reversal $\rho_{pr}(k)$ such that $\pi_k = b$. We use this case if $a - 1$ is part of a descending strip $\langle a - 1, a - 2, \ldots, a - l\rangle$ with at least two elements, otherwise we use Case b.

Case b: If there is a prefix reversal breakpoint $[a - 1, b]$, then we can apply the prefix transposition $\rho_{pt}(i + 1, k)$ such that $\pi_k = b$.

If a is part of a descending strip $\langle a, a - 1, \ldots, a - i\rangle$, then there is a prefix reversal breakpoint whose elements are $a + 1$ and another arbitrary element b.

Case c: If there is a prefix reversal breakpoint $[a + 1, b]$, then we can apply the prefix transposition $\rho_{pt}(i + 1, k)$ such that $\pi_k = b$. We use this case if $a + 1$ is part of a descending strip $\langle a + l, \ldots, a + 2, a + 1\rangle$ with at least two elements, otherwise we use Case d.

Case d: If there is a prefix reversal breakpoint $[b, a + 1]$, then we can apply the prefix reversal $\rho_{pr}(k)$ such that $\pi_k = b$. □

Fig. 2 illustrates all the cases. The implementation of the method to remove one prefix reversal breakpoint is straightforward and requires $O(1)$ time. Lines 23–35 in Algorithm 1 show our implementation. Let $\pi_1 = a$, variable *x* receives the position of $a - 1$ in Fig. 2(a) and (b), and variable *y* receives the position of $a + 1$ in Fig. 2(c) and (d).

4.2. Dealing with $\pi_1 = 1$

$\phi(\pi)$為含有1的strip

Let us denote by $\phi(\pi)$ the strip that contains the element 1 in π, which can occur in ascending order ($\phi(\pi) = \langle 1, 2, \ldots, i\rangle$) or descending order ($\phi(\pi) = \langle i, i-1, \ldots, 1\rangle$). This section deals with permutations π such that $\pi_1 = 1$. Our algorithm simply sends $\phi(\pi)$ in ascending order to the end of the permutation, before the element $n+1$ in the extended representation. Lines 6–7 of Algorithm 1 execute this action.

Nothing actually happens in the first time $\phi(\pi)$ is sent to the end of the permutation, what is important is the property shown in Lemma 11 that occurs when a prefix operation takes $\phi(\pi)$ away from there.

Algorithm 1: Sorting by Prefix Reversals and Prefix Transpositions

```
   Input: π, n
1  d ← 0
2  while π ≠ ι do
3      i ← 1
4      while |π_{i+1} − π_i| = 1 do
5          i ← i + 1
6      if π_1 = 1 then
           // Send ⟨π_1, π_2, ..., π_i⟩ to the last position
7          π ← π · ρ_pt(i + 1, n + 1)
8      else
           // Try to remove two prefix reversal breakpoints
9          kab ← π^{-1}_{π_1 − 1} + 1
10         ja ← π^{-1}_{π_kab − 1} + 1; jb ← π^{-1}_{π_kab + 1} + 1
11         kcd ← π^{-1}_{π_1 + 1} + 1
12         jc ← π^{-1}_{π_kcd − 1} + 1; jd ← π^{-1}_{π_kcd + 1} + 1
13         if |π_{kab−1} − π_kab| ≠ 1 then
14             if π_ja ≠ 1 and |π_{ja−1} − π_ja| ≠ 1 then
15                 π ← π · ρ_pt(ja, kab)
16             else if π_jb ≠ 1 and |π_{jb−1} − π_jb| ≠ 1 then
17                 π ← π · ρ_pt(jb, kab)
18         else if |π_{kcd−1} − π_kcd| ≠ 1 then
19             if π_jc ≠ 1 and |π_{jc−1} − π_jc| ≠ 1 then
20                 π ← π · ρ_pt(jc, kcd)
21             else if π_jd ≠ 1 and |π_{jd−1} − π_jd| ≠ 1 then
22                 π ← π · ρ_pt(jd, kcd)
23         else
               // Remove one prefix reversal breakpoint
24             if π_1 ≤ π_i then
                   // π_1 ... π_i is an increasing sequence
25                 x ← π^{-1}_{π_1 − 1}
26                 if π_x = π_{x+1} + 1 then
27                     π ← π · ρ_pr(x − 1)
28                 else
29                     π ← π · ρ_pt(i + 1, x + 1)
30             else
                   // π_1 ... π_i is a decreasing sequence
31                 y ← π^{-1}_{π_1 + 1}
32                 if π_y = π_{y−1} − 1 then
33                     π ← π · ρ_pt(i + 1, y + 1)
34                 else
35                     π ← π · ρ_pr(y − 1)
36     d ← d + 1
37 return d
```

