

Sorting by Weighted Reversals, Transpositions, and Inverted Transpositions

Martin Bader and Enno Ohlebusch

Computer Science Faculty,
University of Ulm, 89069 Ulm, Germany.
Email: martin.bader@uni-ulm.de
enno.ohlebusch@uni-ulm.de

Abstract. During evolution, genomes are subject to genome rearrangements that alter the ordering and orientation of genes on the chromosomes. If a genome consists of a single chromosome (like mitochondrial, chloroplast or bacterial genomes), the biologically relevant genome rearrangements are (1) *inversions*—also called *reversals*—where a section of the genome is excised, reversed in orientation, and reinserted and (2) *transpositions*, where a section of the genome is excised and reinserted at a new position in the genome; if this also involves an inversion, one speaks of an *inverted transposition*. To reconstruct ancient events in the evolutionary history of organisms, one is interested in finding an optimal sequence of genome rearrangements that transforms a given genome into another genome. It is well known that this problem is equivalent to the problem of “sorting” a signed permutation into the identity permutation. The complexity of the problem is still unknown. The best polynomial-time approximation algorithm, recently devised by Hartman and Sharan, has a 1.5 performance ratio. However, it applies only to the case in which reversals and transpositions are weighted equally. Because in most organisms reversals occur more often than transpositions, it is desirable to have the possibility of weighting reversals and transpositions differently. In this paper, we provide a 1.5-approximation algorithm for sorting by weighted reversals, transpositions and inverted transpositions for biologically realistic weights.

1 Introduction

During evolution, genomes are subject to genome rearrangements that alter the ordering and orientation (strandedness) of genes on the chromosomes. Because these events are rare compared to point mutations, they can give us valuable information about ancient events in the evolutionary history of organisms. For this reason, one is interested in the most “plausible” genome rearrangement scenario between two (or multiple) species. More precisely, given two genomes, one wants to find an optimal (shortest) sequence of rearrangement operations that transforms one into the other. Here we will focus on genomes that consists of a single (circular) molecule of DNA such as mitochondrial, chloroplast or bacterial genomes. As usual, the genomes are represented by a signed permutation,

i.e., an ordering of signed genes where the sign indicates the orientation (the strand). In this paper we do not consider unsigned permutations. In the single chromosome case, the relevant genome rearrangements are *inversions* (where a section of the genome is excised, reversed in orientation, and reinserted) and *transpositions* (where a section of the genome is excised and reinserted at a new position in the genome; if this also involves an inversion, one speaks of an *inverted transposition*). As is usually done in bioinformatics, we will use the terms “reversal” and “transreversal” as synonyms for “inversion” and “inverted transposition.” It is well known that the problem of finding an optimal sequence of rearrangement operations that transforms a permutation into another permutation is equivalent to the problem of “sorting” a permutation by the same set of operations into the identity permutation. Let us briefly recall what is known for various sets of operations. In a seminal paper, Hannenhalli and Pevzner showed that the problem of sorting by reversals can be solved in polynomial time [13]. The Hannenhalli-Pevzner theory was simplified [5] and the running time of their algorithm was improved several times. To date, a subquadratic time algorithm [19] is available, and the reversal distance problem (which asks solely for the minimum number of required reversals, but not for the sequence of reversals) is solvable in linear time [1, 6]. It is also worth mentioning that the problem of sorting an *unsigned* permutation by reversals is NP-hard [9] and the currently best approximation algorithm has the performance ratio 1.375 [7].

If one restricts the set of operations to transpositions (T), to transpositions and reversals (T + R), or to transpositions, reversals, and transreversals (T + R + TR), the complexity of the problem is still unknown. There exist polynomial-time approximation algorithms, and the best of them are listed in the table below.

operations	T	T + R	T + R + TR
performance ratio	1.375	2	1.5
references	[10]	[17, 20]	[15]

