DOI: 10.1007/978-3-642-54423-1_58

Sorting Permutations by Prefix and Suffix Versions of Reversals and Transpositions

Carla Negri Lintzmayer and Zanoni Dias

Institute of Computing, University of Campinas (Unicamp), Brazil
{carlanl,zanoni}@ic.unicamp.br

Abstract. Reversals and transpositions are the most common kinds of genome rearrangements, which allow us to establish the divergence between individuals along evolution. When the rearrangements affect segments from the beginning or from the end of the genome, we say they are prefix or suffix rearrangements, respectively. This paper presents the first approximation algorithms for the problems of Sorting by Prefix Reversals and Suffix Reversals, Sorting by Prefix Transpositions and Suffix Transpositions and Sorting by Prefix Reversals, Prefix Transpositions, Suffix Reversals and Suffix Transpositions, all of them with factor 2. We also present the intermediary algorithms that lead us to the main results.

1 Introduction

We assume that the evolution distance between two individuals is given by the minimum number of rearrangements needed to transform one genome into another. If we represent them as permutations and assume that one is the identity, the problem is to find the minimum number of operations that sort the other.

The problems of Sorting by Reversals and Sorting by Transpositions (the most common rearrangements) are well studied, so that their best-known algorithms have approximation factor 1.375 [3,8]. In addition, both are NP-hard [6,5].

When rearrangements affect segments from the beginning of the genome they are prefix rearrangements. For Sorting by Prefix Reversals and for Sorting by Prefix Transpositions, the best-known algorithms have approximation factor of 2 [9,7]. The former was proved to be NP-hard [4] while the latter remains an open problem. Sharmin *et al.* [11] considered a variation in which prefix reversals and prefix transpositions were allowed and gave a 3-approximation algorithm.

In addition to rearrangements restricted to the prefix of a permutation, it is also possible to consider their suffix version. It is reasonable to believe that it is easier to break a genome at one point than at two or more. Besides, if this happens, either the first or the second part could be reversed; thus, characterizing the prefix/suffix reversals. The same analogy can be used for the prefix/suffix transpositions since it would require breaking a genome at two points, which can be more difficult, but it is still easier than at three points, as a transposition would. However, notice that if a problem involves only prefix rearrangements, there is no need to study a problem that allows only the suffix versions of the

A. Pardo and A. Viola (Eds.): LATIN 2014, LNCS 8392, pp. 671–682, 2014.

same rearrangements, since they are equivalent. Hence, this paper will study problems of sorting permutations by reversals and transpositions involving both prefix and suffix versions of them. Note that there are no records of a similar study in the literature.

The paper is divided as follows: Section 2 presents important definitions related to our problems. Section 3 describes the algorithms developed. Section 4 shows the results. Finally, Section 5 concludes and suggests future work.

2 Definitions

Given a permutation $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, the identity permutation $\iota = (1\ 2\ \ldots\ n)$ and the reverse permutation $\eta = (n\ \ldots\ 2\ 1)$, we introduce now some concepts important to this paper and to the Genome Rearrangements area.

A **composition** between two permutations π and σ is the operation "$\cdot$" in which $\pi \cdot \sigma = (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \ldots\ \pi_{\sigma_n})$. The **inverse** permutation of π is π^{-1}, in which $\pi^{-1}_{\pi_i} = i$, for $1 \leq i \leq n$, and it satisfies $\pi \cdot \pi^{-1} = \iota$.

We extend π by setting $\pi_0 = 0$ and $\pi_{n+1} = n + 1$. We can now define some important concepts related to permutations.

A **reversal** $\rho(i, j)$, $1 \leq i < j \leq n$, is a rearrangement that transforms π into $\pi \cdot \rho(i, j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$. A **prefix reversal** $\rho_p(j)$ is a reversal $\rho(1, j)$, $1 < j \leq n$, while a **suffix reversal** $\rho_s(i)$ is $\rho(i, n)$, $1 \leq i < n$.

A **transposition** $\tau(i, j, k)$, $1 \leq i < j < k \leq n+1$, is a rearrangement that transforms π into $\pi \cdot \tau(i, j, k) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j+1}\ \ldots\ \pi_{k-1}}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}}\ \pi_k\ \ldots\ \pi_n)$. A **prefix transposition** $\tau_p(j, k)$ is a transposition $\tau(1, j, k)$, $2 \leq j < k \leq n+1$, while a **suffix transposition** $\tau_s(i, j)$ is a transposition $\tau(i, j, n+1)$, $1 \leq i < j \leq n$.

If a problem involves some kind of reversal, a breakpoint exists between a pair of consecutive elements π_i and π_{i+1} if $|\pi_{i+1} - \pi_i| \neq 1$. For both Sorting by Prefix Reversals (SBPR) and Sorting by Prefix Reversals and Prefix Transpositions (SBPRPT), $1 \leq i \leq n$ and π_0 and π_1 never form a breakpoint. For both Sorting by Suffix Reversals (SBSR) and Sorting by Suffix Reversals and Suffix Transpositions (SBSRST), $0 \leq i \leq n-1$ and π_n and π_{n+1} never form a breakpoint. For them, ι is the unique permutation without breakpoints. For both Sorting by Prefix Reversals and Suffix Reversals (SBPRSR) and Sorting by Prefix Reversals, Prefix Transpositions, Suffix Reversals and Suffix Transpositions (SBPRPTSRST), $1 \leq i \leq n-1$ and neither π_0 and π_1, nor π_n and π_{n+1} form breakpoints. For them, ι and η are the unique permutation without breakpoints.

