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], and Celina M.H. de Figueiredo[1]

[1] Universidade Federal do Rio de Janeiro, Brasil
{lfignacio,celina}@cos.ufrj.br
[2] Universidade Federal Fluminense, Brasil
luis@vm.uff.br
[3] Universidade Federal do ABC, Brasil
hausen@compscinet.org

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 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 algorithm that runs in $O(n^2)$ time. In this paper, we propose the first correct adaptation of this algorithm to run in $O(n \log n)$ time.

Keywords: comparative genomics, genome rearrangement, sorting by transpositions, approximation algorithms.

1 Introduction

By comparing the orders of common genes between two organisms, one may estimate the series of mutations that occurred in the underlying evolutionary process. In a simplified genome rearrangement model, each mutation is a transposition, and the sole chromosome of each organism is modeled by a permutation, which means that there are no duplicated or deleted genes. A *transposition* is a rearrangement of the gene order within a chromosome, in which two contiguous blocks are swapped. The transposition distance is the minimum number of transpositions required to transform one chromosome into another. Bulteau *et al.* [3] proved that the problem of determining the transposition distance between two permutations – or Sorting by Transpositions (SBT) – is NP-hard.

Several approaches to handle 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 [2] designed a 1.5-approximation $O(n^2)$ algorithm, based on the cycle structure of the *breakpoint graph*. Hartman and Shamir [10], by

D. Brown and B. Morgenstern (Eds.): WABI 2014, LNBI 8701, pp. 26–37, 2014.
© Springer-Verlag Berlin Heidelberg 2014

considering *simple permutations*, proposed an easier 1.5-approximation algorithm and, by exploiting a balanced tree data structure, decreased the running time to $O(n^{\frac{3}{2}}\sqrt{\log n})$. Feng and Zhu [7] developed the balanced *permutation tree* data structure, further decreasing the complexity of Hartman and Shamir's 1.5-approximation algorithm to $O(n \log n)$.

Elias and Hartman [6] obtained, by a thorough computational case analysis of cycles of the breakpoint graph, a 1.375-approximation $O(n^2)$ algorithm. Firoz *et al.* [8] tried to lower the running time of this algorithm to $O(n \log n)$ via a simple application of permutation trees, but we later found counter-examples [5] that disprove the correctness of Firoz *et al.*'s strategy.

In this paper, we propose a new algorithm that uses the strategy of Elias and Hartman towards *bad full configurations*, implemented using permutation trees and achieving both a 1.375 approximation ratio and $O(n \log n)$ time complexity. Section 2 contains basic definitions, Section 3 presents a strategy to find in linear time a sequence of two transpositions in which both are 2-moves, if it exists, and Section 4 describes our 1.375-approximation algorithm for SBT.

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 is a unique integer in the range $1, \ldots, n$. The *transposition* $t(i,j,k)$, where $1 \leq i < j < k \leq n+1$ *over* π, is the permutation $\pi \cdot t(i,j,k)$ where the product interchanges the two contiguous blocks $\pi_i\, \pi_{i+1} \ldots \pi_{j-1}$ and $\pi_j\, \pi_{j+1} \ldots \pi_{k-1}$. A sequence of q transpositions *sorts* a permutation π if $\pi\, t_1\, t_2 \cdots t_q = \iota$, where every t_i is a transposition and 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.

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 edges are partitioned into two sets, the directed *reality edges* $R = \{\, \overrightarrow{i} = (+\pi_i,\ -\pi_{i+1}) \mid i = 0, \ldots, n\}$ and the undirected *desire edges* $D = \{(+i,\ -(i+1)) \mid i = 0, \ldots, n\}$). Fig. 1 shows $G([\mathbf{0}\, 10\, 9\, 8\, 7\, 1\, 6\, 11\, 5\, 4\, 3\, 2\, \mathbf{12}])$, the horizontal lines represent the edges in R and the arcs represent the edges in D.

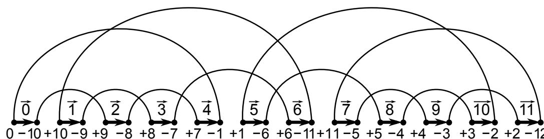

Fig. 1. $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$. The cycles $C_2 = \langle 1\,3\,6 \rangle$ and $C_3 = \langle 5\,8\,10 \rangle$ intersect, but C_2 and C_3 are not interleaving; the cycles $C_1 = \langle 0\,2\,4 \rangle$ and $C_2 = \langle 1\,3\,6 \rangle$ are interleaving, and so are $C_3 = \langle 5\,8\,10 \rangle$ and $C_4 = \langle 7\,9\,11 \rangle$.