The biologically most relevant scenario is the T + R + TR case because in reality genomes are reorganized by all three kinds of operations. A drawback of Hartman and Sharan’s [15] 1.5-approximation algorithm is that it applies only to the case in which reversals and transpositions are weighted equally (called the unweighted case in this paper). Because a transposition can create two cycles in the reality-desire diagram while a reversal can create at most one cycle (see below), the algorithm generally favors transpositions. Consequently, the sequence of rearrangement operations returned by that algorithm will often significantly deviate from the “true” evolutionary history because in most organisms transpositions are observed much less frequently than reversals. Thus, it is desirable to have the possibility of weighting reversals and transpositions differently. Given such weights, the weighted genome rearrangement problem asks for a sorting sequence of rearrangement operations such that the sum of the weights of the operations in the sequence is minimal. That is, a shortest sequence is not necessarily optimal. However, this problem is poorly studied. To our knowledge, there are only two algorithms that tackle it. The first is a $(1+\epsilon)$ -approximation

algorithm devised by Eriksen [11]. It uses a weight proportion 2:1 (transposition:reversal) and has the tendency to use as much reversals as possible. The second algorithm is implemented in the software tool DERANGE II [8]. It is a greedy algorithm that works on the breakpoint distance and can only guarantee an approximation ratio of 3. In this paper, we will present a 1.5-approximation algorithm for any weight proportion between 1:1 and 2:1. Hence, our result closes the gap between the result of Hartman and Sharan [15] for the 1:1 proportion and that of Eriksen [11] for the 2:1 proportion. As the previous state of the art approximation algorithms for this problem, our algorithm proceeds by case analysis. In contrast to them, however, it is based on a (nontrivial) lower bound on the weighted rearrangement distance that is based on the number of odd *and* the number of even cycles. The running time of our algorithm is $O(n^2)$ in the naive implementation, but the time complexity can be improved to $O(n^{3/2} \log n)$.

2 Preliminaries

A *signed circular permutation* $\pi = (\pi_1 \dots \pi_n)$ is a permutation of $(1 \dots n)$, in which the indices are cyclic (i.e., n is followed by 1) and each element is labeled by plus or minus. We will use the term “permutation” as short hand for signed permutation. The *reflection* of a permutation π is the permutation $(-\pi_n \dots -\pi_1)$. It is considered to be equivalent to π . Two consecutive elements π_i, π_{i+1} form an *adjacency* if $\pi_i = +x$ and $\pi_{i+1} = +(x+1)$, or if $\pi_i = -x$ and $\pi_{i+1} = -(x-1)$. Otherwise, they form a *breakpoint*. A *segment* $\pi_i \dots \pi_j$ (with $j \geq i$) of a permutation π is a consecutive sequence of elements in π , with π_i as first element and π_j as last element. There are three possible rearrangement operations on a permutation π . A *transposition* $t(i, j, k)$ (with $i < j$ and $k < i$ or $k > j$) is an operation that cuts the segment $\pi_i \dots \pi_{j-1}$ out of π , and reinserts it before the element π_k . A *reversal* $r(i, j)$ (with $i < j$) is an operation that inverts the order of the elements of the segment $\pi_i \dots \pi_{j-1}$. Additionally, the sign of every element in the segment is flipped. A *transreversal* $tr(i, j, k)$ (with $i < j$ and $k < i$ or $k > j$) is the composition $t(i, j, k) \circ r(i, j)$ of a reversal and a transposition. In other words, the segment $\pi_i \dots \pi_{j-1}$ will be cut out of π , inverted, and reinserted before π_k . A *sequence* of operations op_1, op_2, \dots, op_k applied to a permutation π yields the permutation $op_k \circ op_{k-1} \circ \dots \circ op_1(\pi)$. In the following, reversals have weight w_r and transpositions as well as transreversals have weight w_t . As reversals usually occur much more frequently than transpositions and transreversals, we assume that $w_r \leq w_t$. The weight of a sequence is the sum of the weights of the operations in it. The problem of *sorting by weighted reversals, transpositions, and inverted transpositions* is defined as follows: Given a permutation π , find a sequence (of these operations) of minimum weight that transforms π into the identity permutation. This minimum weight will be denoted by $w(\pi)$.

In practice, it is also of interest to sort linear permutations. It has been proven by Hartman and Sharan [15] that sorting circular permutations is linearly equivalent to sorting linear permutations if yet another operation *revrev* is used

that inverts two consecutive segments of the permutation. As long as the weights for transreversals and revrevs are the same, the proof also holds for sorting with weighted operations. Hence, our algorithm for circular permutations can be adapted to an algorithm for linear permutations that also uses revrevs.