If a problem involves only some kind(s) of transposition(s), then a breakpoint exists between a pair of consecutive elements π_i and π_{i+1} if $\pi_{i+1} - \pi_i \neq 1$. For Sorting by Prefix Transpositions (SBPT), $1 \leq i \leq n$ and π_0 and π_1 never form a breakpoint. For Sorting by Suffix Transpositions (SBST), $0 \leq i \leq n-1$ and π_n and π_{n+1} never form a breakpoint. For Sorting by Prefix Transpositions and Suffix Transpositions (SBPTST), $1 \leq i \leq n-1$ and neither π_0 and π_1, nor π_n and π_{n+1} form breakpoints. For them, ι is the unique permutation without breakpoints.

Given a set β of rearrangements allowed in a sorting problem, we denote the number of breakpoints of a permutation by $b_\beta(\pi)$. If there is no breakpoint

between two elements, we say there is an **adjacency** between them. The **sorting distance** of a permutation π, denoted by $d_\beta(\pi)$, is defined as the minimum number of operations in β needed to transform π into ι. Since the identity has the smallest number of breakpoints and it is (usually) the only one with this feature, we can say that sorting π is equivalent to reducing its number of breakpoints. This allows us to establish lower bounds for rearrangement distances.

For SBR, it was proved [1] that $d_\rho(\pi) \geq \lceil b_\rho(\pi)/2 \rceil$. For SBPR, $d_{\rho_p}(\pi) \geq b_{\rho_p}(\pi)$, as demonstrated by Fischer and Ginzinger [9]. For SBT [2], $d_\tau(\pi) \geq \lceil b_\tau(\pi)/3 \rceil$. For SBPT, Dias and Meidanis [7] showed that $d_{\tau_p}(\pi) \geq \lceil b_{\tau_p}(\pi)/2 \rceil$. For SBPRPT, Sharmin *et al.* showed that $d_{\rho_p\tau_p}(\pi) \geq \lceil b_{\rho_p\tau_p}(\pi)/2 \rceil$.

Since the other problems were yet to be considered in the literature, we now define their trivial lower bounds. Because of the equivalences, for SBSR, $d_{\rho_s}(\pi) \geq b_{\rho_s}(\pi)$, for SBST, $d_{\tau_s}(\pi) \geq \lceil b_{\tau_s}(\pi)/2 \rceil$, and for SBSRST, $d_{\rho_s\tau_s}(\pi) \geq \lceil b_{\rho_s\tau_s}(\pi)/2 \rceil$.

Theorem 1. *For an arbitrary permutation π, $d_{\rho_p\rho_s}(\pi) \geq b_{\rho_p\rho_s}(\pi)$, $d_{\tau_p\tau_s}(\pi) \geq \lceil b_{\tau_p\tau_s}(\pi)/2 \rceil$, and $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \geq \lceil b_{\rho_p\tau_p\rho_s\tau_s}(\pi)/2 \rceil$.*

A **strip** is a subsequence $\pi_i, \ldots, \pi_j$ of π, with $1 \leq i \leq j \leq n$, such that (i) either $i = 1$ or π_{i-1} and π_i form a breakpoint; (ii) either $j = n$ or π_j and π_{j+1} form a breakpoint; and (iii) the other elements of the subsequence form adjacencies. A strip of length greater or equal to two is *ascending* if $\pi_k = \pi_{k+1} - 1$ for all $i \leq k < j$. It is *descending* if $\pi_k = \pi_{k+1} + 1$. Otherwise, it is a *singleton*.

A **breakpoint graph** [1] of a permutation π is a graph $G(\pi) = (V, E)$ in which $V = \{\pi_0, \pi_1, \ldots, \pi_{n+1}\}$ and E contains *black edges* and *gray edges*. A black edge e exists if and only if (i) $e = (\pi_i, \pi_{i+1})$ and π_i and π_{i+1} form a breakpoint, $0 \leq i \leq n$; (ii) $e = (\pi_0, \pi_1)$ and prefix operations are involved; and (iii) $e = (\pi_n, \pi_{n+1})$ and suffix operations are involved. A gray edge e exists if and only if $e = (\pi_i, \pi_j)$ for some $0 \leq i < j \leq n+1$ with $\pi_j = \pi_i \pm 1$ and $j \neq i+1$. The convention is to draw black edges as straight lines and gray edges as dashed arcs.

Let (π_i, π_j) be a gray edge. Since $\pi_j = \pi_i \pm 1$, at least one black edge either begins or ends at π_i as well as at least one black edge either begins or ends at π_j. Hence, we can classify such edge into at least one of the four types in Fig. 1.

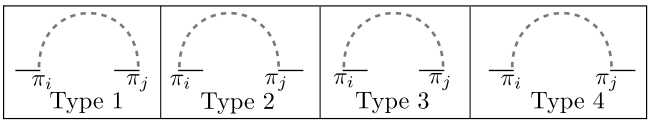

Fig. 1. Classification of gray edges

3 Algorithms

The following subsections describe the algorithms that we have developed, in addition to some of the existing algorithms related to ours.

3.1 Algorithms for Sbprsr

Fischer and Ginzinger [9] published the first 2-approximation algorithm (and the best so far) for Sbpr, which we will call `2-PR`. They used the breakpoint graph and defined requirements for each type of gray edge. Based on that, it is possible to establish what to do when considering not only prefix reversals, but also suffix reversals. Lemma 1 shows what we called *good edges*: edges for which is possible to remove one breakpoint with one or two reversals. `2-PR` only deals with *good prefix edges*, that is, items 1, 3, 5, and 7 of Lemma 1 without the constraints over j.