Every vertex in $G(\pi)$ has degree 2, so $G(\pi)$ 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.

Non-trivial bounds on the transposition distance were obtained by using the breakpoint graph [2], after applying a transposition t, the number of cycles of odd length in $G(\pi)$, denoted $c_{odd}(\pi)$, is changed such that $c_{odd}(\pi t) = c_{odd}(\pi) + x$, where $x \in \{-2, 0, 2\}$ and t is said to be an *x-move for π*. Since $c_{odd}(\iota) = n + 1$, we have the lower bound $d(\pi) \geq \left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil$, where the equality holds if, and only if, π can be sorted with only 2-moves.

Hannenhalli and Pevzner [9] proved that every permutation π can be transformed into a simple one $\hat{\pi}$, by inserting new elements on appropriate positions of π, preserving the lower bound for the distance, $\left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil = \left\lceil \frac{(m+1)-c_{odd}(\hat{\pi})}{2} \right\rceil$ where m is such that $\hat{\pi} = [\mathbf{0}\hat{\pi}_1 \dots \hat{\pi}_m \boldsymbol{m{+}1}]$. Additionally, a sequence that sorts $\hat{\pi}$ can be transformed into a sequence that sorts π, which implies that $d(\pi) \leq d(\hat{\pi})$. This method is commonly used in the literature, as in Hartman and Shamir's [10] and Elias and Hartman's [6] approximation algorithms.

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 (name given by the relative order of such a triplet of reality edges), otherwise it is *unoriented*. If there exists a 2-move that may be applied to π, then π is *oriented*, otherwise π is *unoriented*.

A sequence of q transpositions in which exactly r transpositions are 2-moves is a *(q, r)-sequence*. A $\frac{q}{r}$-sequence is a (x, y)-sequence such that $x \leq q$ and $\frac{x}{y} \leq \frac{q}{r}$.

A cycle in π is determined by its reality edges, in the order that they appear, starting from the leftmost edge. The notation $C = \langle x_1 x_2 \dots x_\ell \rangle$, where $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, $\dots$, $\overrightarrow{x_\ell}$ are reality edges, and $x_1 = \min\{x_1, x_2, \dots, x_\ell\}$, characterizes an ℓ-cycle.

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, i.e. $x < a < y < b$ or $a < x < b < y$. Similarly, two triplets of reality edges $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$ and $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$ are *interleaving* if these six edges occur in an alternating order, i.e. $x < a < y < b < z < c$ or $a < x < b < y < c < z$. Two cycles C and C' *intersect* if there is a pair of reality edges in C that intersects with a pair of reality edges in C', and two 3-cycles are *interleaving* if their respective triplets of reality edges are interleaving. See Fig. 1.

A *configuration* of π is a subset of the cycles in $G(\pi)$. A configuration $\mathcal{C}$ is *connected* if, for any two cycles C_1 and C_k in $\mathcal{C}$, there are cycles $C_1, ..., C_{k-1} \in \mathcal{C}$ such that, for each $i \in \{1, 2, ..., k-1\}$, the cycles C_i and C_{i+1} are either intersecting or interleaving. If the configuration $\mathcal{C}$ is connected and maximal, then $\mathcal{C}$ is a *component*. Every permutation admits a unique decomposition into disjoint components. For instance, in Fig. 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 a configuration $\mathcal{C}$. An *open gate* is a pair of reality edges of C that does not intersect any other pair of reality edges in $\mathcal{C}$. If a configuration

$\mathcal{C}$ has only 3-cycles and no open gates, then $\mathcal{C}$ is a *full configuration*. Some full configurations, such as the one in Fig. 2(a), do not correspond to the breakpoint graph of any permutation [6].

A configuration $\mathcal{C}$ that has k edges is in the *cromulent form*[1] if every edge from $\overrightarrow{0}$ to $\overrightarrow{k-1}$ is in $\mathcal{C}$. Given a configuration $\mathcal{C}$ having k edges, a *cromulent relabeling* (Fig. 2b) of $\mathcal{C}$ is a configuration $\mathcal{C}'$ such that $\mathcal{C}'$ is in the cromulent form and there is a function σ satisfying that, for every pair of edges $\overrightarrow{i}$, $\overrightarrow{j}$ in $\mathcal{C}$ such that $i < j$, we have that $\overrightarrow{\sigma(i)}, \overrightarrow{\sigma(j)}$ are in $\mathcal{C}'$ and $\sigma(i) < \sigma(j)$.