2.1 The reality-desire diagram

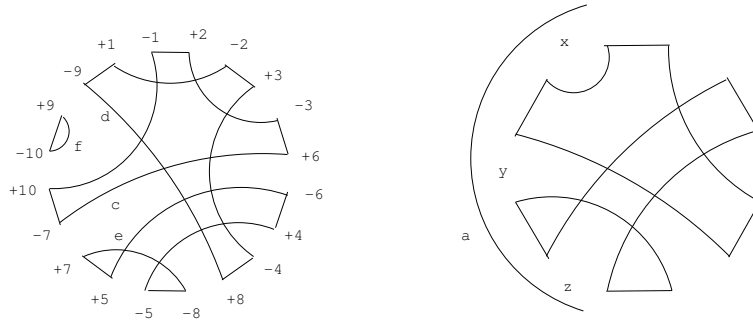


Fig. 1. Left: The reality-desire diagram of $\pi = (+1 + 9 + 10 + 7 - 5 + 8 + 4 + 6 + 3 + 2)$ contains the cycles c, d, e , and f . Cycles d and e are intersecting, cycles c and d are interleaving, and all other pairs of cycles do not intersect. Right: The configuration that consists of the cycles d and e . Labels x, y , and z mark three positions in the configuration and the arc a consists of these positions.

The reality-desire diagram [18] is a graph that helps us analysing the permutation; see Fig. 1. It is a variation of the breakpoint graph first described in [3]. The reality-desire diagram of a permutation $\pi = (\pi_1 \dots \pi_n)$ can be constructed as follows. First, the elements of π are placed counterclockwise on a circle. Second, each element x of π labeled by plus is replaced with the two nodes $-x$ and $+x$, while each element x labeled by minus is replaced with $+x$ and $-x$. We call the first of these nodes the *left node* of x and the other the *right node* of x . Third, *reality-edges* are drawn from the right node of π_i to the left node of π_{i+1} for each index i (indices are cyclic). Fourth, *desire-edges* or *chords* are drawn from node $+x$ to node $-(x+1)$ for each element x of π . We can interpret reality-edges as the actual neighborhood relations in the permutation, and desire-edges as the desired neighborhood relations. As each node is assigned exactly one reality-edge and one desire-edge, the reality-desire diagram decomposes into cycles. The *length* of a cycle is the number of chords in it. A k -cycle is a cycle of length k . If k is odd (even), we speak of an odd (even) cycle. The number of odd (even) cycles in π is denoted by $c_{\text{odd}}(\pi)$ ($c_{\text{even}}(\pi)$). It is easy to see that a 1-cycle corresponds to an adjacency and vice versa. A reversal cuts the permutation at two positions, while a transposition (transreversal) cuts it at three positions. Hence each of the operations cuts two or three reality-edges and

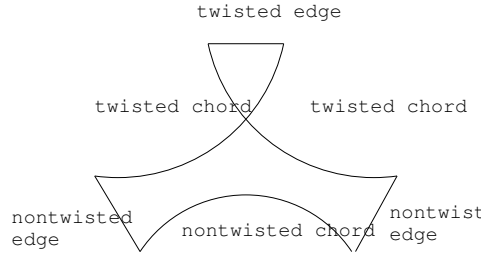


Fig. 2. An example for twisted reality-edges and twisted chords.

moves the nodes. We say that the operation *acts* on these edges. Desire-edges are never changed by an operation.

2.2 Some observations about cycles

The following notions are illustrated in Fig. 1. A *configuration* is a subset of the cycles of the reality-desire diagram of a permutation. Configurations help us to focus on a few cycles in the reality-desire diagram instead of examining the whole diagram. A *position* in a configuration is the position between two consecutive reality-edges in the configuration. An *arc* a is a series of consecutive positions of a configuration, bounded by two reality-edges r_1 and r_2 . Two chords d_1 and d_2 are *intersecting* if they intersect in the reality-desire diagram. More precisely, the endpoints of the chords must alternate along the circle in the configuration. Two cycles are intersecting if a pair of their chords is intersecting. Two cycles are *interleaving* if their reality-edges alternate along the circle. A rearrangement operation is called x_y -*move* if it increases the number of cycles by x and the operation is of type y (where r stands for a reversal, t for a transposition, and tr for a transreversal). For example, a transposition that splits one cycle into three is a 2_t -move. A reversal that merges two cycles is a -1_r -move. An $m_1m_2 \dots m_n$ -sequence is a sequence of n operations in which the first is an m_1 -move, the second an m_2 -move and so on. A cycle c is called *r -oriented* if there is a 1_r -move that acts on two of the reality-edges of c . Otherwise, the cycle is called *r -unoriented*. A cycle c is called *t -oriented* if there is a 2_t -move or a 2_{tr} -move that acts on three of the reality-edges of c . Otherwise, the cycle is called *t -unoriented*. A reality-edge is called *twisted* if its adjacent chords are intersecting; see Fig. 2. A chord is called *twisted* if it is adjacent to a twisted reality-edge; otherwise, it is called *nontwisted*. A cycle is called *k -twisted* if k of its reality-edges are twisted. If $k = 0$, we also say that the cycle is *nontwisted*.