Lemma 1. *Let π be an arbitrary permutation. There is a sequence of at most two prefix reversals or suffix reversals that removes one breakpoint if $G(\pi)$ contains at least one gray edge (π_i, π_j): (1) of type 1 with $i = 1$ and $j \leq n$; (2) of type 2 with $j = n$ and $i \geq 1$; (3) of type 3 with $i = 1$ and $j \leq n$; (4) of type 3 with $j = n$ and $i \geq 1$; (5) of type 2 with $i \neq 0$ and $j \leq n$; (6) of type 1 with $j \neq n + 1$ and $i \geq 1$; (7) of type 3 with $i > 1$ and $j \leq n$; (8) of type 3 with $j < n$ and $i \geq 1$.*

Proof. If (π_i, π_j) is a gray edge, then $\pi_j = \pi_i \pm 1$. To create an adjacency between π_i and π_j without creating new breakpoints one must perform, for each type of edge: (1) one prefix reversal $\rho_p(j-1)$; (2) one suffix reversal $\rho_s(i+1)$; (3) one prefix reversal $\rho_p(j-1)$; (4) one suffix reversal $\rho_s(i+1)$; (5) two prefix reversals $\rho_p(j)$ and $\rho_p(j-i)$; (6) two suffix reversals $\rho_s(i)$ and $\rho_s(n+1-(j-i))$; (7) two prefix reversals $\rho_p(i)$ and $\rho_p(j-1)$; (8) two suffix reversals $\rho_s(j)$ and $\rho_s(i+1)$. □

If a permutation π does not contain good prefix edges, then it is of the form $\pi = (\underbrace{p_1 \ \ldots \ 1}_{\ell_1} \ \underbrace{p_2 \ \ldots \ p_1{+}1}_{\ell_2} \ \ldots\ldots \ \underbrace{t \ \ldots \ p_{b_{\rho_p}(\pi)-1}{+}1}_{\ell_{b_{\rho_p}(\pi)}} \ t{+}1 \ t{+}2 \ \ldots \ n)$ with $t \leq n$, that is, π consists in $b_{\rho_p}(\pi) \geq 2$ decreasing strips of size $\ell_i \geq 2$, $1 \leq i \leq b_{\rho_p}(\pi)$ [9]. The following $2b_{\rho_p}(\pi)$ prefix reversals transforms π into ι: $\rho_p(t) \cdot \rho_p(t-\ell_1) \cdot \rho_p(t) \cdot \rho_p(t-\ell_2) \cdot \ldots \cdot \rho_p(t) \cdot \rho_p(t-\ell_{b_{\rho_p}(\pi)})$.

`2-PR` scans π from left to right trying to find a good prefix edge in $G(\pi)$ in the order that the four appear in Lemma 1. If a good prefix edge does not exist, the algorithm applies the sequence given above, guaranteeing that $d_{\rho_p}(\pi) \leq 2b_{\rho_p}(\pi)$. Using the lower bound, we can see that it is indeed a 2-approximation algorithm.

By searching for a good prefix edge from right to left on a permutation, we created a new algorithm, `2-PRg`. Other than that, it works exactly as `2-PR`. Then we simply modified `2-PR` and `2-PRg` for Sbsr and called the new algorithms `2-SR` and `2-SRg`, respectively. They only deal with the gray edges presented in items 2, 4, 6, and 8 of Lemma 1 without the constraints over i, called *good suffix edges*.

Finally, we created two algorithms for Sbprsr, which will be called `2-PRSR` and `2-PRSRg`, respectively. Both of them search for any of the eight good edges given by Lemma 1, in that order. The only difference between them is how to scan the permutation: `2-PRSR` searches for good prefix edges from right to left and for good suffix edges from left to right and `2-PRSRg` does the opposite.

When a permutation does not contain a good edge, it is of one of the forms shown by Lemma 2. Now, we can transform it into ι with at most $b_{\rho_p\rho_s}(\pi) + 2$

reversals, as Lemma 3 shows. Despite this, `2-PRSR` and `2-PRSRg` still use 2 operations to eliminate one breakpoint sometimes, leading to either the identity or the reverse permutation. Therefore, $d_{\rho_p\rho_s}(\pi) \leq 2b_{\rho_p\rho_s}(\pi)+1$ and both algorithms have asymptotic approximation factor of 2.

Lemma 2. *If a permutation π does not contain a good edge, then it is of one of the three forms: (1) η; or (2) $\sigma^1 = (\underbrace{p_1 \ \ldots \ 1}_{\ell_1} \ \underbrace{p_2 \ \ldots \ p_1+1}_{\ell_2} \ \ldots\ldots \ \underbrace{n \ \ldots \ p_b+1}_{\ell_{b+1}})$; or (3) $\sigma^2 = (\underbrace{p_b+1 \ \ldots \ n}_{\ell_1} \ \ldots\ldots \ \underbrace{p_1+1 \ \ldots \ p_2}_{\ell_b} \ \underbrace{1 \ \ldots \ p_1}_{\ell_{b+1}})$ where $b = b_{\rho_p\rho_s}(\pi)$ and $\ell_i \geq 2$ for all $1 \leq i \leq b+1$.*

Proof. Omitted due to space restrictions. □

Lemma 3. *Let π be one of the three permutations shown by Lemma 2. If $\pi = \eta$, one reversal $\rho_p(n)$ sorts π. Otherwise, at most $b_{\rho_p\rho_s}(\pi)+2$ reversals sort π.*

Proof. Let $b = b_{\rho_p\rho_s}(\pi)$. If $\pi = \sigma^1$ and b is an odd number, then the $b+1$ reversals $\rho_s(\ell_1+1) \cdot \rho_p(n-\ell_2) \cdot \rho_s(\ell_3+1) \cdot \rho_p(n-\ell_4) \cdot \ldots \cdot \rho_s(\ell_b+1) \cdot \rho_p(n-\ell_{b+1})$ transform π into ι, as we show next. Let π^k, $1 \leq k \leq \frac{b-1}{2}$, be the permutation we obtain after applying the first $2k$ prefix reversals of the sequence given above: $\rho_s(\ell_1+1) \cdot \rho_p(n-\ell_2) \cdot \ldots \cdot \rho_s(\ell_{2k-1}+1) \cdot \rho_p(n-\ell_{2k})$. We will show by induction on k that π^k is of the form $(\underbrace{p_{2k+1} \ldots p_{2k}+1}_{\ell_{2k+1}} \ \underbrace{p_{2k+2} \ldots p_{2k+1}+1}_{\ell_{2k+2}})$ $\ldots\ldots \ \underbrace{n \ldots p_b+1}_{\ell_{b+1}} \ \underbrace{1 \ 2 \ \ldots \ p_{2k-2}+1 \ldots p_{2k-1} \ p_{2k-1}+1 \ldots p_{2k}}_{\ell_1+\ell_2+\ldots\ell_{2k}})$.

