JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 22, Number 11, 2015

Pp. 1044–1056
DOI: 10.1089/cmb.2014.0298

A Faster 1.375-Approximation Algorithm for Sorting by Transpositions*

LUÍS FELIPE I. CUNHA,[1] LUIS ANTONIO B. KOWADA,[2]
RODRIGO DE A. HAUSEN,[3] CELINA M.H. DE FIGUEIREDO[1]

ABSTRACT

Sorting by Transpositions is an NP-hard problem for which several polynomial-time approximation algorithms have been developed. Hartman and Shamir (2006) developed a 1.5-approximation $O(n^{\frac{3}{2}}\sqrt{\log n})$ algorithm, whose running time was improved to $O(n \log n)$ by Feng and Zhu (2007) with a data structure they defined, the permutation tree. Elias and Hartman (2006) developed a 1.375-approximation $O(n^2)$ algorithm, and Firoz et al. (2011) claimed an improvement to the running time, from $O(n^2)$ to $O(n \log n)$, by using the permutation tree. We provide counter-examples to the correctness of Firoz et al.'s strategy, showing that it is not possible to reach a component by sufficient extensions using the method proposed by them. In addition, we propose a 1.375-approximation algorithm, modifying Elias and Hartman's approach with the use of permutation trees and achieving $O(n \log n)$ time.

Key words: approximation algorithms, genome rearrangement, sorting by transpositions.

1 INTRODUCTION

In the study of genome rearrangements, chromosomes are commonly modeled by permutations (Fertin et al., 2009). In the Sorting by Transpositions (SBT) problem, the aim is to find the minimum number of contiguous block interchanges that transforms a given permutation of n elements into the identity permutation; this minimum is called the *transposition distance* (Bafna and Pevzner, 1998). SBT is an NP-hard problem (Bulteau et al., 2012), and tight bounds on the transposition distance are known (Bafna and Pevzner, 1998; Labarre, 2006), but exact values for the transposition distance are known only for a few classes of permutations (Cunha et al., 2013a; Labarre, 2006). Several approaches to handling the SBT problem have been considered. Our focus is to explore approximation algorithms for estimating the transposition distance between permutations, providing better practical results or lowering time complexities.

Bafna and Pevzner (1998) designed a 1.5-approximation $O(n^2)$ algorithm, where n is the length of the input permutation, based on the cycle structure of the *breakpoint graph*. Hartman and Shamir (2006) later

Extended abstracts appeared in Cunha et al. (2013b, 2014b).

[1]COPPE – Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil.
[2]Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, Brazil.
[3]Centro de Matemática, Computação e Cognição, Universidade Federal do ABC, São Paulo, Brazil.

proposed an easier 1.5-approximation algorithm that exploits a balanced tree data structure to decrease the running time to $O(n^{\frac{3}{2}}\sqrt{\log n})$. Feng and Zhu (2007) developed another balanced tree data structure—the *permutation tree*—and further decreased the complexity of Hartman and Shamir's 1.5-approximation algorithm to $O(n\log n)$. Elias and Hartman (2006) obtained, by a thorough computational case analysis of cycles of the breakpoint graph, a 1.375-approximation algorithm that runs in $O(n^2)$ time. Firoz et al. (2011) claimed that this 1.375-approximation algorithm could be easily adapted to run in $O(n\log n)$ time if a permutation tree was used.

In the present article, we show that Firoz et al.'s usage of the so-called ''Query'' procedure to extend a full configuration into a component fails in some situations. We provide an infinite family of permutations for which Firoz et al.'s approach does not find an 11/8-sequence, proving that the immediate use of a permutation tree is not enough to lower the running time of the 1.375-approximation algorithm to $O(n\log n)$. We rectify the use of the permutation tree, proposing a new algorithm that generalizes the bad small component strategy of Elias and Hartman toward *bad small full configurations*, achieving both the 1.375 approximation ratio and the $O(n\log n)$ time complexity. We thus achieve the best approximation ratio and time complexity for the SBT problem, known so far. The correctness of our algorithm is asserted via a branch-and-bound analysis that finds an 11/8-sequence for every combination of bad small full configurations.

The present article is organized as follows: section 2 contains basic definitions, some background on Elias and Hartman's algorithm and on the permutation tree data structure; section 3 discusses Firoz et al.'s approach on the use of a permutation tree to speed up the 1.375-approximation algorithm and provides counterexamples that establish the incorrectness of their approach; section 4 presents a strategy to determine the existence and to find a sequence of two transpositions, in which both are 2-moves, in linear time; section 5 describes our proposed 1.375-approximation $O(n\log n)$ algorithm for SBT, proving its correctness and its worst-case running time; and section 6 contains our final remarks.

2 BACKGROUND

For our purposes, a gene is represented by a unique integer and a chromosome with n genes is a *permutation* $\pi = [\boldsymbol{\pi_0}\pi_1\pi_2 \ldots \pi_n\boldsymbol{\pi_{n+1}}]$, where $\boldsymbol{\pi_0} = 0, \boldsymbol{\pi_{n+1}} = n+1$ and each π_i, where $1 \le i \le n$, is a unique integer in the range $1, \ldots, n$. The *transposition* $t(i, j, k)$ applied to π, where $1 \le i < j < k \le n+1$, is the permutation $\pi \cdot t(i, j, k)$ in which the two contiguous blocks $\pi_i\, \pi_{i+1} \ldots \pi_{j-1}$ and $\pi_j\, \pi_{j+1} \ldots \pi_{k-1}$ are interchanged. A sequence of q transpositions $t_1, t_2, \ldots, t_q$ *sorts* a permutation π if $\pi \cdot t_1 \cdot t_2 \ldots . t_q = \iota$, where ι is the identity permutation $[\mathbf{0}\ 1\ 2 \ldots n\ \boldsymbol{n+1}]$. The *transposition distance* of π, denoted $d(\pi)$, is the length of a minimum sequence of transpositions that sorts π.

The breakpoint graph Nontrivial bounds on the transposition distance were obtained by using the breakpoint graph (Bafna and Pevzner, 1998). Given a permutation π, the *breakpoint graph* of π is $G(\pi) = (V, R \cup D)$. The set of vertices is $V = \{0, -1, +1, -2, +2, \ldots, -n, +n, -(n+1)\}$, and the set of edges is partitioned into two subsets, the directed *reality edges* $R = \{\overrightarrow{i} = (+\pi_i, -\pi_{i+1}) | i = 0, \ldots, n\}$ and the undirected *desire edges* $D = \{(+i, -(i+1)) \mid i = 0, \ldots, n\}$. Figure 1 shows $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$, where the arrows represent the directed edges in R and the arcs represent the undirected edges in D.