Lemma 1. *A 2-cycle is r -oriented if and only if it is 2-twisted.*

Proof. There are only two possible configurations for a 2-cycle. If the cycle is 2-twisted, a reversal that acts on its reality-edges splits the cycle into two 1-cycles (adjacencies). Otherwise, no such move is possible.

Lemma 2. (proven in [14]) *A 3-cycle is t -oriented if and only if it is 2- or 3-twisted.*

Lemma 3. (proven in [12]) *If a cycle c of length ≥ 2 has a nontwisted chord, then there is another cycle d that intersects with this nontwisted chord of c .*

3 The Algorithm

We begin by introducing a new scoring function that allows us to show a very good lower bound for sorting by weighted reversals, transpositions, and inverted transpositions. Then, we will use the fact that a permutation can be transformed into an equivalent *simple permutation* without violating this lower bound. Because the sorting of the original permutation can be mimicked by the sorting of the simple permutation, we merely have to take care of simple permutations.

3.1 A lower bound

It has been proven by Gu et al. [12] that every operation changes the number of odd cycles by at most two. This fact leads to the following lower bound on $d(\pi)$.

Theorem 4. (goes back to [4, 12, 15]) *For any permutation $\pi = (\pi_1 \dots \pi_n)$, the inequality $d(\pi) \geq (n - c_{\text{odd}}(\pi))/2$ holds, where $d(\pi)$ denotes the minimum number of reversals, transpositions, and inverted transpositions required to sort π into the identity permutation.*

For sorting by *weighted* reversals, transpositions, and inverted transpositions, this bound is not good enough because it does not distinguish between the weights of the operations. More precisely, adapting the bound to the weighted case would lead to the bound $w(\pi) \geq (n - c_{\text{odd}}(\pi))w_r/2$ because $w_r \leq w_t$. However, the only way how a reversal can increase c_{odd} by two is to split an even cycle into two odd cycles. We will now define a scoring function that treats such a reversal and a transposition splitting one odd cycle into three odd cycles equally.

Definition 5. *The score $\sigma(\pi)$ of a permutation π is defined by*

$$\sigma(\pi) = c_{\text{odd}}(\pi) + \left(2 - \frac{2w_r}{w_t}\right) c_{\text{even}}(\pi)$$

Let op_i be a rearrangement operation. The weight w_i of op_i is defined to be w_r if op_i is a reversal and w_t otherwise. Furthermore, we define $\Delta\sigma_i = \sigma(op_i(\pi)) - \sigma(\pi)$ to be the gain in score after the application of op_i to the permutation π (a negative gain is possible). It is not difficult to verify that for each operation op_i , the inequality $\Delta\sigma_i/w_i \leq 2/w_t$ holds provided that $w_r \leq w_t \leq 2w_r$. Moreover, for the two operations discussed immediately before Definition 5, the inequality becomes an equality.

Lemma 6. For any permutation $\pi = (\pi_1 \dots \pi_n)$ and weights w_r, w_t with $w_r \leq w_t \leq 2w_r$:

- $\sigma(\pi) = n$ if π is the identity permutation
- $\sigma(\pi) \leq n - 1$ if π is not the identity permutation

Proof. If π is the identity permutation, the reality-desire diagram consists of n 1-cycles (adjacencies), so $\sigma(\pi) = c_{\text{odd}}(\pi) = n$. Otherwise, the diagram has at least one cycle of length ≥ 2 . Therefore, it has at most $n - 1$ cycles. An odd cycle adds 1 to the score, while an even cycle adds $2 - \frac{2w_r}{w_t}$. With $w_t \leq 2w_r$ it follows that $2 - \frac{2w_r}{w_t} \leq 1$. Thus, $\sigma(\pi) \leq n - 1$.