It is easy to see that it holds for $k=1$. Now, assume that π^{k-1} is of the form given above. Since $\pi^k = \pi^{k-1} \cdot \rho_s(\ell_{2k-1}+1) \cdot \rho_p(n-\ell_{2k})$, the result follows. At the end, $\iota = \pi^{(b-1)/2} \cdot \rho_s(\ell_b+1) \cdot \rho_p(n-\ell_{b+1})$.

If $\pi = \sigma^2$ and b is odd, one must apply $\rho_p(n)$ to transform it into σ^1 and then apply the $b+1$ reversals given above.

If $\pi = \sigma^2$ and b is an even number, then the $b+1$ reversals $\rho_p(n-\ell_{b+1}) \cdot \rho_s(\ell_b+1) \cdot \rho_p(n-\ell_{b-1}) \cdot \rho_s(\ell_{b-2}+1) \cdot \ldots \cdot \rho_p(n-\ell_3) \cdot \rho_s(\ell_2+1) \cdot \rho_p(n-\ell_1)$ sort π. This also can be shown by a similar induction as the one above. If $\pi = \sigma^1$ and b is even, one must apply $\rho_p(n)$ to transform it into σ^2 and then apply the reversals given. □

3.2 Algorithms for Sbptst

Dias and Meidanis [7] presented a 2-approximation algorithm for Sbpt, here called `2-PT`, which always removes one breakpoint with one prefix transposition. They also demonstrated that there is at most one prefix transposition that removes two breakpoints at once, which leads to a greedy 2-approximation algorithm for the problem, called `2-PTg` [10]: first it tries to remove two breakpoints and if this is not possible, it removes only one, as `2-PT` does. Therefore, both

guarantee that $d_{\tau_p}(\pi) \leq b_{\tau_p}(\pi) - 1$, since the last transposition always removes two breakpoints [7], and both are 2-approximation algorithms.

It is simple to make suffix versions of both `2-PT` and `2-PTg`, which we will call `2-ST` and `2-STg`, respectively. `2-STg` also tries to remove two breakpoints at once. If this is not possible, then it removes only one, as `2-ST` does. So, both also guarantee that $d_{\tau_s}(\pi) \leq b_{\tau_s}(\pi) - 1$.

We created two algorithms for Sbptst, called `2-PTST` and `2-PTSTg`. The former always removes one breakpoint at a time, randomly choosing between a prefix or a suffix transposition to do so. To remove one breakpoint with one prefix transposition $\tau_p(i+1, j)$, let π_i be the last element of the first strip of an arbitrary permutation π. If $\pi_i = n$, then choose $j = \pi^{-1}_{\pi_1 - 1} + 1$. Otherwise, choose $j = \pi^{-1}_{\pi_i + 1}$. To remove one breakpoint with one suffix transposition $\tau_s(i+1, j)$, let π_j be the first element of the last strip of π. If $\pi_j = 1$, then choose $i = \pi^{-1}_{\pi_n + 1} - 1$. Otherwise, choose $i = \pi^{-1}_{\pi_j - 1}$. The basic idea is to increase the first or last strip with either their previous or their next element.

`2-PTSTg` is more interesting, since it tries to remove two breakpoints using either prefix or suffix. A prefix transposition $\tau_p(i, j)$ removes two breakpoints from π if $j = \pi^{-1}_{\pi_1 - 1} + 1$, $i = \pi^{-1}_{\pi_j - 1} + 1$ and $2 \leq i < j \leq n$. It is easy to see that π_1 determines uniquely j and j determines uniquely i. A suffix transposition $\tau_s(i, j)$ removes two breakpoints from π if $i = \pi^{-1}_{\pi_n + 1}$, $j = \pi^{-1}_{\pi_{i-1} + 1}$ and $2 \leq i < j \leq n$. Again, π_n determines uniquely i and i determines uniquely j. If this removal is not possible, then we have to choose how to remove only one breakpoint as described above, which is always possible. Therefore, $d_{\tau_p \tau_s}(\pi) \leq b_{\tau_p \tau_s}(\pi)$. Also, note that the last transposition removes only one breakpoint. Hence, `2-PTST` and `2-PTSTg` are 2-approximation algorithms.

3.3 Algorithms for Sbprptsrst

Sharmin *et al.* [11] presented the Sorting by Prefix Reversals and Prefix Transpositions problem and provided a 3-approximation algorithm, called here `3-PRPT`. It also uses the breakpoint graph to decide which operation to perform and it is similar to `2-PR`; however, the use of a second operation allows the four types of gray edges to be considered *good edges*. In addition, they gave an important concept for their algorithm and for ours, presented in Lemma 4. Now, based on their work we can define what to do with each type of gray edge while also considering suffix reversals and suffix transpositions, as Lemma 5 shows.

Lemma 4. *[11] Let (π_i, π_j) be a gray edge of type 1. Then there is at least one black edge (π_{k-1}, π_k) for some $i < k < j$, that is called a* **trapped black edge**.