不會移除任何的Breakpoint，但保證最多只出現兩次

將$\phi(\pi)$移到最後

先移除2個breakpoint

後移除1個breakpoint並保持最大strip（方向敏感）

當$\phi(\pi)$是升序在尾端時，演算法1會移動元素n到尾端

Lemma 11. *When $\phi(\pi)$ is in ascending order at the end of the permutation π, the prefix operation performed by Algorithm 1 that moves it away from there forces the element n to be moved to the end of the permutation.*

Proof. It is somewhat obvious, but one should keep in mind that in order to move $\phi(\pi)$ away from the last position the prefix operations must act on it, which means they must have the form $\rho_{pr}(n)$ and $\rho_{pt}(j, n+1)$. 這兩種操作可以將$\phi(\pi)$移到最後

The prefix transpositions shown in Fig. 1 would act on $\phi(\pi)$ only if sub-sequences $\langle b+1, \ldots, n+1\rangle$ in Fig. 1(a) and (c), or $\langle b-1, \ldots, n+1\rangle$ in Fig. 1(b) and (d) were comprised solely of $n+1$.

升序 a會變成$\phi(\pi)$的最後一個元素

In Fig. 1(a) and (b), the prefix transposition will place a after $a-1$, which is the last element in $\phi(\pi)$. Therefore, the prefix transposition does not remove $\phi(\pi)$ from the last position, it just adds new elements to it.

Fig. 1(c) and (d) do not remove $\phi(\pi)$ from the last position because they cannot even act on $\phi(\pi)$. Note that $\phi(\pi)$ is in ascending order and hence the first element cannot be one unit lower than the last element on $\phi(\pi)$, unless $\phi(\pi)$ has only one element, but in this case we would have used the scenario in Fig. 1(a) and (b).

We follow by inspecting the prefix operations shown in Fig. 2, which correspond to Lines 24–35 in Algorithm 1. We inspect each scenario one by one.

- Fig. 2(a): the sub-sequence $\langle a-1,\ldots,a-l,\ldots,n+1\rangle$ should be comprised solely by $n+1$, which does not happen because the strip $\langle a-1,\ldots,a-l\rangle$ has at least two elements. 只能為n-1，但至少要兩元素，故矛盾
- Fig. 2(b): the sub-sequence $\langle b,\ldots,n+1\rangle$ should be comprised solely by $n+1$. So, $a-1$ is the last element in $\phi(\pi)$. The prefix transposition $\rho_{pt}(j,n+1)$ will place $\langle a,\ldots,a+i\rangle$ after $\phi(\pi)$ and hence $\phi(\pi\cdot\rho_{pt}(j,n+1)) = \langle 1,\ldots,a,a+1,\ldots,a_i\rangle$. Therefore, the prefix transposition does not really remove $\phi(\pi)$ from the last position, it just adds new elements.
- Fig. 2(c): the sub-sequence $\langle b,\ldots,n+1\rangle$ should be comprised solely by $n+1$. In addition, this scenario requires the last strip $\langle a+l,\ldots,a+1\rangle$ to have more than two elements and to be in descending order, which cannot be managed since we want the last strip to be $\phi(\pi)$ in ascending order.
- Fig. 2(d): the sub-sequence $\langle a+1,\ldots,n+1\rangle$ should be comprised solely by $n+1$. In addition, $\rho_{pr}(n)$ should bring the first element a close to the last element $a+1$. However, $a+1$ is indeed $n+1$ and hence a is n. So this scenario can force $\phi(\pi)$ to leave the last position, and the side effect is that it also sends the strip that contains the element n in ascending order to the end of the permutation. □

Lemma 12. *Let π be a permutation of the form $\pi = (\ldots\ n\ \phi(\pi)\ n+1)$, $\phi(\pi)$ in ascending order, Algorithm 1 will not split the pair $[n,1]$ unless we can find a prefix transposition ρ_{pt} such that $\pi\cdot\rho_{pt} = \iota$.*

Proof. We start by disregarding any prefix transposition shown in Fig. 1 since they would break the pair $[n,1]$ only if one of two scenarios was true.

- $\pi_j = 1$, then the prefix transposition $\rho_{pt}(j,k)$ would bring the element 1 to the beginning of the permutation, which we explicitly deny.
- $\pi_k = 1$, it implies that we can find a block $\pi[1..j-1]$, $j<k$, such that $\pi_1 = n+1$ and $\pi_{j-1} = 0$, which can never happen.