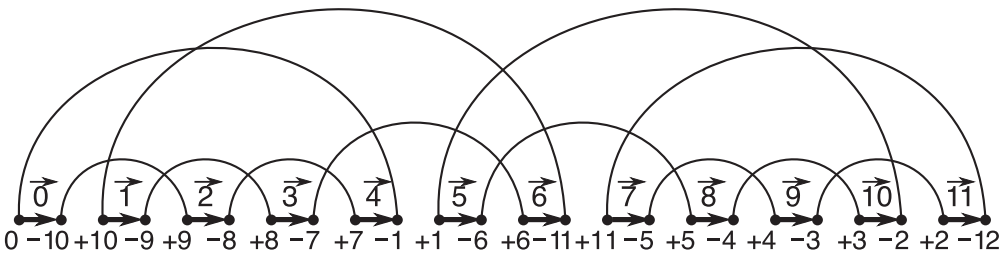

FIG. 1. $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$. The set of cycles is $\{C_1 = \langle 024\rangle, C_2 = \langle 136\rangle, C_3 = \langle 5810\rangle, C_4 = \langle 7911\rangle\}$. The cycles C_2 and C_3 intersect, but are not interleaving; the cycles C_1 and C_2 are interleaving, and so are C_3 and C_4. The cycle C_1 is the leftmost cycle.

Since every vertex in $G(\pi)$ has degree 2, the graph can be partitioned into disjoint cycles. We shall use the terms *a cycle in* π and *a cycle in* $G(\pi)$ interchangeably to denote the latter. A cycle in π has length ℓ (or it is an ℓ-*cycle*), if it has exactly ℓ reality edges. A permutation π is a *simple permutation* if every cycle in π has length at most 3, as the example in Figure 1.

After applying a transposition t, the number of cycles of odd length in $G(\pi)$, denoted $c_{odd}(\pi)$, changes in such a way that $c_{odd}(\pi \cdot t) = c_{odd}(\pi) + x$, where $x \in \{-2, 0, 2\}$; the transposition t is thus classified as an *x-move for* π. Since $c_{odd}(\iota) = n+1$, we have the lower bound $d(\pi) \geq \left[\frac{(n+1)-c_{odd}(\pi)}{2}\right]$, where the equality holds if, and only if, π can be sorted solely with 2-moves.

Hannenhalli and Pevzner (1999) proved that every permutation π can be transformed, in $O(n)$ time, into a simple one $\hat{\pi}$, by inserting new elements in appropriate positions of π, preserving the lower bound for the distance, $\left[\frac{(n+1)-c_{odd}(\pi)}{2}\right] = \left[\frac{(m+1)-c_{odd}(\hat{\pi})}{2}\right]$, where m is such that $\hat{\pi} = [\mathbf{0}\hat{\pi_1}...\hat{\pi_m}\mathbf{m+1}]$. Additionally, in a sequence that sorts $\hat{\pi}$, every transposition can be transformed, in $O(\log n)$ time (Feng and Zhu, 2007), into a sequence with the same number of transpositions that sorts π, which implies that $d(\pi) \leq d(\hat{\pi})$. The approach of finding a sorting sequence for π via a simple permutation $\hat{\pi}$ is commonly used in approximation algorithms for SBT (Elias and Hartman, 2006; Hartman and Shamir, 2006).

A transposition t (i, j, k) *affects* a cycle C if it contains one of the following reality edges: $\overrightarrow{i+1}$, or $\overrightarrow{j+1}$, or $\overrightarrow{k+1}$. A cycle is *oriented* if there is a 2-move that affects it, otherwise it is *unoriented* (these names come from the order of such triplet of reality edges in the breakpoint graph). If π contains an oriented cycle then π is *oriented*, otherwise π is *unoriented*.

A sequence of q transpositions of which exactly r transpositions are 2-moves is a (q,r)-sequence. A q/r-sequence is an (x,y)-sequence such that $x \leq q$ and $x/y \leq q/r$.

Interactions between cycles A cycle in π can be uniquely identified by its reality edges, in the order that they appear, starting from the leftmost edge. The notation $C = \langle x_1 x_2 \ldots x_\ell \rangle$, where $\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_\ell}$ are reality edges, and $x_1 = \min\{x_1, x_2, \ldots, x_\ell\}$, characterizes an ℓ-cycle. The *leftmost cycle* is the cycle that contains the edge $\overrightarrow{0}$.

Let $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$, where $x < y < z$, be reality edges in a cycle C, and $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$, where $a < b < c$ be reality edges in a different cycle C'. The pair of reality edges $\overrightarrow{x}, \overrightarrow{y}$, *intersects* the pair $\overrightarrow{a}, \overrightarrow{b}$, if these four edges occur in an alternating order in the breakpoint graph, that is, either $x < a < y < b$ or $a < x < b < y$, and we say C and C' *intersect*. Similarly, a triplet of reality edges $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$ *interleaves* a triplet $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$ if these six edges occur in an alternating order: $x < a < y < b < z < c$ or $a < x < b < y < c < z$. Two 3-cycles *interleave* if their respective triplets of reality edges interleave. Figure 1 also illustrates these concepts.

A *configuration* of π is a subset of the cycles in $G(\pi)$. A configuration $\mathscr{C}$ is *connected* if there is a sequence of $C_1, \ldots, C_k$ in $\mathscr{C}$ such that $C_1 = C$, $C_k = C'$ and for each $i \in \{1, 2, \ldots, k-1\}$, the cycles C_i and C_{i+1} are intersecting. If the configuration $\mathscr{C}$ is connected and maximal, then $\mathscr{C}$ is a *component*. Every permutation admits a unique decomposition into disjoint components. For instance, in Figure 1, the configuration $\{C_1, C_2, C_3, C_4\}$ is a component, but the configuration $\{C_1, C_2, C_3\}$ is connected but not a component.

Let C be a 3-cycle in configuration $\mathscr{C}$. An *open gate* is a pair of reality edges in C that does not intersect any other pair of reality edges in $\mathscr{C}$. If a configuration $\mathscr{C}$ has only 3-cycles and no open gates, then $\mathscr{C}$ is a *full configuration*. Some full configurations do not correspond to the breakpoint graph of any permutation. A full configuration corresponds to a permutation if, and only if, the *complement configuration* is Hamiltonian (Elias and Hartman, 2006), as illustrated in Figure 2b. Figure 2a shows the full configuration $F = \{\langle 0\,7\,9\rangle, \langle 1\,3\,6\rangle, \langle 2\,4\,11\rangle, \langle 5\,8\,11\rangle\}$, which does not correspond to any permutation but is important in the analysis of our algorithm and will be studied in detail in section 5.