Theorem 7. For any permutation π and weights w_r, w_t with $w_r \leq w_t \leq 2w_r$, we have

$$w(\pi) \geq lb(\pi) \text{ where } lb(\pi) = c_{\text{even}}(\pi)w_r + \left(\frac{n - c_{\text{odd}}(\pi)}{2} - c_{\text{even}}(\pi) \right) w_t$$

Proof. Let op_1, op_2, \dots, op_k be an optimal sorting sequence of π , i.e., $w(\pi) = \sum_{i=1}^k w_i$. We have $\sigma(\pi) + \sum_{i=1}^k \Delta\sigma_i = n$ because π is transformed into the identity permutation, which has score n . It follows from $\Delta\sigma_i \leq w_i \frac{2}{w_t}$ that $n \leq \sigma(\pi) + \sum_{i=1}^k w_i \frac{2}{w_t} = \sigma(\pi) + w(\pi) \frac{2}{w_t}$. Hence $w(\pi) \geq (n - \sigma(\pi)) \frac{w_t}{2} = lb(\pi)$.

3.2 Transformation into simple permutations

The analysis of cycles of arbitrary length is rather complicated. For this reason, a permutation will be transformed into a so-called simple permutation. A cycle is called *long* if its length is greater than 3. A permutation is called *simple* if it contains no long cycles. According to [13–15, 17], there is a padding algorithm that transforms any permutation π into a simple permutation $\tilde{\pi}$. Each transformation step increases n and c_{odd} by 1, and leaves c_{even} unchanged. Hence $lb(\tilde{\pi}) = lb(\pi)$. As the padding algorithm just adds elements to π , π can be sorted by using a sorting sequence of $\tilde{\pi}$ in which the added elements are ignored. Consequently, the resulting sorting sequence of π has the same or a smaller weight than the sorting sequence of $\tilde{\pi}$. In the next subsection, we will present an algorithm that takes a simple permutation $\tilde{\pi}$ as input and outputs a sorting sequence op_1, op_2, \dots, op_k of $\tilde{\pi}$ such that $\sum_{i=1}^k w_i \leq 1.5 lb(\tilde{\pi})$. Altogether, this yields a 1.5-approximation for sorting by weighted reversals, transpositions, and inverted transpositions because $w(\pi) \leq \sum_{i=1}^k w_i \leq 1.5 lb(\tilde{\pi}) = 1.5 lb(\pi) \leq 1.5 w(\pi)$.

Note that it is not possible to transform 2-cycles into 3-cycles as done in [15] because these transformations would change the score and the lower bound.

3.3 The algorithm for simple permutations

Given a simple permutation π , the overall goal is to find a sorting sequence op_1, op_2, \dots, op_k of π such that $\sum_{i=1}^k \Delta\sigma_i \geq \sum_{i=1}^k w_i \frac{4}{3w_t}$. By a reasoning similar