We follow by inspecting the prefix operations shown in Fig. 2.

- Fig. 2(a): we disregard this prefix reversal because $\langle a-1,\ldots,a-l\rangle$ is a decreasing strip with at least two elements, which cannot be managed since we want that strip to be $\phi(\pi)$ in ascending order.
- Fig. 2(b): the prefix transposition $\rho_{pt}(j,k)$ could lead to one of two scenarios: $\pi j = 1 = c$
  - $\pi_j = 1$, then $\langle a,a+1,\ldots,a+i\rangle$ is a strip such that $a+1 = n+1$. In addition, the sub-sequence $\langle c,\ldots,a-1\rangle$ must be $\phi(\pi)$ and the sub-sequence $\langle b,\ldots,n+1\rangle$ must be comprised solely of $n+1$. Indeed, this entire configuration is a permutation that is just one step away from the identity and $\rho_{pt}(j,k)$ is the operation we need to perform this last step. 這是最後一步的情況
  - $\pi_k = 1$, then $a-1 = n+1$ and the first element a should be equal to $n+2$, which is impossible by definition. So, we disregard this case.
- Fig. 2(c): we disregard this prefix transposition $\rho_{pt}(j,k)$ because it leads to impossible scenarios:
  - $\pi_j = 1$, then the sub-sequence $\langle c,\ldots,a+l,\ldots,a+1\rangle$ should be $\phi(\pi)$ in ascending order, but it is impossible because $\langle a+l,\ldots,a+1\rangle$ is a decreasing strip with at least two elements.
  - $\pi_k = 1$, then $\langle a+l,\ldots,a+1\rangle$ is a decreasing strip with at least two elements ended by n, which is impossible because the element $n+1$ only appears in the extended form and never leaves the end of the permutation.
- Fig. 2(d): we disregard this prefix reversal because it only works if we make $a+1 = 1$, and hence $a = 0$, which is impossible. □

Lemma 13. *The prefix transposition that moves $\pi_1 = 1$ to the end of the permutation occurs at most twice during the execution of Algorithm 1.*

Proof. If element 1 is found in the beginning of the permutation, then $\phi(\pi)$ is moved in ascending order to the end of the permutation. Lemma 11 shows that $\phi(\pi)$ is taken away from there when we are ready to move the element n into its place. After that, no prefix operation in Figs. 1 and 2 will split the pair $[n,n+1]$ because it is not a prefix reversal breakpoint. Therefore, the pair $[n,n+1]$ will be split if the element 1 appears for the second time in the first position. In that case, a prefix transposition will send $\phi(\pi)$ to the end and hence create the permutation $\pi = (\ldots\ n\ \phi(\pi)\ n+1)$ such that $\phi(\pi)$ is in ascending order.

Lemma 12 shows that $[n,1]$ will not be split unless we can find an operation that immediately transforms the permutation into the identity. As a consequence, element 1 will never appear in the beginning of a permutation different from the identity, so the prefix transposition that moves $\pi_1 = 1$ to the end of the permutation will never be used more than twice. □

Lemma 14. *The sequence produced by Algorithm 1 to sort π has at most $b_{pr}(\pi) + 2$ prefix operations.*

Proof. Algorithm 1 always removes one prefix reversal breakpoint if element 1 is not in the beginning of the permutation.

Lemma 13 says that $\pi_1 = 1$ occurs at most twice during the execution of Algorithm 1. Let $\phi(\pi) = \langle 1, 2, \ldots, i\rangle$ and $\rho_{pt}(i+1, n+1)$ be the prefix transposition applied on π, we know that $[\pi_i, \pi_{i+1}]$ is a prefix reversal breakpoint, otherwise π_i would not be the last element in $\phi(\pi)$. Let us assume $[\pi_n, \pi_{n+1}]$ is a prefix reversal breakpoint, which represents the first time we send $\pi_1 = 1$ to the end of the permutation. In this case, $\rho_{pt}(i+1, n+1)$ does not increase nor decrease the number of prefix reversal breakpoints in our sequence. 第二次調動$\phi(\pi)$，會增加一個Breakpoint

We now assume $[\pi_n, \pi_{n+1}]$ is not a prefix reversal breakpoint, which represents the last time we send $\pi_1 = 1$ to the end of the permutation. In this case, $\rho_{pt}(i+1, n+1)$ increases the number of prefix reversal breakpoints by one. However, when that happens, we create the permutation $(\ldots\ n\ \phi(\pi)\ n+1)$, which by Lemma 12 the pair $[n, 1]$ will not be broken until we reach a permutation $\sigma = (0\ i+1\ i+2\ \ldots\ n\ 1\ 2\ \ldots\ i\ n+1)$ that is just one prefix transposition away from the identity. We observe that the last prefix transposition will remove two breakpoints. In short, we increase the number of breakpoints when we send $\pi_1 = 1$ to the end of the permutation for the last time, but we guarantee that later we will always perform an operation that removes two breakpoints, thus we have two operations that remove one breakpoint.