Lemma 5. *Let π be an arbitrary permutation. There is a sequence of at most three prefix reversals, prefix transpositions, suffix reversals or suffix transpositions that removes at least one breakpoint if $G(\pi)$ contains at least one gray edge (π_i, π_j): (1) of type 4 with $\pi_1 \neq 1$, $i = 1$, and $j \leq n$; (2) of type 4 with $\pi_n \neq n$, $j = n$, and $i \geq 1$; (3) of type 1 with $\pi_1 \neq 1$, $i = 1$, and $j \leq n$; (4) of type 2 with $\pi_n \neq n$, $j = n$, and $i \geq 1$; (5) of type 3 with $\pi_1 = 1$, $i \geq 1$, and $j \leq n$;*

(6) of type 3 with $\pi_n = n$, $i \geq 1$, and $j \leq n$; (7) of type 2 with $\pi_1 = 1$, $i \geq 1$, and $j \leq n$, where π_i is the last element of the first strip of π; (8) of type 1 with $\pi_n = n$, $i \geq 1$, and $j \leq n$, where π_j is the first element of the last strip of π.

Proof. If there is a gray edge (π_i, π_j) then $\pi_j = \pi_i \pm 1$. To create an adjacency between π_i and π_j without creating new breakpoints one must perform, respectively, for each type of edge: (1) one prefix transposition $\tau_p(k, j+1)$ where (π_{k-1}, π_k) is a trapped black edge, $i < k < j$; (2) one suffix transposition $\tau_s(k, j+1)$ where (π_{k-1}, π_k) is a trapped black edge, $i < k < j$; (3) one prefix reversal $\rho_p(j-1)$; (4) one suffix reversal $\rho_s(i+1)$; (5) one prefix transposition $\tau_p(i+1, j)$; (6) one suffix transposition $\tau_s(i+1, j)$; (7) one prefix reversal $\rho_p(j)$, followed by one prefix reversal $\rho_p(j-i)$, and by one operation to handle an edge of type 4 or 1; (8) one suffix reversal $\rho_s(i)$, followed by one suffix reversal $\rho_s(n+1-(j-i))$, and by one operation to handle an edge of type 4 or 2. □

`3-PRPT` only deals with the gray edges shown in items 1, 3, 5, and 7 of Lemma 5 without the constraints over j, also called *good prefix edges*. It scans $G(\pi)$ from left to right to find its first good prefix edge, it decides its type (in the order that the four appear in the lemma), in addition to performing the required operation(s). Thus, it guarantees that $d_{\rho_p\tau_p}(\pi) \leq 3b_{\rho_p\tau_p}(\pi)/2$, which, using the lower bound, proves that it has an approximation factor of 3.

As explained, if a good prefix edge is of types 3 or 4, the algorithm applies one prefix transposition. However, since a prefix transposition can remove at most 2 breakpoints, we developed a greedy version, which we will call `3-PRPTg`, whose features are: (i) it scans the permutation from right to left; (ii) it tries to find gray edges in a different order, namely items 1, 5, 3, and 7 of Lemma 5; (iii) when there is an edge (π_i, π_j) of type 4, it tries to find the best trapped black edge (π_{k-1}, π_k), $i < k < j$, such that $\pi_{j+1} = \pi_k \pm 1$ and $j \leq n-1$; (iv) when it is trying to find an edge of type 3, it searches for a π_j such that $\pi_1 = \pi_{j-1} \pm 1$.

The suffix versions of both `3-PRPT` and `3-PRPTg` will be called `3-SRST` and `3-SRSTg`, respectively. Of course, they only deal with gray edges given by items 2, 4, 6, and 8 of Lemma 5 without the constraints over i, which are *good suffix edges*. They work similarly to their prefix versions, but `3-SRST` scans the permutation from right to left and `3-SRSTg` scans from left to right. Besides, `3-SRSTg` (i) searches for gray edges in the order of the items 2, 6, 4, and 8 of Lemma 5; (ii) when there is an edge (π_i, π_j) of type 4, it tries to find the best trapped black edge (π_{k-1}, π_k), $i < k < j$, such that $\pi_{j+1} = \pi_k \pm 1$ and $i \geq 2$; and (iii) tries to find a π_i such that $\pi_n = \pi_{i+1} \pm 1$ when it is searching for a type 3 edge.

Finally, we created `2-PRPTSRST` and `2-PRPTSRSTg`, algorithms for SBPRPT-SRST. They can handle all the good edges described in Lemma 5, but they do not consider the edges described in items 7 and 8. When the other six edges does not exist, π is of one of the forms shown by Lemma 6 and the algorithms perform either a prefix reversal $\rho_p(n)$ or a prefix transposition to concatenate the first strip with the last one. Because of this, they can never separate the elements n and 1, unless the black edge between them is the last one (disregard the edges (π_0, π_1) and (π_n, π_{n+1})). This will guarantee that $d_{\rho_p\tau_p\rho_s\tau_s}(\pi) \leq b_{\rho_p\tau_p\rho_s\tau_s}(\pi)+2$,

Algorithm 1. Good edges of type 4

PRPTSRST_EDGE_TYPE_4(π, $\mathtt{n}$)
1 **if** $\pi_1 \neq 1$ **and** $G(\pi)$ has a GPE $(1, \pi_{\mathtt{jp}})$ of type 4 **and** $\mathtt{jp} \leq \mathtt{n}$ **then**
2 $(\mathtt{kp}-1, \mathtt{kp}) \leftarrow$ trapped black edge;
3 **if** $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE $(\pi_{\mathtt{is}}, \mathtt{n})$ of type 4 **and** $\mathtt{is} \geq 1$ **then**
4 $(\mathtt{ks}-1, \mathtt{ks}) \leftarrow$ trapped black edge;
5 **if** $\pi_{\mathtt{kp}} - 1 = \pi_{\mathtt{jp}+1} \pm 1$ **then** $\pi \leftarrow \pi \cdot \tau_p(\mathtt{kp}, \mathtt{jp}+1)$
6 **else if** $\pi_{\mathtt{ks}} - 1 = \pi_{\mathtt{js}+1} \pm 1$ **then** $\pi \leftarrow \pi \cdot \tau_s(\mathtt{ks}, \mathtt{js}+1)$
7 **else if** $\mathtt{jp} < \mathtt{n} - \mathtt{is}$ **then** $\pi \leftarrow \pi \cdot \tau_p(\mathtt{kp}, \mathtt{jp}+1)$
8 **else** $\pi \leftarrow \pi \cdot \tau_s(\mathtt{ks}, \mathtt{js}+1)$
9 **else** $\pi \leftarrow \pi \cdot \tau_p(\mathtt{kp}, \mathtt{jp}+1)$
10 **else if** $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE $(\pi_{\mathtt{is}}, \mathtt{n})$ of type 4 **and** $\mathtt{is} \geq 1$ **then**
11 $(\mathtt{ks\text{-}1}, \mathtt{ks}) \leftarrow$ trapped black edge;
12 $\pi \leftarrow \pi \cdot \tau_s(\mathtt{ks}, \mathtt{js}+1)$
13 **return** π