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

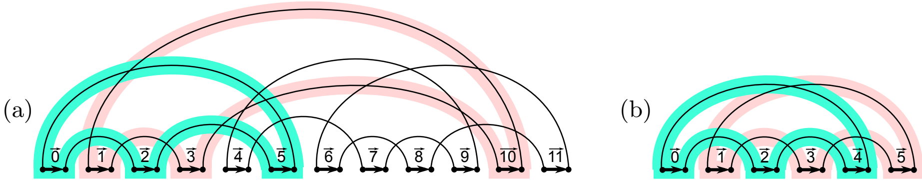

Fig. 2. (a) Full configuration $\{C_1, C_2, C_3, C_4\} = \{\langle 0\,2\,5\rangle, \langle 1\,3\,10\rangle, \langle 4\,7\,9\rangle, \langle 6\,8\,11\rangle\}$. (b) The cromulent relabeling of $\{C_1, C_2\}$ is $\{\langle 0\,2\,4\rangle, \langle 1\,3\,5\rangle\}$.

Elias and Hartman's algorithm Elias and Hartman [6] performed a systematic enumeration of all components having nine or less cycles, 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 $\mathcal{C}$ is a connected configuration $\mathcal{C} \cup \{C\}$, where $C \notin \mathcal{C}$. 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 named *sufficient configuration*, which has an (x, y)-, or $\frac{x}{y}$-, sequence if it is possible to apply such a sequence to its cycles.

Lemma 1. *[6] Every unoriented sufficient configuration of nine cycles has an* $\frac{11}{8}$*-sequence.*

Components with less than nine cycles are called *small components*. Elias and Hartman showed that there are just five kinds of small components that do not have an $\frac{11}{8}$-sequence; these components are called *bad small components*. Small components that have an $\frac{11}{8}$-sequence are *good small components*.

Lemma 2. *[6] 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* $E = \{\langle 0\,2\,16\rangle, \langle 1\,3\,5\rangle, \langle 4\,6\,8\rangle, \langle 7\,9\,11\rangle, \langle 10\,12\,14\rangle, \langle 13\,15\,17\rangle\}$.

[1] *cromulent*: neologism coined by David X. Cohen, meaning "normal" or "acceptable."

If a permutation has bad small components, it is still possible to find $\frac{11}{8}$-sequences, as Lemma 3 states.

Lemma 3. *[6] 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. *[6] If every cycle in $G(\pi)$ is a 3-cycle, and there are at least eight cycles, then π has an $\frac{11}{8}$-sequence.*

Lemmas 1 and 3, and Corollary 1 form the theoretical basis for Elias and Hartman's $\frac{11}{8} = 1.375$-approximation algorithm for SBT, shown in Algorithm 1.

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

```
1  Transform permutation π into a simple permutation π̂.
2  Check if there is a (2,2)-sequence. If so, apply it.
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  while G(π̂) contains a marked 3-cycle C do
6      if C is oriented then
7          Apply a 2-move to it.
8      else
9          Try to sufficiently extend C eight times (to obtain a configuration with
           at most 9 cycles).
10         if sufficient configuration with 9 cycles has been achieved then
11             Apply an 11/8-sequence.
12         else It is a small component
13             if it is a good component then
14                 Apply an 11/8-sequence.
15             else
16                 Unmark all cycles of the component.
17 (Now G(π̂) has only bad small components.)
18 while G(π̂) contains at least eight cycles do
19     Apply an (11,8)-sequence
20 While G(π̂) contains a 3-cycle, apply a (3,2)-sequence.
21 Mimic the sorting of π using the sorting of π̂.
```

Feng and Zhu's permutation tree Feng and Zhu [7] introduced the *permutation tree*, a binary balanced tree that represents a permutation, and provided four 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, to *split* a permutation tree into two in $O(\log n)$ time, and to *query* a permutation tree and find reality edges that intersect a given pair $\overrightarrow{i}, \overrightarrow{j}$ in $O(\log n)$ time.