In summary, our sequence would have at most $(b_{pr}(\pi) + 1) + 2 - 1 = b_{pr}(\pi) + 2$ operations, which occurs when we move $\pi_1 = 1$ to the end of the permutation twice. □ 1st 2nd last

Theorem 1. *Algorithm 1 is an asymptotic 2-approximation algorithm to the Sorting by Prefix Reversals and Prefix Transpositions Problem.*

Proof. We know from Lemma 5 that $d_{prpt}(\pi) \geq \frac{b_{pr}(\pi)}{2}$ and we know from Lemma 14 that Algorithm 1 uses at most $b_{pr}(\pi)+2$ operations. So the approximation ratio is $\frac{b_{pr}(\pi)+2}{\frac{b_{pr}(\pi)}{2}} = 2 + \frac{4}{b_{pr}(\pi)}$. The larger the size of permutations, the more likely it is that they have many prefix reversal breakpoints. In this case, $\lim_{b_{pr}(\pi)\to\infty}(2 + \frac{4}{b_{pr}(\pi)}) = 2 + \epsilon$. □

Theorem 2. *Algorithm 1 runs in $O(n^2)$.*

Proof. Lines 6–36 require $O(1)$ time and lines 4–5 require $O(n)$. The while loop in line 2 will run at most $b_{pr}(\pi) + 2 = O(n)$ times according to Lemma 14. Therefore, the stated complexity follows. □

We end this section with a conjecture about the diameter D_{prpt}. This conjecture comes from what we observed in our experiments and is valid for $5 \leq n \leq 13$.

Conjecture 1. *Let $D_{prpt}(n)$ be the greatest distance between two permutations in S_n using prefix reversals and prefix transpositions, then $D_{prpt}(n) = n - \lfloor \frac{n+1}{3} \rfloor$ for $n \geq 5$.*

5. Experimental results

This section presents a comparative analysis of the four algorithms that provide valid solutions to the Sorting by Prefix Reversals and Prefix Transpositions Problem. We implemented all the algorithms in `Python`.

We will denote Algorithm 1 as **2-app**, the 3-approximation algorithm presented by Sharmin et al. [22] as **3-app**, the algorithm presented by Dias and Meidanis [12] for the Sorting by Prefix Transpositions Problem as **ptSort**, and the algorithm presented by Fischer and Ginzinger [16] for the Sorting by Prefix Reversals Problem as **prSort**.

Table 1 presents the average number of operations performed by each algorithm to sort all possible small permutations up to size 12. The total number of permutations used to generate this table was $\sum_{n=2}^{12} n! = 522,956,312$.

In Table 1, the column **Dist** shows the average distance for each set of permutations and the column **Diam** shows the diameter. We observe that **2-app** (Algorithm 1) is the one that provides closest solutions to the real distance followed by **3-app** (Sharmin et al. algorithm).

We expanded our analysis to include large permutations. We chose 100 thousand permutations randomly with size x for each x in the range [2..500]. When $x \leq 8$, the number of distinct permutations is less than 100 thousand, so we simply picked all possible permutations. The total number of permutations used in this second analysis was $\sum_{n=2}^{8} n! + (500 - 8) \times 10^5 = 49,246,232$.

Fig. 3 shows how the approximation ratio behaves when n grows. Since we do not know the exact distance of these permutations, the ratio was calculated between the number of operations answered by each algorithm and the lower bound in Lemma 5.

The **prSort** algorithm obtained poor results as the approximation ratio converges to a value between 2.4 and 2.5. We notice that **2-app** presented an approximation ratio close to 1.4, which is the best result in our analysis. The **3-app** and **ptSort** algorithms behaved alike, and the approximation ratio did not exceed 2. That is an interesting observation because **3-app** was developed to the problem we are dealing with, while **ptSort** was developed to the problem of sorting permutation by prefix transpositions.