Algorithm 2. Good prefix edges of type 1 and good suffix edges of type 2

PRPTSRST_EDGE_TYPE_1_2(π, $\mathtt{n}$)
1 **if** $\pi_1 \neq 1$ **and** $G(\pi)$ has a GPE $(1, \pi_{\mathtt{jp}})$ of type 1 **and** $\mathtt{jp} \leq \mathtt{n}$ **then**
2 **if** $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE $(\pi_{\mathtt{is}}, \mathtt{n})$ of type 2 **and** $\mathtt{is} \geq 1$ **then**
3 **if** $\mathtt{jp} < \mathtt{n} - \mathtt{is}$ **then** $\pi \leftarrow \pi \cdot \rho_p(\mathtt{jp}-1)$
4 **else** $\pi \leftarrow \pi \cdot \rho_s(\mathtt{is}+1)$
5 **else** $\pi \leftarrow \pi \cdot \rho_p(\mathtt{jp}-1)$
6 **else if** $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE $(\pi_{\mathtt{is}}, \mathtt{n})$ of type 2 **and** $\mathtt{jp} \leq \mathtt{n}$ **then**
7 $\pi \leftarrow \pi \cdot \rho_s(\mathtt{is}+1)$
8 **return** π

as Theorem 2 shows. With the lower bound, we can prove that the asymptotic approximation factor of both algorithms is 2.

The difference between `2-PRPTSRST` and `2-PRPTSRSTg` is that the former searches for good prefix edges from left to right, searches for good suffix edges from right to left and follows the order given by the lemma. The latter searches for good prefix edges from right to left, searches for good suffix edges from left to right, follows the order of items 1, 2, 5, 6, 3, and 4 of Lemma 5, and tries to find edges of types 3 and 4 that allow the removal of 2 breakpoints at once. Algs. 4 and 5 present them, respectively. In the algorithms, GPE stands for good prefix edge while GSE stands for good suffix edge.

Lemma 6. *Let $\pi \neq \iota$ be a permutation without the first six edges of Lemma 5. Then π is either η, or of the form $\pi = (\underline{1\ 2\ \ldots\ k}\ \ldots\ldots\ \underline{k{+}i\ \ldots\ k{+}2\ k{+}1}\ \ldots\ldots\ \underline{j{-}1\ j{-}2\ \ldots\ j{-}\ell}\ \ldots\ldots\ \underline{j\ j{+}1\ \ldots\ n})$ with $i \geq 2$ and $\ell \geq 2$, or of the form $\pi = (\underline{n\ n{-}1\ \ldots\ j}\ \pi_{n-j+2}\ \ldots\ldots\ \pi_{n-k}\ \underline{k\ k{-}1\ \ldots\ 1})$ with $\pi_{n-j+2} \neq j-1$ and $\pi_{n-k} \neq k+1$.*

Lemma 7. *Let $\pi \neq \eta$ be of one of the two other permutations given in Lemma 6. One transposition $\tau_p(i{+}1, n{+}1)$, where π_i is the last element of the first strip, transforms π into either $\pi \cdot \tau_p = (\ldots\ldots\ \underline{j\ j{+}1\ \ldots\ n{-}1\ n}\ \underline{1\ 2\ \ldots\ k{-}1\ k})$ or $\pi \cdot \tau_p$*

Algorithm 3. Good edges of type 3

PRPTSRST_EDGE_TYPE_3(π, $\mathtt{n}$)

if $\pi_1 = 1$ **and** $G(\pi)$ has a GPE $(\pi_{\mathtt{ip}}, \pi_{\mathtt{jp}})$ of type 3 **then**
  if $\pi_{\mathtt{n}} = \mathtt{n}$ **and** $G(\pi)$ has a GSE $(\pi_{\mathtt{is}}, \pi_{\mathtt{js}})$ of type 3 **then**
    if $\pi_1 = \pi_{\mathtt{jp}-1} \pm 1$ **then** $\pi \leftarrow \pi \cdot \tau_p(\mathtt{ip}+1, \mathtt{jp})$
    else if $\pi_{\mathtt{n}} = \pi_{\mathtt{is}+1} \pm 1$ **then** $\pi \leftarrow \pi \cdot \tau_s(\mathtt{is}+1, \mathtt{js})$
    else if $\mathtt{jp} < \mathtt{n} - \mathtt{is}$ **then** $\pi \leftarrow \pi \cdot \tau_p(\mathtt{ip}+1, \mathtt{jp})$
    else $\pi \leftarrow \pi \cdot \tau_s(\mathtt{is}+1, \mathtt{js})$
  else $\pi \leftarrow \pi \cdot \tau_p(\mathtt{ip}+1, \mathtt{jp})$
else if $\pi_{\mathtt{n}} = \mathtt{n}$ **and** $G(\pi)$ has a GPE $(\pi_{\mathtt{is}}, \pi_{\mathtt{js}})$ of type 3 **then**
  $\pi \leftarrow \pi \cdot \tau_s(\mathtt{is}+1, \mathtt{js})$
return π