Operations *split* and *join* allow applying a transposition to a permutation π and updating the tree in time $O(\log n)$. Lemma 4 provides a way to determine, in logarithmic time, which transposition should be applied to a permutation, and serves as the basis for the *query* procedure. This method was applied [7] to Hartman and Shamir's 1.5-approximation algorithm [10], to find a $(3, 2)$-sequence that affects a pair of intersecting or interleaving cycles.

Lemma 4. *[2] 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* $\overrightarrow{k}$ *and* $\overrightarrow{\ell - 1}$ *belong to the same cycle, and the pair* $\overrightarrow{k}, \overrightarrow{\ell - 1}$ *intersects the pair* $\overrightarrow{i}, \overrightarrow{j}$*.*

Firoz's et al. *use of the permutation tree* Firoz *et al.* [8] suggested the use of the permutation tree to reduce the running time of Elias and Hartman's [6] algorithm. In [5], we showed that this strategy fails to extend some full configurations.

Firoz *et al.* [8] stated that extensions can be done in $O(\log n)$ time. To do that, they categorized sufficient extensions of a configuration A into *type 1 extensions* – those that add a cycle that closes open gates – 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 by running *query* for an open gate. In a type 2 extension, since there are no open gates, Firoz *et al.* claimed that it is sufficient to perform queries on all pairs of reality edges belonging to the same cycle in a configuration that is being extended. But, as shown in [5], there is an infinite family of configurations for which this strategy fails; some instances are subsets of two cycles of $[\mathbf{0}\, 10\, 9\, 8\, 7\, 1\, 6\, 11\, 5\, 4\, 3\, 2\, \mathbf{12}]$ (Fig. 1). Consider the configuration $A = \{C_1\}$; try to sufficiently extend A (step 9 in Algorithm 1) using the steps proposed by Firoz *et al.*:

1. Configuration A has three open gates. Executing the *query* for an open gate results in a pair of edges belonging the cycle C_2. Therefore, we add this cycle to the configuration A, which becomes $A = \{C_1, C_2\}$.
2. Configuration A has no more open gates. Executing the *query* for every pair of edges in the same cycle of A, we observe that the *query* will return a pair that is already in A. So far, Firoz *et al.*'s method has failed to extend A.

3 Finding a (2, 2)-Sequence in Linear Time

Elias and Hartman [6] proved that, given a simple permutation, a $(2, 2)$-sequence can be found in $O(n^2)$ time. Firoz *et al.* [8] described a strategy for finding and applying a $(2, 2)$-sequence in $O(n \log n)$ time using permutation trees and the result in Lemma 5; see below. But, according to their strategy, it is still necessary to search for an oriented cycle in $O(n)$ time and, after applying the first 2-move, checking for the existence of an oriented cycle, again in $O(n)$ time. However, these steps must be performed $O(n)$ times in the worst case, which implies that Firoz *et al.*'s strategy also takes $O(n^2)$ time.

Algorithm 2. Search $(2,2)$-sequence from K_1

```
1  for i = min K_1+1, ..., mid K_1−1 do
2      if →i belongs to an oriented cycle K_j then
3          if mid K_j < mid K_1 or max K_j < max K_1 then
4              return (2,2)-sequence that affects K_1 and K_j.
5      if →i belongs to an unoriented cycle L_j then
6          if midK_1 < midL_j < maxK_1 < maxL_j then
7              return (2,2)-sequence that affects K_1 and L_j.
8  for i = mid K_1+1, ..., max K_1−1 do
9      if →i belongs to an oriented cycle K_j then
10         if mid K_1 < min K_j then
11             return (2,2)-sequence that affects K_1 and K_j.
12 for i = max K_1+1, ..., n−1 do
13     if →i belongs to an oriented cycle K_j then
14         if max K_1 ≤ min K_j then
15             return (2,2)-sequence affecting K_1 and K_j.
```

Algorithm 4 summarizes our approach towards finding and applying a $(2,2)$-sequence in $O(n)$ time.

Lemma 5. *[2,4,6] 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 non intersecting* 2*-cycles, and the resulting graph contains an oriented cycle after the first transposition is applied;*
2. *there are two non interleaving 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 with checking whether a breakpoint graph satisfies Lemma 5, as described in detail in Algorithm 2. It differs from previous approaches [6,8] in that the leftmost oriented cycle, dubbed K_1, is fixed when verifying conditions 2 and 3, avoiding comparisons between every pair of cycles.