Table 1
Average number of operations used by each algorithm to sort a set of all possible permutations of a given size. **2-app** refers to Algorithm 1. **3-app** refers to the 3-approximation algorithm presented by Sharmin et al. [22]. **ptSort** refers to the algorithm presented by Dias and Meidanis [12]. Finally, **prSort** refers to the algorithm presented by Fischer and Ginzinger [16]. The average distance **Dist** and the diameter **Diam** are also given in this table for comparison purposes.

Size	Dist	Diam	Algorithms			
			2-app	3-app	ptSort	prSort
2	0.500	1	0.500	0.500	0.500	0.500
3	1.000	2	1.000	1.167	1.167	1.500
4	1.583	2	1.792	1.875	1.917	2.792
5	2.175	3	2.558	2.617	2.717	4.008
6	2.736	4	3.286	3.408	3.550	5.256
7	3.332	5	4.022	4.238	4.407	6.496
8	3.895	5	4.730	5.096	5.282	7.737
9	4.471	6	5.449	5.974	6.171	8.977
10	5.039	7	6.157	6.867	7.071	10.214
11	5.598	7	6.868	7.772	7.980	11.448
12	6.164	8	7.575	8.686	8.897	12.680

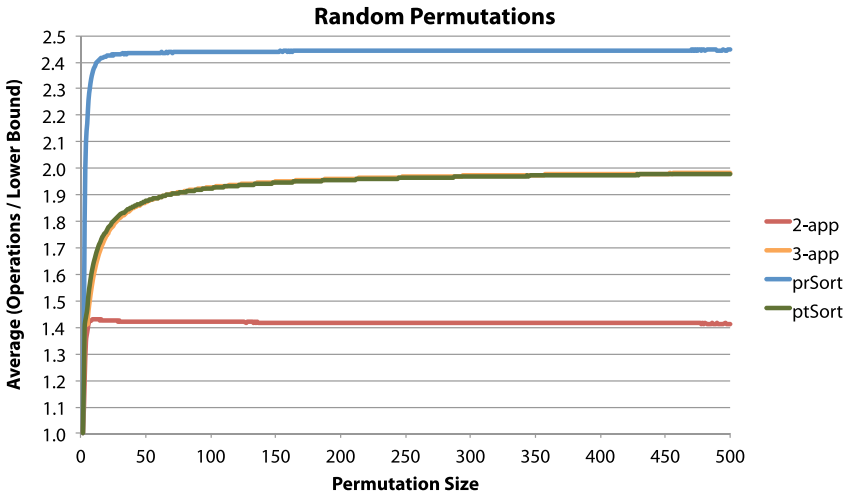


Fig. 3. Approximation ratio of each algorithm when the permutation size grows.

Fig. 4 helps to explain why **3-app** and **ptSort** behave alike. On the left side, we plot the percentage of prefix reversals and prefix transpositions returned by each algorithm and on the right side we show the average decrease in the number of prefix transposition breakpoints caused by the rearrangement operations. Note that here we are using prefix transposition breakpoints instead of prefix reversal breakpoints that were used to prove the approximation ratio. The prefix transposition breakpoints assess how helpful prefix reversals are in finding shorter sorting sequences. However, they do not prove any approximation bound because $d_{prpt}(\pi) \geq \frac{b_{pt}(\pi)}{n}$ as shown by Lemma 7.

Comparing **3-app** with **ptSort**, we observe that the prefix transpositions performed by the former remove, on average, 0.1 less prefix transposition breakpoints than the prefix transpositions performed by the latter. However, they both converge to the same approximation ratio (see Fig. 3) because **3-app** also applies 9.8% of prefix reversals that remove, on average, 2.0 prefix transposition breakpoints.

Fig. 4 also shows that 90.2% of the operations performed by **3-app** are prefix transpositions, which differs from our algorithm **2-app** that uses prefix transpositions in around 70.7% of the cases. By analyzing the variation in the number of prefix transposition breakpoints caused by prefix reversals and prefix transpositions (right side of Fig. 4), we found that prefix reversals remove, on average, 2.2 and 2.0 prefix transposition breakpoints for **2-app** and **3-app**, respectively. Similarly, prefix transpositions remove, on average, 1.1, 0.9 and 1.0 prefix transposition breakpoints for **2-app**, **3-app** and **ptSort**, respectively.

In summary, both prefix reversals and prefix transpositions used by **2-app** remove more prefix transposition breakpoints than their **3-app** counterparts. In addition, **2-app** uses more prefix reversals, which on average remove more prefix transposition breakpoints than prefix transpositions. These facts explain why **2-app** leads to the best results.