A configuration $\mathscr{C}$ that has k edges is in the *cromulent form*[4] if all edges $\overrightarrow{0}, \overrightarrow{1}, \ldots, \overrightarrow{k-1}$ are in $\mathscr{C}$. Given a configuration $\mathscr{C}$ having k edges, a *cromulent relabeling* of $\mathscr{C}$ is a configuration $\mathscr{C}$ such that $\mathscr{C}'$ is in the cromulent form and there is a function σ satisfying that, for every pair of edges $\overrightarrow{i}, \overrightarrow{j}$, in $\mathscr{C}$ such that $i < j$,

[4]*Cromulent* is neologism coined by David X. Cohen, meaning ''normal'' or ''acceptable.''

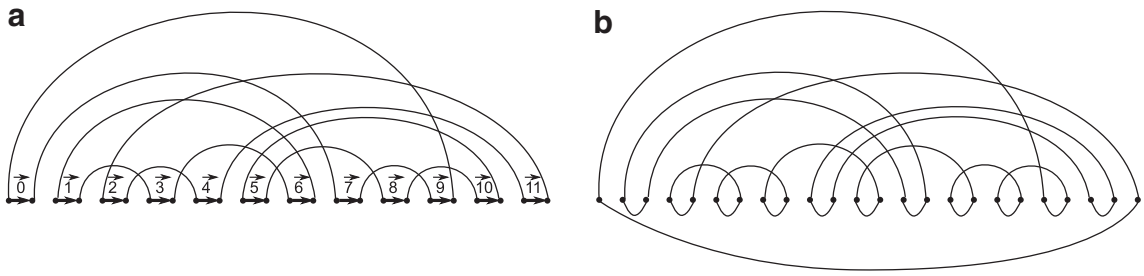

FIG. 2. **(a)** Full configuration $F=\{\langle 079\rangle, \langle 136\rangle, \langle 2411\rangle, \langle 5810\rangle\}$, which does not correspond to a permutation. **(b)** Complement of F, obtained by replacing the reality edges with the edges $-\pi_i\pi_i$ and $0\text{-}\pi_{n+1}$.

we have that $\overrightarrow{\sigma(i)}, \overrightarrow{\sigma(j)}$, are in $\mathscr{C}'$ and $\sigma(i)<\sigma(j)$. For instance, in Figure 2a the cromulent relabeling of the configuration $\{C_1=\langle 0, 7, 9\rangle, C_2=\langle 1, 3, 6\rangle\}$ is $\{C_1=\langle 0, 4, 5\rangle, C_2=\langle 1, 2, 3\rangle\}$.

Given an integer x, a *circular shift* of a configuration $\mathscr{C}$, which is in the cromulent form and has k edges, is a configuration denoted $\mathscr{C}+x$ such that every edge $\overrightarrow{i}$ in $\mathscr{C}$ corresponds to $\overrightarrow{i+x(\mathrm{mod}\ k)}$ in $\mathscr{C}+x$. Two configurations $\mathscr{C}$ and $\mathscr{K}$ are *equivalent* if there is an integer x such that $\mathscr{C}'+x=\mathscr{K}'$, where $\mathscr{C}'$ and $\mathscr{K}'$ are their respective cromulent relabelings.

Elias and Hartman's algorithm Elias and Hartman (2006) performed a systematic enumeration of all components with nine cycles or less, in which all cycles have length 3. Starting from single 3-cycles, components were obtained by applying a series of *sufficient extensions*, as described next. An *extension* of a configuration $\mathscr{C}$ is a connected configuration $\mathscr{C}\cup\{C\}$, where $C\notin\mathscr{C}$. A *sufficient extension* is an extension that either: 1) closes an open gate; or 2) extends a full configuration such that the extension has at most one open gate. A configuration obtained by a series of sufficient extensions is a *sufficient configuration* if an *(x,y)*-, or an *x/y*-sequence can be applied to its cycles.

Lemma 1 *(Elias and Hartman, 2006) Every unoriented sufficient configuration of nine cycles has an* 11/8*-sequence.*

Components with less than nine cycles are called *small components*. Elias and Hartman have shown that, of all small components, only five types of them do not have an 11/8-sequence; these components are called *bad small components*. Small components that have an 11/8-sequence are called *good small components*.

Lemma 2 *(Elias and Hartman, 2006) The bad small components are:*

- $A=\{\langle 0\,2\,4\rangle, \langle 1\,3\,5\rangle\}$**;**
- $B=\{\langle 0\,2\,10\rangle, \langle 1\,3\,5\rangle, \langle 4\,6\,8\rangle, \langle 7\,9\,11\rangle\}$*;*
- $C=\{\langle 0\,5\,7\rangle, \langle 1\,9\,11\rangle, \langle 2\,4\,6\rangle, \langle 3\,8\,10\rangle\}$*;*
- $D=\{\langle 0\,2\,4\rangle, \langle 1\,12\,14\rangle, \langle 3\,5\,7\rangle, \langle 6\,8\,10\rangle$**,** $\langle 9\,11\,13\rangle\}$*; and*
- $E=\{\langle 0\,2\,16\rangle$**,** $\langle 1\,3\,5\rangle, \langle 4\,6\,8\rangle, \langle 7\,9\,11\rangle, \langle 10\,12\,14\rangle, \langle 13\,15\,17\rangle\}$.

If a permutation has bad small components, it is still possible to find an (11,8)-sequence, as Lemma 3 states.

Lemma 3 *(Elias and Hartman, 2006) Let* π *be a permutation with at least eight cycles and containing only bad small components. Then* π *has an* (11,8)*-sequence.*

Corollary 1 *(Elias and Hartman, 2006) If every cycle in* $G(\pi)$ *is a* 3*-cycle, and there are at least eight cycles, then* π *has an* 11/8*-sequence.*

Lemmas 1 and 3 and Corollary 1 form the theoretical basis for Elias and Hartman's $11/8=1.375$-approximation algorithm for SBT, Algorithm 1. The main procedure of Algorithm 1 is: obtain extensions, if a configuration with nine cycles or if a small good component is reached, then an 11/8-sequence is applied (Lemma 1); if a bad small component is reached then no sequence is applied. After all configurations with nine cycles or small good components have been sorted, the permutation just contains small bad components; Lemma 3 states the existence of an (11,8)-sequence.