Algorithm 4. A 2-approximation algorithm for SBPRPTSRST

2-PRPTSRST(π, $\mathtt{n}$)

while $\pi \neq \iota$ **do**
  if $\pi_1 \neq 1$ **and** $G(\pi)$ has a GPE of type 4 **or**
  $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE of type 4 **then**
    $\pi \leftarrow$ PRPTSRST_EDGE_TYPE_4(π, $\mathtt{n}$)
  else if $\pi_1 \neq 1$ **and** $G(\pi)$ has a GPE of type 1 **or**
  $\pi_{\mathtt{n}} \neq \mathtt{n}$ **and** $G(\pi)$ has a GSE of type 2 **then**
    $\pi \leftarrow$ PRPTSRST_EDGE_TYPE_1_2(π, $\mathtt{n}$)
  else if $\pi_1 = 1$ **and** $G(\pi)$ has a GPE of type 3 **or**
  $\pi_{\mathtt{n}} = \mathtt{n}$ **and** $G(\pi)$ has a GSE of type 3 **then**
    $\pi \leftarrow$ PRPTSRST_EDGE_TYPE_3(π, $\mathtt{n}$)
  else if $\pi = \eta$ **then**
    $\pi \leftarrow \pi \cdot \rho_p(\mathtt{n})$
  else
    Let $\mathtt{k}$ be the position of the last element of the first strip of π
    $\pi \leftarrow \pi \cdot \tau_p(\mathtt{k}+1, \mathtt{n}+1)$

$= (\ldots\ldots\ \underline{k\ k{-}1\ \ldots\ 2\ 1}\ \underline{n\ n{-}1\ \ldots\ j{+}1\ j})$ *without changing the number of breakpoints. After that, it is always possible to keep removing at least one breakpoint with one operation, if the algorithms never separate the elements* 1 *and* n.

Lemma 8. *Let* (π_i, π_j) *be a gray edge of type 4 of either prefix or suffix of an arbitrary permutation* $\pi \neq \iota$. *If the edge between the elements* 1 *and* n *is the only trapped black edge between* π_i *and* π_j, *then actually* $i = 1$, $j = n$ *and the permutation is either of the form* $\pi' = (\underline{k\ k{-}1\ k{-}2\ \ldots\ 2\ 1}\ \underline{n\ n{-}1\ \ldots\ k{+}2\ k{+}1})$ *or of the form* $\pi'' = (\underline{k{+}1\ k{+}2\ \ldots\ n{-}1\ n}\ \underline{1\ 2\ \ldots\ k{-}2\ k{-}1\ k})$.

Besides, by acting on such edge, `2-PRPTSRST` *and* `2-PRPTSRSTg` *will be performing either their last or their last but one operation.*

Lemma 9. *The operation explained at Lemma 7 is performed at most once by* `2-PRPTSRST` *and* `2-PRPTSRSTg`, *if both never separate the elements* 1 *and* n.

Theorem 2. *Both algorithms* `2-PRPTSRST` *and* `2-PRPTSRSTg` *sort any permutation* $\pi \neq \iota$ *using at most* $b_{\rho_p\tau_p\rho_s\tau_s}(\pi) + 2$ *operations.*

Algorithm 5. A 2-approximation algorithm for SBPRPTSRST, greedy version

```
2-PRPTSRSTG(π, n)
while π ≠ ι do
    if π_1 ≠ 1 and G(π) has a GPE of type 4 or
    π_n ≠ n and G(π) has a GSE of type 4 then
        π ← PRPTSRST_EDGE_TYPE_4(π, n)
    else if π_1 = 1 and G(π) has a GPE of type 3 or
    π_n = n and G(π) has a GSE of type 3 then
        π ← PRPTSRST_EDGE_TYPE_3(π, n)
    else if π_1 ≠ 1 and G(π) has a GPE of type 1 or
    π_n ≠ n and G(π) has a GSE of type 2 then
        π ← PRPTSRST_EDGE_TYPE_1_2(π, n)
    else if π = η then
        π ← π · ρ_p(n)
    else
        Let k be the position of the last element of the first strip of π
        π ← π · τ_p(k + 1, n + 1)
```

Proof. Directly from Lemmas 6, 7, 8, and 9, whose proofs were omitted due to space restrictions.

4 Results

All the algorithms have complexity $O(n^2)$, since the distance is $O(n)$ and they spent linear time to choose and to apply an operation at each step. They were implemented in C language and executed in a Intel Core 2 of 2.13 GHz, 4GB RAM running Ubuntu 12.04.2 LTS under the same set of 190000 arbitrary permutations, being 10000 of each size n, for n varying between 10 and 1000 in intervals of 5. Figure 2 shows the results. The x-axis presents a value of n and the y-axis presents the average of the approximation factors of the permutations of that size, calculated using the theoretical lower bound of the distance.

We can see that the simple change of scanning the permutation at a different order had better results. This means that bigger operations (specially reversals) are preferable. As expected, problems which involve only prefix rearrangements are equivalent to their suffix versions. Besides, it was expected that problems with both prefix and suffix versions of a rearrangement would obtain better results than those that allow only the prefix version. It is interesting to notice that this did not happen for `2-PTST`. Finally, for $n \geq 100$, the average approximation factor of `2-PRSRg` is below 1.131, of `2-PTSTg` is below 1.314 and of `2-PRPTSRSTg` is below 1.382, which are the best algorithms for the three new problems we presented. Besides, the maximum factor of this three problems over all permutations tested is below 1.342, 1.596 and 1.600, respectively, when $n \geq 100$.