We did one last experiment to assess the approximation ratio convergence when the number of prefix reversal breakpoints grows. This test reveals that having permutations with more prefix reversal breakpoints favors the use of prefix transpositions.

具有更多前綴反轉斷點的排列有利於使用前綴換位

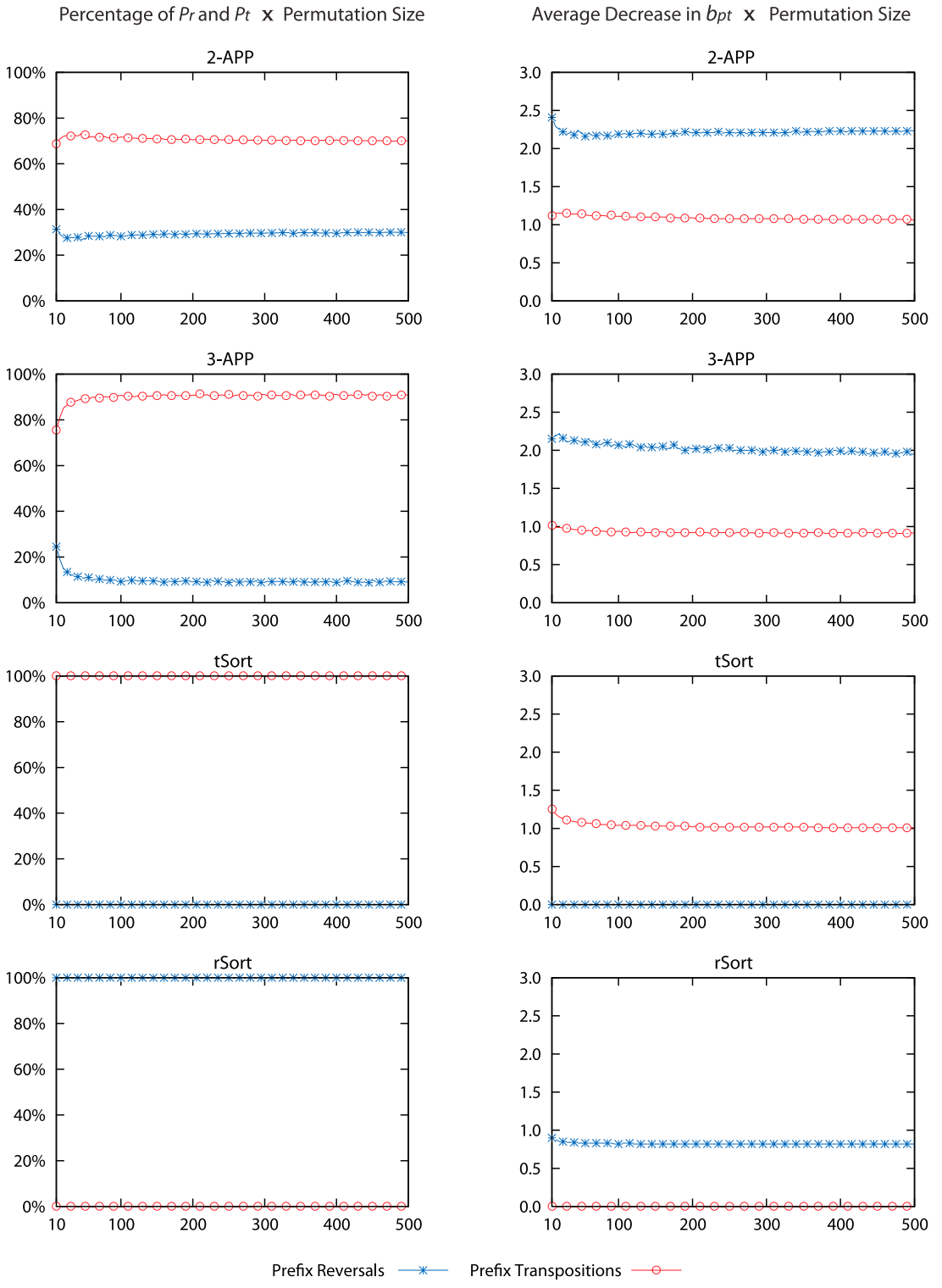


Fig. 4. On the left side, we plot the percentage of prefix reversals and prefix transpositions used by each algorithm. On the right side, we show the average decrease in the number of prefix transposition breakpoints caused by prefix reversals and prefix transpositions.