Algorithm 1: Elias and Hartman's Sort(π)

```
Transform permutation π into a simple permutation π̂.
Check if there is a (2,2)-sequence. If so, apply it.
While G(π̂) contains a 2-cycle, apply a 2-move.
π̂ consists of 3-cycles. Mark all 3-cycles in G(π̂).
while G(π̂) contains a marked 3-cycle C do
    if C is oriented then
        Apply a 2-move.
    else
        Try to sufficiently extend C eight times (to obtain a configuration with at most nine cycles).
        if sufficient configuration with nine cycles has been achieved then
            Apply an 11/8-sequence.
        else It is a small component
            if it is a good component then
                Apply an 11/8-sequence.
            else
                Unmark all cycles of the component.
(Now G(π̂) has only bad small components.)
while G(π̂) contains at least eight cycles do
    Apply an (11,8)-sequence
While G(π̂) contains a 3-cycle, apply a (3,2)-sequence.
Mimic the sorting of π using the sorting of π̂.
```

Feng and Zhu's permutation tree Feng and Zhu (2007) introduced the *permutation tree*, a binary balanced tree that represents a permutation. In logarithmic time the operation of applying a transposition could be done, and also the *Query* procedure of finding a pair of reality edges that intersects another given pair of reality edges, as well. The *Query* procedure is the method used in Hartman and Shamir's (2006) 1.5-approximation algorithm to find a (3,2)-sequence that affects a pair of intersecting or interleaving cycles. Besides that, Firoz et al. (2011) claimed the *Query* procedure could be used to sufficiently extend a configuration in Algorithm 1.

Let $\pi = [\boldsymbol{\pi_0}\pi_1\pi_2 \ldots \pi_n\boldsymbol{\pi_{n+1}}]$ be a permutation. The corresponding permutation tree has n leaves, labeled $\pi_1, \pi_2, \ldots, \pi_n$; every internal node represents an interval of consecutive elements $\pi_i, \pi_{i+1}, \ldots, \pi_{k-1}$, with $i<k$, and is labeled by the maximum number in the interval. Therefore, the root of the tree is labeled with n. Furthermore, the left child of a node represents the interval $\pi_i, \ldots, \pi_{j-1}$, and the right child represents $\pi_j, \ldots, \pi_k$, with $i<j<k$. Feng and Zhu provided algorithms: to *build* a permutation tree in $O(n)$ time; to *join* two permutation trees into one in $O(h)$ time, where h is the height difference between the trees; and to *split* a permutation tree into two in $O(\log n)$ time.

The operations *split* and *join* allow us to apply a transposition to a permutation π, updating the tree, in time $O(\log n)$. Based on Lemma 4, the *Query* procedure (Algorithm 2) solves the problem of finding a pair of reality edges intersecting another given pair of reality edges.

Lemma 4 *(Bafna and Pevzner, 1998) Let $\overrightarrow{i}$ and $\overrightarrow{j}$ be two reality edges in an unoriented cycle C, $i<j$. Let $\pi_k = max_{i<m\leq j}\pi_m$, $\pi_\ell = \pi_k + 1$. Then, the reality edges $\overrightarrow{k}$ and $\overrightarrow{\ell-1}$ belong to the same cycle, and the pair $\overrightarrow{k}, \overrightarrow{\ell-1}$ intersects pair $\overrightarrow{i}, \overrightarrow{j}$.*

Algorithm 2: *Query(π,i,j)*

```
input: permutation π, integers i and j
Let T be the permutation tree of π
Split T, into three permutation trees, T1, T2 and T3, corresponding to [π0, π1, ..., πi], [πi+1, ..., πj],
    and [πj+1, ..., πn, πn+1], respectively.
Let πk = root (T2). (the largest element in the interval πi+1, ..., πJ)
Let πℓ = πk + 1
Return the pair k, ℓ − 1 (by Lemma 4, k→, l−1→ intersects, i→, J→)
```

Firoz et al. (2011) suggested the use of the permutation tree data structure to reduce the running time of Algorithm 1 to $O(n\log n)$, but in section 3 we show that the strategy, in the manner proposed by Firoz et al., fails to extend some 3-cycles into a full configuration with nine cycles.

3. THE USE OF THE PERMUTATION TREE BY FIROZ ET AL.

Firoz et al. (2011) stated that Step 9 in Algorithm 1 could be done in $O(\log n)$ time. To do so, they categorized the sufficient extensions of a configuration A obtained by a *Query* call into *type 1 extensions*—those that add a cycle that closes an open gate—and *type 2 extensions*—those that extend a full configuration by adding a cycle C such that $A \cup \{C\}$ has at most one open gate.

A type 1 extension can be performed in logarithmic time with $Query(\pi, i, j)$, where $\overrightarrow{i}$, $\overrightarrow{j}$ form an open gate. For a type 2 extension, since there are no open gates, Firoz et al. claimed that it would be sufficient to perform queries with every pair of reality edges that belonged to the same cycle in the configuration that is being extended. Example 1 shows that this strategy is flawed.

Example 1 *Consider the permutation* $\pi = [\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}]$, *whose breakpoint graph is depicted in Figure 1. It is a simple permutation having only unoriented 3-cycles. Mark all the cycles* $C_1 = \langle 0,2,4\rangle$, $C_2 = \langle 1, 3, 6\rangle$, $C_3 = \langle 5,8,10\rangle$, *and* $C_4 = \langle 7,9,11\rangle$. *Let* $A = \{C_1\}$ *be the configuration to be sufficiently extended (step 9 in Algorithm 1). Using the method proposed by Firoz et al., we have that:*

1. *Configuration A has three open gates* $\overrightarrow{0}$, $\overrightarrow{2}; \overrightarrow{2}$, $\overrightarrow{4}; \overrightarrow{4}$, $\overrightarrow{0}$. *Execute* $Query(\pi, 0, 2)$, *which returns the pair* $\overrightarrow{1}$, $\overrightarrow{6}$, *in cycle* $C_2 = \langle 1, 3, 6\rangle$ *(or, alternatively* $Query(\pi, 2, 4)$, *which yields the same result). Therefore,* C_2 *is added to the configuration A, which becomes* $A = \{C_1, C_2\}$.
2. *Configuration A has no more open gates. We must execute* $Query(\pi,i,j)$ *for every pair of elements* $\overrightarrow{i}$, $\overrightarrow{j}$ *in the same cycle of the configuration such that* $i<j$; *it is easy to observe that each execution returns a pair that is already in A. So far, Firoz et al.'s method has failed to extend A.*
3. *Since A is not a component, unmark all the cycles in A.*
4. *The marked cycles are now* C_3 *and* C_4. *Considering either* $A = \{C_3\}$ *or* $A = \{C_4\}$, *Firoz et al.'s method only extends A as far as* $\{C_3, C_4\}$. *Again, A is not a component.*

Therefore, Firoz et al.'s method fails to find the component $\{C_1, C_2, C_3, C_4\}$.