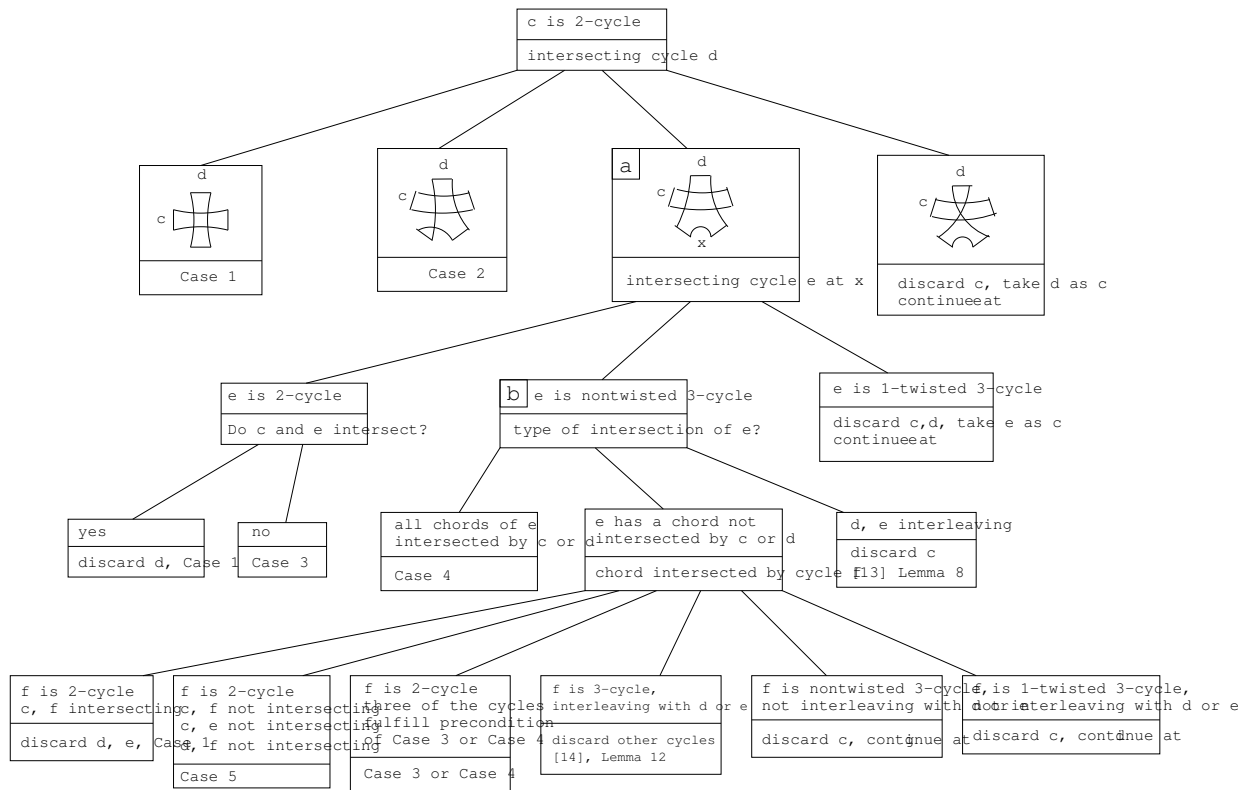


Table 1. The algorithm’s decision tree if it begins with an r-unoriented 2-cycle c . All cycles are considered to be r-unoriented 2-cycles or t-unoriented 3-cycles because r-oriented 2-cycles or t-oriented 3-cycles can directly be eliminated. Cross-references α and β can be found in this table, γ and δ in Table 2, while ε , ζ , and η are in Table 3.

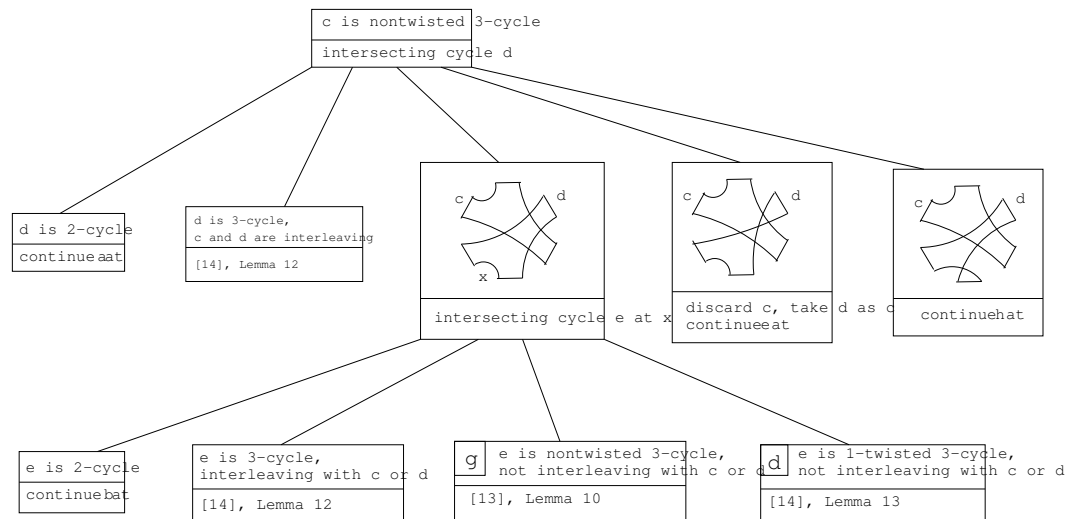


Table 2. The algorithm’s decision tree if it begins with a nontwisted 3-cycle c . All cycles are considered to be r-unoriented 2-cycles or t-unoriented 3-cycles because r-oriented 2-cycles or t-oriented 3-cycles can directly be eliminated. Cross-references α and β can be found in Table 1, γ and δ in this table, while ε , ζ , and η are in Table 3.