We randomly generated 100 million permutations with size $n = 100$. We grouped these permutations by the number of prefix reversal breakpoints in order to generate Fig. 5. The Y-axis represents the approximation ratio and the X-axis

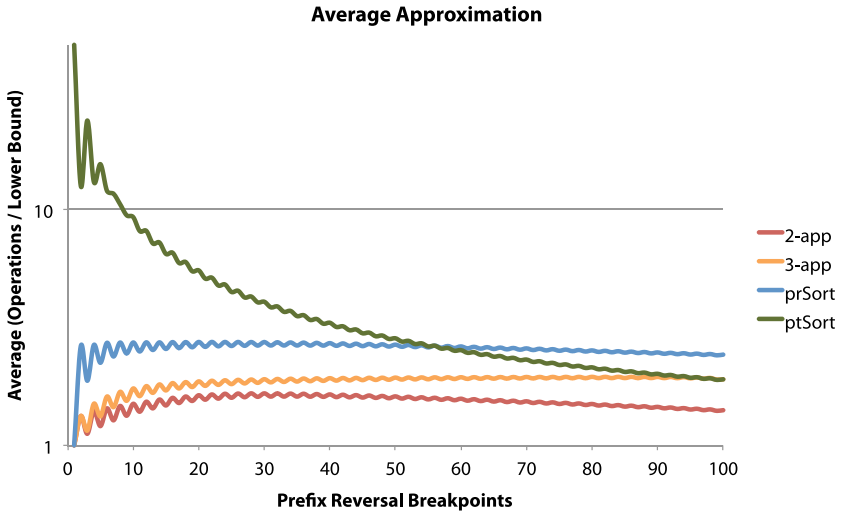


Fig. 5. Approximation ratio convergence when the number of prefix reversal breakpoints grows. The *Y*-axis represents the approximation ratio and the *X*-axis represents the number of prefix reversal breakpoints.

represents the number of prefix reversal breakpoints. The first conclusion is that **2-app** is the best no matter the number of prefix reversal breakpoints. Our curve converges to an approximation average of 1.4.

We also observed that the comparison between **ptSort** and **prSort** depends on the number of prefix reversal breakpoints. The higher the number of prefix reversal breakpoints, the better the results provided by **ptSort**. The intersection point occurred at 55 prefix reversal breakpoints. That behavior indicates that it is easier to remove prefix reversal breakpoints with prefix transpositions when the number of breakpoints is high, which is reasonable.

When the number of prefix reversal breakpoints is close to 100, the average approximation ratio of **ptSort** equals to that of **3-app**. Thus, we finally established which condition makes those algorithms to have similar behavior. In Fig. 5, the curves for **ptSort** and **3-app** converge to an average approximation ratio close to 1.9.

A noteworthy information not evident in the graphs is how often each algorithm provides the smallest sorting sequence. In our experiments, **2-app** systematically found strictly shorter sorting sequences than the other algorithms on permutations longer than 30 elements. For permutations up to size 30, **2-app** provides the smallest sorting sequence in 95.62% of the cases, followed by **3-app** and **ptSort** that provide the smallest sorting sequence in 10.12% and 7.72% of the cases, respectively. These values do not add up to 100% because of ties.

6. Conclusions and future work

In this paper, we presented an asymptotic 2-approximation algorithm for the problem of sorting permutations by prefix reversals and prefix transpositions that runs in $O(n^2)$. This is the best approximation to date.

Our analysis shows that our algorithm is the best on small permutations ranging from 2 to 12 and on large permutations up to 500 elements. That indicates that our approach is better than any other in theoretical and practical aspects.

The approximation factor 2 was proved using the lower bound that states that the number of prefix reversal breakpoints should be twice as higher as the distance. We believe this lower bound can be improved, which could lead to a better approximation factor.

We will keep on studying this problem: our next step is to evaluate the conditions that make our algorithm results differ from the distance. That could possibly lead to improvements on some hard to sort permutations.

The Sorting by Prefix Reversals and Prefix Transpositions Problem is relatively new, and hence few considerations about the diameter have been made. We have analyzed our data and developed a conjecture that is supported by small permutations up to size 13. We intend to study a formal proof for this conjecture.

Another research line we intend to study is the Sorting by Prefix Reversals and Prefix Transpositions Problem on signed permutations. In that case, the reversals change the order of the segment and the sign of each element in this segment. Signed reversals are more significant for the biology because signs can represent gene orientation, which changes when one inversion affects a stretch of DNA sequence in the genome.