Although the permutation in Example 1 has only one small component, it is a counterexample to the correctness of Firoz et al.'s strategy for dealing with type 2 extensions. The same problem happens for sufficient configurations with more than nine cycles, such as:

$$\sigma = [\mathbf{0}\ 25\ 24\ 23\ 22\ 1\ 21\ 26\ 20\ 19\ 18\ 2\ 17\ 27\ 16\ 15\ 14\ 3\ 13\ 28\ 12\ 11\ 10\ 4\ 9\ 29\ 8\ 7\ 6\ 5\ \mathbf{30}].$$

By Lemma 1, every configuration of nine cycles has an 11/8-sequence. Figure 3 shows an example of a breakpoint graph of σ with 10 cycles, for which any configuration with 9 cycles has an 11/8-sequence. However, Firoz et al.'s approach fails to find such a sequence, for it performs the following sequence of operations: i) starting from any configuration having a unique cycle, the first call to *Query* correctly finds another intersecting cycle, which is added to the configuration (Fig. 4); ii) with this configuration having two interleaving cycles, every possible invocation of *Query* returns one of the cycles already in the configuration, which means that their strategy cannot further extend the configuration; iii) the resulting configuration, with only two cycles, is a bad small component, so the cycles are unmarked; iv) if an unmarked cycle still remains, it is selected to start a configuration, and we return to the first step in this sequence of operations. The procedure finishes after unmarking all 10 cycles, incorrectly presuming that the breakpoint graph only has bad small components with two cycles.

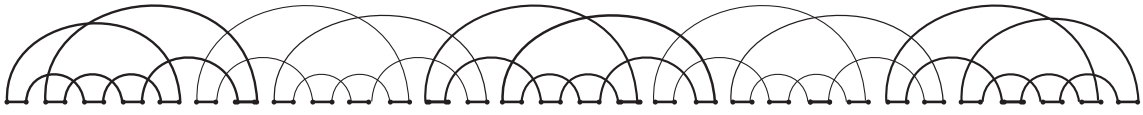

FIG. 3. Breakpoint graph of a permutation σ for which Firoz et al.'s method fails. Note that σ has 10 cycles, and that σ is obtained by setting $k = 5$ in Equation (2).

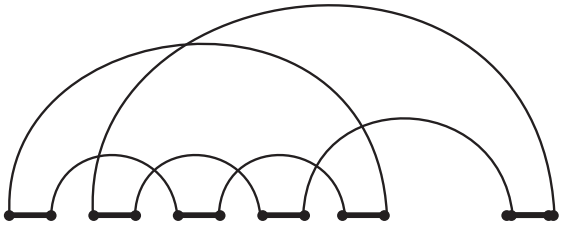

FIG. 4. A configuration that is not maximal returned by a *Query* call on σ.

Notice that, according to Step 18 in Algorithm 1, if a permutation contains only bad small components, then an 11/8-sequence can be applied. The permutation γ (whose breakpoint graph is in Fig. 5) is one of those permutations:

$$\gamma = [\mathbf{0}\ 5\ 4\ 3\ 2\ 1\ 6\ 11\ 10\ 9\ 8\ 7\ 12\ 17\ 16\ 15\ 14\ 13\ 18\ 23\ 22\ 21\ 20\ 19\ 24\ 29\ 28\ 27\ 26\ 25\ \mathbf{30}],$$

Notice how the breakpoint graphs in Figures 4 and 5 differ, but according to Firoz et al.'s approach they would be indistinguishable. In Figure 5, the bad small components can be separately handled, which is not the case for configurations with two interleaving cycles in Figure 4, for these configurations intersect other cycles. Algorithm 1 does not have a rule to deal with this last case.

Note that the permutation of Example 1 and the permutation above σ just describe examples belonging to a family of permutations such that any type 2 extension fails. Actually, an infinite family can be constructed as follows: let k be any integer greater than or equal to 2, and let $f(i)$ be the sequence of six integers

$$i\ 5k-4i\ 5k+i\ 5k-4i-1\ 5k-4i-2\ 5k-4i-3. \tag{1}$$

Consider σ^k a permutation of $6k-1$ elements defined using Equation (1) as:

$$[\mathbf{0}\ 5k\ 5k-1\ 5k-2\ 5k-3\ f(1)\ f(2)\ldots\ f(k-1)\ k\ \mathbf{6k}], \tag{2}$$

whose breakpoint graph has a similar structure to those in Figure 1 (where in Equation (2) we set $k=2$) and 3. If we start from a configuration having any cycle, it is impossible to extend it past a configuration of more than two cycles using Firoz et al.'s approach.

Some other configurations cannot be extended using only the *Query* procedure either, such as the full configuration illustrated in Figure 2. This is a bad small configuration (Elias and Hartman, 2006) that does not correspond to the breakpoint graph of any permutation, but this configuration may appear during the sorting of a larger permutation.

4. FINDING AND APPLYING A (2,2)-SEQUENCE IN LINEAR TIME

In order to implement Step 2 of Algorithm 1, Elias and Hartman (2006) proved that, given a simple permutation, a (2,2)-sequence can be found in $O(n^2)$ time.

Firoz et al. (2011) described a strategy for finding and applying a (2,2)-sequence in $O(n\log n)$ time using permutation trees. But, according to their strategy, for each one of the $O(n)$ oriented 3-cycles, we apply a 2-move and check in $O(n)$ time for the existence of an oriented cycle in the resulting graph, which implies that Firoz et al.'s strategy may run in $\Omega(n^2)$ time in the worst case. One such case is illustrated in Figure 6, where the cycles drawn in solid lines—one oriented, the other unoriented—are interleaving, so by case 3 of Lemma 5 there is a (2,2)-sequence that affects them. However, if the first 2-move is applied to any of the

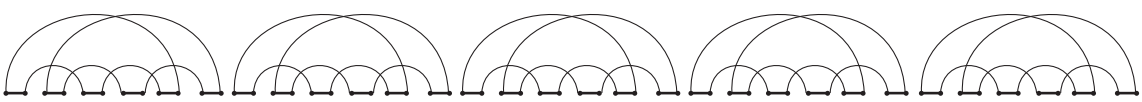

FIG. 5. The breakpoint graph of permutation γ.