Given a simple permutation π, it is trivial 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 [4] 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 $(2,2)$-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 non-intersecting 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 over the 3-cycle, and the second 2-move is over the 2-cycles; ii) if the 3-cycle is unoriented, then first we apply the 2-move over the 2-cycles, and the second 2-move is over 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 no even cycle satisfies the previous conditions, but there is an oriented cycle, the 3-cycles must be scanned for the existence of a $(2, 2)$-sequence, as required conditions 2 and 3 in Lemma 5.

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

1. Check for the existence of an oriented cycle K_j non-interleaving K_1 or an unoriented cycle L_j interleaving K_1. Algorithm 2 solves that: between $\min K_1$ and $\operatorname{mid} K_1$, between $\operatorname{mid} K_1$ and $\max K_1$, and to the right of $\max K_1$, search for an oriented cycle K_i non-interleaving K_1 or an unoriented cycle L_i interleaving K_1.
2. If every oriented cycle interleaves K_1 and no unoriented cycle interleaves K_1, then check for the existence of two oriented cycles K_i, K_j that are intersecting but not interleaving. Notice that if there is a pair of non-interleaving oriented cycles, then the cycles intersect each other, otherwise one of the cycles would be non-interleaving K_1, and Algorithm 2 would have this case already covered (see Fig. 3). Algorithm 3 describes how to verify the existence of two intersecting oriented cycles that are also interleaving K_1.

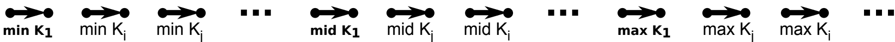

Fig. 3. Oriented cycles represented by their reality edges. All oriented cycles interleave K_1, but K_i and K_j non-interleave each other.

4 Sufficient Extensions Using *Query*

At the end of Section 2, we discussed Firoz's *et al.* use of the permutation tree, and as proven in [5], their strategy does not account for configurations with less than nine cycles that are not components, since successive invocations of the *query* procedure may result in a full configuration with less than nine cycles that is not a small component. Our proposed strategy generalizes the definitions related to small components by defining a *small configuration*, a configuration with less than nine cycles.

Algorithm 3. Finding intersecting oriented cycles interleaving K_1.

1 s_1 = sequence of edges belonging to oriented cycles from left to right between $\min K_1$ and $\operatorname{mid} K_1$.
2 s_2 = sequence of edges belonging to oriented cycles from left to right between $\operatorname{mid} K_1$ and $\max K_1$.
3 **if** *s_1 and s_2 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.

Algorithm 4. 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 K_1 returns a sequence* **then**
10 Apply $(2,2)$-sequence.
11 **else if** *Finding intersecting oriented cycles interleaving K_1* **then**
12 Apply $(2,2)$-sequence.
13 **else**
14 There are no $(2,2)$-sequences to apply.

A small configuration is said to be *full* if it has no open gates. Small configurations are also classified as *good* if they have an $\frac{11}{8}$-sequence, or as *bad* otherwise.

Algorithm 1 applies an $\frac{11}{8}$-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 the existence of an $(11,8)$-sequence in every combination of bad small components with at least 8 cycles.

By doing extensions using the *query* procedure, we can deal with bad small full configurations, which may or may not be bad small components. The possible bad small full configurations are the bad small components A, B, C, D and E, from Lemma 2, and one more full configuration

$$F = \{\langle 0\,7\,9\rangle, \langle 1\,3\,6\rangle, \langle 2\,4\,11\rangle, \langle 5\,8\,10\rangle\},$$

which is the only bad small full configuration that is not a component [6].

Our strategy (Algorithm 5) is similar to Elias and Hartman's (Algorithm 1): we apply an $\frac{11}{8}$-sequence to every sufficient unoriented configuration of nine cycles, and additionally to every good small full configuration; the main difference is that, whenever a combination of bad small full configuration is found, a decision to apply an $\frac{11}{8}$-sequence is made according to Lemma 6.

Lemma 6. *Every combination of F with one or more copies of either B, C, D or E has an $\frac{11}{8}$-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, in [1], enumerates every possible breakpoint graph and provides an $\frac{11}{8}$-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 $\frac{11}{8}$-sequence. Let $F_i F^j$ 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 $\frac{11}{8}$-sequence for $F_i F^j$, 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 $\frac{11}{8}$-sequences for the cases enumerated above were also found through a computerized case analysis [1]. Note that $F_i F^j$ is equivalent to $F_{i+6} F^j$ for $i = \{0, 1, \ldots, 5\}$, which simplifies our analysis. □

The combinations of F with F for which our branch-and-bound case analysis cannot find an $\frac{11}{8}$-sequence are: F_1F^0, F_2F^0, F_3F^0, F_5F^0, F_5F^2, F_5F^3 and F_5F^4.