Acknowledgments

This work was made possible by a Postdoctoral Fellowship from FAPESP to UD (number 2012/01584-3) and by project funding from CNPq to ZD (numbers 306730/2012-0, 477692/2012-5 and 483370/2013-4).

The authors thank Espaço da Escrita—Coordenadoria Geral da Universidade—UNICAMP—for the language services provided.

The authors thank the Center for Computational Engineering and Sciences at Unicamp for financial support through the FAPESP/CEPID Grant 2013/08293-7.

The authors also acknowledge "Laboratório Multiusuário de Bioinformática da Embrapa" for the use of computational resources.

References

[1] V. Bafna, P. Pevzner, Genome rearrangements and sorting by reversals, SIAM J. Comput. 25 (1996) 272–289.
[2] V. Bafna, P.A. Pevzner, Sorting by transpositions, SIAM J. Discrete Math. 11 (1998) 224–240.
[3] P. Berman, S. Hannenhalli, M. Karpinski, 1.375-approximation algorithm for sorting by reversals, in: Proceedings of the 10th Annual European Symposium on Algorithms, ESA'2002, Rome, Italy, pp. 200–210.
[4] L. Bulteau, G. Fertin, I. Rusu, Pancake flipping is hard, in: Mathematical Foundations of Computer Science 2012, in: Lecture Notes in Computer Science, vol. 7464, 2012, pp. 247–258.
[5] L. Bulteau, G. Fertin, I. Rusu, Sorting by transpositions is difficult, SIAM J. Comput. 26 (2012) 1148–1180.
[6] A. Caprara, Sorting permutations by reversals and eulerian cycle decompositions, SIAM J. Discrete Math. 12 (1999) 91–110.
[7] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. Shields, I. Sudborough, W. Voit, An (18/11)n upper bound for sorting by prefix reversals, Theoret. Comput. Sci. 410 (2009) 3372–3390.
[8] B. Chitturi, I. Sudborough, Bounding prefix transposition distance for strings and permutations, Theoret. Comput. Sci. 421 (2012) 15–24.
[9] D.A. Christie, A 3/2-approximation algorithm for sorting by reversals, in: Proceedings of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA'98, San Francisco, California, United States, pp. 244–252.
[10] U. Dias, Z. Dias, Heuristics for the transposition distance problem, J. Bioinform. Comput. Biol. 11 (2013) 1350013.
[11] U. Dias, G.R. Galvão, C.N. Lintzmayer, Z. Dias, A general heuristic for genome rearrangement problems, J. Bioinform. Comput. Biol. 12 (2014) 1450012.
[12] Z. Dias, J. Meidanis, Sorting by prefix transpositions, in: Proceedings of the 9th International Symposium on String Processing and Information Retrieval, SPIRE'2002, Lisbon, Portugal, pp. 65–76.
[13] H. Dweighter, Problem e2569, Amer. Math. Monthly 82 (1975) 1010.
[14] I. Elias, T. Hartman, A 1.375-approximation algorithm for sorting by transpositions, IEEE/ACM Trans. Comput. Biol. Bioinform. 3 (2006) 369–379.
[15] G. Fertin, A. Labarre, I. Rusu, E. Tannier, S. Vialette, Combinatorics of Genome Rearrangements, The MIT Press, 2009.
[16] J. Fischer, S. Ginzinger, A 2-approximation algorithm for sorting by prefix reversals, in: Algorithms—ESA 2005, in: Lecture Notes in Computer Science, vol. 3669, 2005, pp. 415–425.
[17] W. Gates, C. Papadimitriou, Bounds for sorting by prefix reversal, Discrete Math. 27 (1979) 47–57.
[18] M.H. Heydari, I.H. Sudborough, On the diameter of the pancake network, J. Algorithms 25 (1997) 67–94.
[19] J. Kececioglu, D. Sankoff, Exact and approximation algorithms for the inversion distance between two chromosomes, Algorithmica 13 (1995) 80–110.
[20] A. Labarre, Lower bounding edit distances between permutations, SIAM J. Discrete Math. 27 (2013) 1410–1428.
[21] S. Lakshmivarahan, J.S. Jwo, S.K. Dhall, Symmetry in interconnection networks based on Cayley graphs of permutation groups: a survey, Parallel Comput. 19 (1993) 361–407.
[22] M. Sharmin, R. Yeasmin, M. Hasan, A. Rahman, M.S. Rahman, Pancake flipping with two spatulas, Electron. Notes Discrete Math. 36 (2010) 231–238.