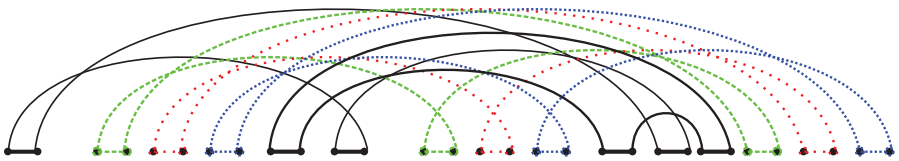

FIG. 6. A breakpoint graph for which Firoz et al.'s strategy takes $\Omega(n^2)$ time in the worst case to find a (2,2)-sequence.

dashed cycles, the resulting breakpoint graph has no oriented cycle, hence the transposition is undone and another oriented cycle is selected; this procedure would continue until the solid oriented cycle is selected.

Algorithm 5 summarizes our proposed approach toward finding and applying a (2,2)-sequence in $O(n)$ time. It is a direct application of Algorithms 3 and 4 to the cases stated in Lemma 5.

Algorithm 3: Search (2,2)-sequence from K_1.

```
1  for i = min K1 + 1, ... ,mid K1 − 1 do
2      if →i belongs to an oriented cycle Kj then
3          if mid Kj < mid K1 then
4              return (2,2)-sequence that affects K1 and Kj.
5          else if max Kj < max K1 then
6              return (2,2)-sequence that affects K1 and Kj.
7          else if max K1 < mid Kj then
8              return (2,2)-sequence that affects K1 and Kj.
9      if →i belongs to an unoriented cycle Lj then
10         if mid K1 < mid Lj < max K1 < max Lj then
11             return (2,2)-sequence that affects K1 and Lj
12         else if min Lj < min K1 < mid Lj < mid K1 < max Lj < max K1 then
13             return (2,2)-sequence that affects K1 and Lj.
14 for i = mid K1 + 1, ... ,max K1 − 1 do
15     if →i belongs to an oriented cycle Kj then
16         if mid K1 < min Kj then
17             return (2,2)-sequence that affects K1 and Kj.
18 for i = max K1 + 1, ... , n − 1 do
19     if →i belongs to an oriented cycle Kj then
20         if max K1 < min Kj then
21             return (2,2)-sequence affecting K1 and Kj.
```

Lemma 5 *(Bafna and Pevzner, 1998; Christie, 1999; Elias and Hartman, 2006) Given a breakpoint graph of a simple permutation, there exists a* (2,2)*-sequence if any of the following conditions is met:*

1. *There are either four* 2*-cycles, or two intersecting* 2*-cycles, or two nonintersecting* 2*-cycles, and the resulting graph contains an oriented cycle after the first transposition is applied;*
2. *There are two noninterleaving oriented* 3*-cycles;*
3. *There is an oriented cycle interleaving an unoriented cycle.*

Our strategy to find a (2,2)-sequence in linear time starts by checking whether the breakpoint graph satisfies first case of Lemma 5, as described in detail between lines 1 and 4 in Algorithm 5. In our approach, it is unnecessary to try all pairs of cycles to verify that conditions 2 and 3 in Lemma 5 are satisfied. It differs from previous methods (Elias and Hartman, 2006; Firoz et al., 2011) in that the leftmost oriented cycle of the breakpoint graph, named K_1, is fixed when verifying for conditions 2 and 3.

Given a simple permutation π, it is immediate to enumerate all of its cycles in linear time. The size of each cycle, and whether it is oriented, are both determined in constant time.

Christie (1999) proved that every permutation has an even number (possibly zero) of even cycles; he also showed that, given a simple permutation, when the number of even cycles is not zero, there exists a (2,2)-

FIG. 7. Oriented cycles represented by their reality edges. All oriented cycles interleave K_1, but there are i and j such that K_i and K_j are noninterleaving.

sequence that affects those cycles if, and only if, there are either four 2-cycles, or there are two intersecting even cycles. Therefore, in these cases, a (2,2)-sequence can be applied in $O(\log n)$ using permutation trees. If there is only a pair of nonintersecting 2-cycles, it remains to check if there is a 3-cycle intersecting both even cycles: i) if the 3-cycle is oriented, then first we apply the 2-move applied to the 3-cycle, and the second 2-move is applied to 2-cycles; ii) if the 3-cycle is unoriented, then first we apply the 2-move applied to the 2-cycles, and the second 2-move is applied to the 3-cycle, which turns oriented after the first transposition. There is also a (2,2)-sequence if there is an oriented cycle intersecting at most one even cycle.

However, if there are no even cycles in the permutation, but there is an oriented cycle, the 3-cycles must be scanned for the existence of a (2,2)-sequence, as conditions 2 and 3 require in Lemma 5.

Algorithm 4: Finding intersecting oriented cycles interleaving K_1.

```
1  s1 = sequence of edges belonging to oriented cycles from left to right between min K1 and mid K1.
2  s2 = sequence of edges belonging to oriented cycles from left to right between mid K1 and max K1.
3  if the sequences of cycles corresponding to s1 and s2 are different then
4      There is a pair of intersecting oriented cycles, exists a (2,2)-sequence.
5  else
6      All oriented cycles are mutually interleaving.
```

To check, in linear time, for the existence of a pair of cycles satisfying either condition 2 or 3 in Lemma 5, consider the oriented cycles of the breakpoint graph, in the order $K_1 = \langle a_1\ b_1\ c_1\rangle$, $K_2 = \langle a_2\ b_2\ c_2\rangle$, $K_3 = \langle a_3\ b_3\ c_3\rangle$, … such that $a_1 < a_2 < a_3 < \ldots$, and the unoriented cycles in the order $L_1 = \langle x_1\ y_1\ z_1\rangle$, $L_2 = \langle x_2\ y_2\ z_2\rangle$, $L_3 = \langle x_3\ y_3\ z_3\rangle$, … such that $x_1 < x_2 < x_3 < \ldots$. Given any 3-cycle $C = \langle abc\rangle$, let min $C = a$, mid $C = \min\{b,c\}$, and max $C = \max\{b,c\}$, that is, if C is unoriented, then min $C = a$, mid $C = b$, max $C = c$, whereas if C is oriented, then min $C = a$, mid $C = c$, max $C = b$. The main idea is:

1. Check for the existence of an oriented cycle K_j noninterleaving K_1 or an unoriented cycle L_j interleaving K_1. Algorithm 3 searches for an oriented cycle K_i noninterleaving K_1 or an unoriented cycle L_i interleaving K_1. The search is done between $minK_1$ and mid K_1, between mid K_1 and max K_1, and to the right of max K_1.
2. If Algorithm 3 does not return any oriented cycle noninterleaving K_1, then every oriented cycle interleaves K_1 but no unoriented cycle interleaves K_1. Hence, we must check for the existence of two oriented cycles K_i, K_j that are intersecting but not interleaving. Note that if K_i, K_j were nonintersecting oriented cycles, Algorithm 3 would have this case already covered (see Fig. 7), since K_i or K_j would not interleave K_1. Algorithm 4 describes how to verify the existence of two intersecting oriented cycles that are also interleaving with K_1.

Algorithm 5: Find and Apply (2,2)-sequence

```
1  if there are four 2-cycles then
2      Apply (2,2)-sequence.
3  else if there is a pair of intersecting 2-cycles then
4      Apply (2,2)-sequence.
5  else if there is a 3-cycle intersecting a pair of 2-cycles then
6      Apply (2,2)-sequence.
7  else if there is a pair of 2-cycles and an oriented 3-cycle intersecting at most one of them then
8      Apply (2,2)-sequence.
9  else if Search (2,2)-sequence from K1 returns a sequence then
10     Apply (2,2)-sequence.
11 else if Finding intersecting oriented cycles interleaving K1 then
12     Apply (2,2)-sequence.
13 else
14     There are no (2,2)-sequences to apply.
```

5. SUFFICIENT EXTENSIONS USING THE *QUERY* PROCEDURE

In section 3, we discussed Firoz et al.'s use of the permutation tree and proved that their strategy does not account for every configuration with less than nine cycles that is not a component, since successive invocations of *Query* may result in a full configuration with less than nine cycles that is not a small component. Our proposed strategy generalizes the definitions regarding small components to *small configurations*—configurations with less than nine cycles.

A small configuration is *full* if it has no open gates. Small configurations are also classified as *good* if they have an 11/8-sequence, or as *bad* otherwise.

Algorithm 1 applies an 11/8-sequence to every sufficient unoriented configuration of nine cycles, and also to every good small component. After that, the permutation contains just bad small components, and Lemma 3 states that there exists an (11,8)-sequence for every combination of bad small components with at least eight cycles.

Our approach can handle bad small full configurations, which may or may not be bad small components, during the course of an extension via successive invocations of *Query*. The possible bad small full configurations are the bad small components A, B, C, D, and E, from Lemma 2, and the full configuration $F = \{\langle 079\rangle, \langle 136\rangle, \langle 2411\rangle, \langle 5810\rangle\}$, the only bad small full configuration that is not a component (Elias and Hartman, 2006).

Our strategy (Algorithm 6) is similar to Elias and Hartman's (Algorithm 1): apply an 11/8-sequence to every sufficient unoriented configuration of nine cycles and also to every good small full configuration; the main difference is that, whenever a combination of bad small full configurations is found, a decision to apply an 11/8-sequence is made according to Lemmas 6 and 7.

We developed a tool (Cunha et al., 2014a) that finds 11/8-sequences for a given configuration using branch-and-bound, where a *branch* is obtained by either applying a 2-move or a 0-move, and the moves are *bounded* by the ratio between the number of total moves and the number of 2-moves, which cannot be greater than 1.375. The algorithm either returns an 11/8-sequence, whenever it exists, or fails after trying all possible sequences.

Lemma 6 *Every combination of F with one or more copies of either B, C, D, or E has an* 11/8*-sequence.*

Proof. Consider all breakpoint graphs of F and its circular shifts combined with B, C, D, E, and their circular shifts. A combination of a pair of small full configurations is obtained by starting from one small full configuration and inserting a new one in different positions in the breakpoint graph. Altogether, there are 324 such graphs. A computerized case analysis (Cunha et al., 2014a) enumerates all possible breakpoint graphs and provides an 11/8-sequence for each of them. ■

Notice that Lemma 6 considers neither combinations of F with F, nor combinations of F with A. We have found that almost every combination of F with F has an 11/8-sequence, as Lemma 7 states. Let F_iF^j be the configuration obtained by inserting the circular shift $F+j$ between the edges $\overrightarrow{i}$ and $\overrightarrow{i+1}$ of F.

Lemma 7 *There exists an* 11/8*-sequence for* F_iF^j*, if:*

- $i \in \{0,4\}$ *and* $j \in \{0,1,2,3,4,5\}$*;*
- $i \in \{1,2,3\}$ *and* $j \in \{1,2,3,4,5\}$*; or*
- $i = 5$ *and* $j \in \{1,5\}$.

Proof. The 11/8-sequences for the cases enumerated above were also found through a computerized case analysis (Cunha et al., 2014a). Note that F_iF^j is equivalent to $F_{i+6}F^j$ for $i = \{0, 1, \ldots, 5\}$, which simplifies our analysis. ■

Only seven combinations of F with F have no 11/8-sequence: F_1F^0, F_2F^0, F_3F^0, F_5F^0, F_5F^2, F_5F^3, and F_5F^4. We will return to them shortly.

All combinations of one copy of F and one copy of A have less than eight cycles. It only remains to analyze the combinations of F and two copies of A, denoted F–A–A. The *good F–A–A combinations* are the

F–A–A combinations for which an 11/8-sequence exists. Out of 57 combinations of *F–A–A*, only 31 are good. The complete list of combinations is in Cunha et al. (2014a).

Combinations of F with A, B, C, D, E, and F that have 11/8-sequences are called *well-behaved combinations*—the ones in Lemmas 6, 7, and the good *F–A–A* combinations. The remaining combinations having F are called *naughty*: the seven combinations of $F - F$ that have no 11/8-sequence, and the 57 combinations of *F–A–A*.

For extensions that yield a bad small configuration, Algorithm 6 adds their cycles to a set $\mathscr{S}$ (line 32). Later, if a well-behaved combination is found among the cycles in $\mathscr{S}$, an 11/8-sequence is applied (line 37) and the set is emptied. If all combinations in $\mathscr{S}$ are naughty, another bad small configuration can be obtained and added to it in the next iteration (line 6).