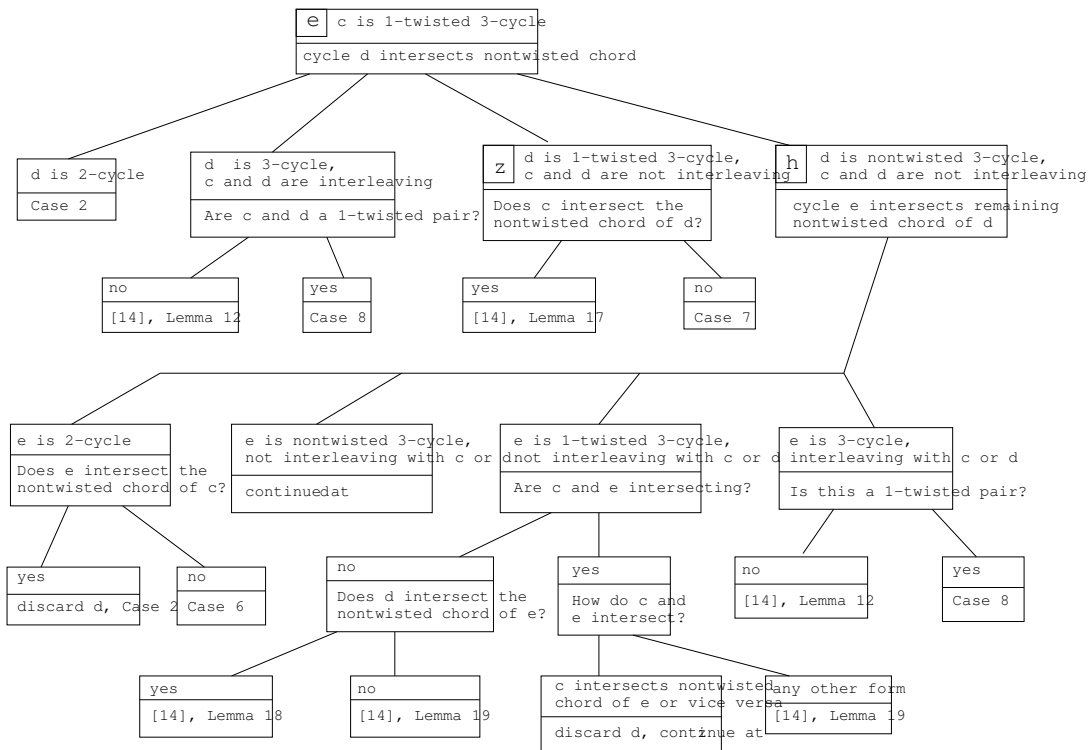


Table 3. The algorithm's decision tree if it begins with a 1-twisted 3-cycle c . Again, all cycles are r -unoriented 2-cycles or t -unoriented 3-cycles. Cross-references α and β can be found in Table 1, γ and δ in Table 2, while ε , ζ and η are in this table.

to the proof of Theorem 7, it then follows $\sum_{i=1}^k w_i \leq 1.5 lb(\pi)$. To achieve this goal, we search for a “starting sequence” op_1, \dots, op_j of at most four operations (i.e., $1 \leq j \leq 4$) such that $\sum_{i=1}^j \Delta\sigma_i \geq \sum_{i=1}^j w_i \frac{4}{3w_t}$. This procedure is iterated (i.e., we next search for a starting sequence of $op_j \circ \dots \circ op_1(\pi)$ etc.) until the identity permutation is reached.