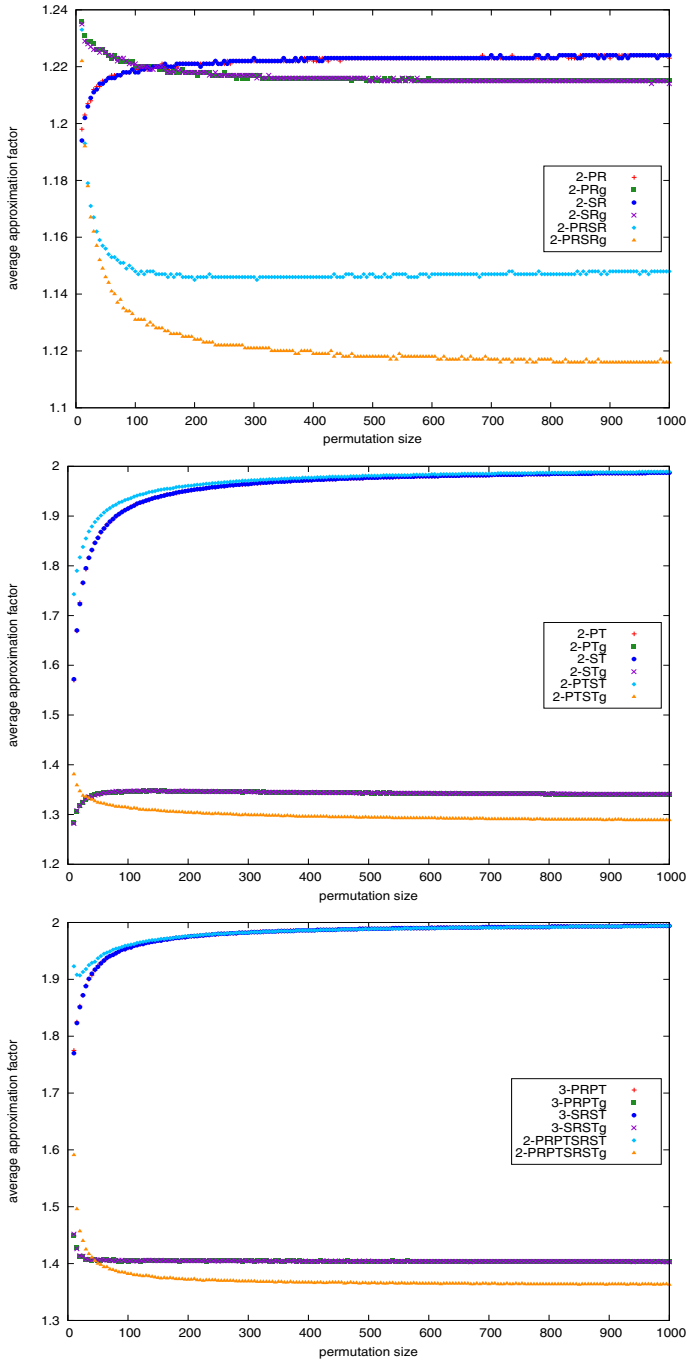

Fig. 2. Average approximation factor of all implemented algorithms when the permutation size grows

5 Conclusion

We introduced the study of suffix rearrangements along with prefix rearrangements. We showed lower bounds for the distances and described approximation algorithms of factor 2 to three new problems, considering some existing algorithms. Simple considerations, such as bigger operations and greedy choices, proved to be better options and improved the first versions of the algorithms. Future work will be directed not only to create new algorithms, but also to find results related to both distance and diameter of the problems.

Acknowledgements. This work was partially supported by São Paulo Research Foundation - FAPESP (grants 2013/01172-0 and 2013/08293-7) and National Counsel of Technological and Scientific Development - CNPq (grants 477692/2012-5 and 483370/2013-4). We thank Espaço da Escrita - Coordenadoria Geral da Universidade - UNICAMP - for the language services provided.

References

1. Bafna, V., Pevzner, P.A.: Genome Rearrangements and Sorting by Reversals. In: Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS 1993), pp. 148–157 (1993)
2. Bafna, V., Pevzner, P.A.: Sorting by Transpositions. SIAM Journal on Discrete Mathematics 11(2), 224–240 (1998)
3. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation Algorithm for Sorting by Reversals. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 200–210. Springer, Heidelberg (2002)
4. Bulteau, L., Fertin, G., Rusu, I.: Pancake flipping is hard. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 247–258. Springer, Heidelberg (2012)
5. Bulteau, L., Fertin, G., Rusu, I.: Sorting by Transpositions Is Difficult. SIAM Journal on Computing 26(3), 1148–1180 (2012)
6. Caprara, A.: Sorting Permutations by Reversals and Eulerian Cycle Decompositions. SIAM Journal on Discrete Mathematics 12(1), 91–110 (1999)
7. Dias, Z., Meidanis, J.: Sorting by Prefix Transpositions. In: Laender, A.H.F., Oliveira, A.L. (eds.) SPIRE 2002. LNCS, vol. 2476, pp. 65–76. Springer, Heidelberg (2002)
8. Elias, I., Hartman, T.: A 1.375-Approximation Algorithm for Sorting by Transpositions. IEEE/ACM Transactions on Computational Biology and Bioinformatics 3(4), 369–379 (2006)
9. Fischer, J., Ginzinger, S.W.: A 2-Approximation Algorithm for Sorting by Prefix Reversals. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 415–425. Springer, Heidelberg (2005)
10. Galvão, G.R., Dias, Z.: On the performance of sorting permutations by prefix operations. In: Proceedings of the 4th International Conference on Bioinformatics and Computational Biology (BICoB 2012), Las Vegas, Nevada, USA, pp. 102–107 (2012)
11. Sharmin, M., Yeasmin, R., Hasan, M., Rahman, A., Rahman, M.S.: Pancake Flipping with Two Spatulas. Electronic Notes in Discrete Mathematics 36, 231–238 (2010), International Symposium on Combinatorial Optimization (ISCO 2010)