All combinations of one copy of F and one of A have less than eight cycles. It only remains to analyse 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 $\frac{11}{8}$-sequence exists. Out of 57 combinations of F–A–A, only 31 are good. The explicit list of combinations is in [1].

Combinations of F and A, B, C, D, E that have an $\frac{11}{8}$-sequence 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*.

For extensions that yield a bad small configuration, Algorithm 5 adds their cycles to a set $\mathcal{S}$ (line 18). Later, if a well-behaved combination is found among the cycles in $\mathcal{S}$, an $\frac{11}{8}$-sequence is applied (line 21) and the set is emptied. The set $\mathcal{S}$ may just contain naughty combinations and in the next iteration (line 6) another bad small configuration may be obtained and added to $\mathcal{S}$. We have shown [1] that every combination of three copies of F is well-behaved, even if

each pair of F is naughty; the same can also be said of every combination of F and three copies of A such that each triple F–A–A is naughty. Therefore, at most 12 cycles are in $\mathcal{S}$, since there are in the worst case three copies of F; or one copy of F and three copies of A. In all these cases we apply $\frac{11}{8}$-sequences as proved in [1].

New Algorithm. The previous results allow us to devise Algorithm 5, that basically obtains configurations using the *query* procedure, and applies $\frac{11}{8}$-sequences to configurations of size at most 9. It differs from Algorithm 1 not only in the use of permutation trees, but also because we continuously deal with bad small full configurations instead of only at the end.

Algorithm 5. New 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 4).
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 to it.
10     else
11         Try to sufficiently extend C eight times.
12         if C is a sufficient configuration with 9 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 5 runs in $O(n \log n)$ time.*

Proof. Steps 1 through 5 can be implemented to run in linear time (proofs in [6] and in Sect. 3). Step 11 runs in $O(\log n)$ time using permutation trees. The

comparisons in Steps 12, 14, 15, 17 and 20 are done in constant time using lookup tables of size bound by a constant. Updating the set $\mathcal{S}$ also requires constant time, since it has at most 12 cycles. 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 cycles decreases, in the worst case, once in three iterations. 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 $O(n \log n)$. □

5 Conclusion

The goal of this paper is to lower the time complexity of Elias and Hartman's [6] 1.375-approximation algorithm down to $O(n \log n)$. Our new approach provides, so far, both the lowest fixed approximation ratio and time complexity of any non-trivial algorithm for sorting by transpositions.

We have previously shown that a simple application of permutation trees [7], as claimed in [8], does not suffice to correctly improve the running time of Elias and Hartman's algorithm. In order to lower the time complexity, it is necessary to add more configurations [1] to the original analysis in [6], and also to perform some changes in the sorting procedure, as shown in Algorithm 5.

References

1. http://compscinet.org/research/sbt1375 (2014)
2. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discrete Math. 11(2), 224–240 (1998)
3. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Discrete Math. 26(3), 1148–1180 (2012)
4. Christie, D.A.: Genome Rearrangement Problems. Ph.D. thesis, University of Glasgow, UK (1999)
5. Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., de Figueiredo, C.M.H.: On the 1.375-approximation algorithm for sorting by transpositions in $O(n \log n)$ time. In: Setubal, J.C., Almeida, N.F. (eds.) BSB 2013. LNCS, vol. 8213, pp. 126–135. Springer, Heidelberg (2013)
6. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans. Comput. Biol. Bioinformatics 3(4), 369–379 (2006)
7. Feng, J., Zhu, D.: Faster algorithms for sorting by transpositions and sorting by block interchanges. ACM Trans. Algorithms 3(3), 1549–6325 (2007)
8. Firoz, J.S., Hasan, M., Khan, A.Z., Rahman, M.S.: The 1.375 approximation algorithm for sorting by transpositions can run in $O(n \log n)$ time. J. Comput. Biol. 18(8), 1007–1011 (2011)
9. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. J. ACM 46(1), 1–27 (1999)
10. Hartman, T., Shamir, R.: A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Inf. Comput. 204(2), 275–290 (2006)