We have shown (Cunha et al., 2014a) that every combination of three copies of F is well-behaved, even if each pair of $F - F$ is naughty; the same can be said of every combination of F and three copies of A, even if each triple *F–A–A* is naughty. Therefore, at most 12 cycles are in $\mathscr{S}$, since it may contain at most three copies of F, or one copy of F and three copies of A, in the worst case. For each of these cases, there exists an 11/8-sequence (Cunha et al., 2014a).

Proposed algorithm Algorithm 6 is a direct application of the results in the section. In a nutshell, it obtains configurations using the *Query* procedure and applies 11/8-sequences to configurations of size at most 9. Algorithm 6 differs from Algorithm 1 not only in the use of permutation trees, but also because the main loop handles bad small full configurations, instead of only dealing with them at the end.

Algorithm 6: Proposed algorithm based on Elias and Hartman's algorithm

```
1  Transform permutation π into a simple permutation π̂.
2  Find and apply (2,2)-sequence (Algorithm 5).
3  While G(π̂) contains a 2-cycle, apply a 2-move.
4  π̂ consists of 3-cycles. Mark all 3-cycles in G(π̂).
5  Let 𝒮 be an empty set.
6  while G(π̂) contains at least eight 3-cycles do
7      Start a configuration 𝒞 with a marked 3-cycle.
8      if the cycle in 𝒞 is oriented then
9          Apply a 2-move.
10     else
11         Try to sufficiently extend 𝒞 eight times using the Query procedure.
12         if 𝒞 is a sufficient configuration with nine cycles then
13             Apply an 11/8-sequence.
14         else
               𝒞 is a small full configuration
15             if 𝒞 is a good small configuration then
16                 Apply an 11/8-sequence.
17             else
                   𝒞 is a bad small configuration
18                 Add every cycle in 𝒞 to 𝒮.
19                 Unmark all cycles in 𝒞.
20                 if 𝒮 contains a well-behaved combination then
21                     Apply an 11/8-sequence.
22                     Mark the remaining 3-cycles in 𝒮.
23                     Remove all cycles from 𝒮.
24 While G(π̂) contains a 3-cycle, apply a 4/3-sequence or a 3/2-sequence.
25 Mimic the sorting of π using the sorting of π̂.
```

Theorem 1 *Algorithm 6 runs in O(n*log*n) time.*

Proof. Steps 1 through 5 can be implemented to run in linear time [Elias and Hartman (2006); Feng and Zhu (2007), and section 4]. Step 17 runs in $O(\log n)$ time using permutation trees. The comparisons in Steps 12, 15, and 20 are done in $O(1)$ time using lookup tables whose sizes are bounded by a constant. Updating the set $\mathscr{S}$ also requires constant time, since it has at most 12 cycles (case where $\mathscr{S}$ contains $F - F - F$). Every sequence of transpositions of size bounded by a constant can be applied in time $O(\log n)$ due to the use of permutation trees. The time complexity of the loop between Steps 6 to 23 is $O(n\log n)$, since the number of 3-cycles is linear in n, and the number of cycles decreases, in the worst case, every third iteration. In Step 24, the search for a 4/3– or a 3/2-sequence is done in constant time, since the number of cycles is bounded by a constant. Steps 24 and 25 also run in time $O(n\log n)$, according to Feng and Zhu (2007). ■

6. FINAL REMARKS

Although sorting permutations by transpositions is an NP-hard problem, some approximation strategies have been successful. This article describes a 1.375-approximation algorithm that rectifies a previous attempt (Firoz et al., 2011) of using the permutation tree data structure to achieve a running time of $O(n\log n)$. We have managed to achieve both the 1.375 approximation and the $O(n\log n)$ running time. The approximation ratio is guaranteed by a new computational case analysis (Cunha et al., 2014a) that finds 11/8-sequences for bad small full configurations. The running time is attained by providing, for the first time, a correct linear-time strategy for finding and applying a (2,2)-sequence.

ACKNOWLEDGMENTS

This work has been partially supported by grants from Brazilian agencies Faperj, CNPq, and CAPES.

AUTHOR DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Bafna, V., and Pevzner, P.A. 1998. Sorting by transpositions. *SIAM J. Discrete Math.* 11, 224–240.

Bulteau, L., Fertin, G., and Rusu, I. 2012. Sorting by transpositions is difficult. *SIAM J. Discrete Math.* 26, 1148–1180.

Christie, D.A. 1999. Genome rearrangement problems [Ph.D. thesis]. University of Glasgow, UK.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2013a. Advancing the transposition distance and diameter through lonely permutations. *SIAM J. Discrete Math.* 27, 1682–1709.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2013b. On the 1.375-approximation algorithm for sorting by transpositions in O(nlogn) time. Proceedings of the 8th Brazilian Symposium on Bioinformatics, volume 8213 of Lectures Notes in Bioinformatics, pp. 126–135.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2014a. 1.375-approximation for SBT in O(n log n) time. Available at: http://compscinet.org/research/sbt1375 Accessed September 2, 2015.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2014b. A faster 1.375-approximation algorithm for sorting by transpositions. Proceedings of the 14th Workshop on Algorithms in Bioinformatics, volume 8701 of Lectures Notes in Bioinformatics, pp. 26–37.

Elias, I., and Hartman, T. 2006. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 369–379.

Feng, J., and Zhu, D. 2007. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Trans. Algorithms* 3, 1549–6325.

Fertin, G., Labarre, A., Rusu, I., et al. 2009. *Combinatorics of Genome Rearrangements*. The MIT Press, New York.

Firoz, J.S., Hasan, M., Khan, A.Z., and Rahman, M.S. 2011. The 1:375 approximation algorithm for sorting by transpositions can run in O(nlogn) time. *J. Comput. Biol.* 18, 1007–1011.

Hannenhalli, S., and Pevzner, P.A. 1999. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM* 46, 1–27.

Hartman, T., and Shamir, R. 2006. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Inf. Comput.* 204, 275–290.

Labarre, A. 2006. New bounds and tractable instances for the transposition distance. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 380–394.

Address correspondence to:
Luís Felipe I. Cunha
Centro de Tecnologia
Universidade Federal do Rio de Janeiro
Bloco H, Sala 317-10
Rio de Janeiro 21941972
Brazil

E-mail: lfignacio@cos.ufrj.br