The algorithm starts by searching for an arbitrary cycle c of length ≥ 2 in the reality-desire diagram of π . If the cycle is an r-oriented 2-cycle or a t-oriented 3-cycle, the starting sequence can consist solely of the operation op_1 that eliminates this cycle (i.e., op_1 is a 1_r , 2_t or 2_{tr} move that cuts the cycle into 1-cycles). This is because $\Delta\sigma_1/w_1 = 2/w_t \geq 4/3w_t$. Otherwise, according to Lemma 3, c must have a nontwisted chord that is intersected by another cycle d . The algorithm now searches for this cycle and examines the configuration of the cycles c and d . Depending on the configuration found, the algorithm either directly outputs a starting sequence that meets the requirements or, again by Lemma 3, there must be a chord in the configuration that is intersected by a cycle e that is not yet in the configuration. Consequently, the algorithm searches for this cycle and adds it to the configuration. This goes on until a configuration is found for which a starting sequence can be provided. The algorithm is based on a decision tree that can be found in Tables 1, 2, and 3. Note that every configuration consists of at most four cycles.

A careful inspection of the starting sequences described in [14] and [15] for configurations that do not contain 2-cycles reveals that these sequences also work in our case. Therefore, we merely have to consider configurations with at least one r-unoriented 2-cycle (recall that r-oriented 2-cycles can immediately be eliminated). These cases are listed below and example configurations can be found in Fig. 3.

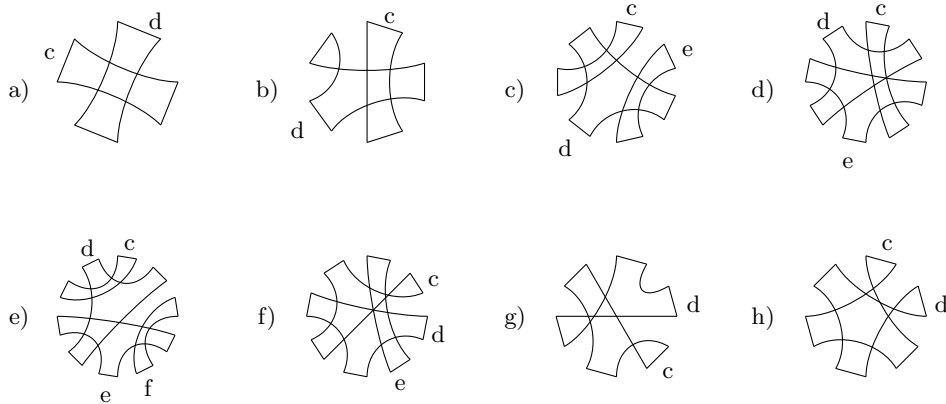


Fig. 3. Example configurations for the new cases to be taken into account.

Case 1 c and d are two intersecting 2-cycles (Fig. 3a).

Case 2 A 2-cycle c intersects the nontwisted chord of a 1-twisted 3-cycle d (Fig. 3b).

Case 3 c and e are 2-cycles, whereas d is a nontwisted 3-cycle. c and e are not intersecting, and each nontwisted chord of d is intersected by c or e (Fig. 3c).

Case 4 c is a 2-cycle, whereas d and e are intersecting nontwisted 3-cycles. c intersects the nontwisted chords of d and e that are not intersected by the other 3-cycle (Fig. 3d).

Case 5 d and e are two intersecting nontwisted 3-cycles, whereas c and f are 2-cycles. c intersects with the nontwisted chord of d that is not intersected by e , and f intersects with the nontwisted chord of e that is not intersected by d . c and d do not intersect with f , and e does not intersect with c (Fig. 3e).

Case 6 c is a 1-twisted 3-cycle and d is a nontwisted 3-cycle that intersects the nontwisted chord of c . The remaining chord of d (the one not intersected by c) is intersected by a 2-cycle e that does not intersect the nontwisted chord of c (Fig. 3f).

Case 7 c and d are two intersecting 1-twisted 3-cycles. d intersects the nontwisted chord of c , but c does not intersect the nontwisted chord of d (Fig. 3g).

Case 8 Two 1-twisted 3-cycles c and d form a 1-twisted pair (Fig. 3h).

Although the last two cases do not contain a 2-cycle, they have to be taken into account because in these cases we need a further intersecting cycle, which may be a 2-cycle.

To exemplify our method, we will give the starting sequences for Cases 4 and 6. Figs. 4 and 5 depict the configurations before and after the application of an operation in the sequence. In each configuration, the reality-edges on which the next operation acts are marked with \mathbf{x} or $*$. If three edges are marked with $*$, the operation is a transposition. If two edges are marked with \mathbf{x} and one is marked with $*$, the operation is a transreversal, and the segment between the two \mathbf{x} will be inverted. If two edges are marked with \mathbf{x} and none is marked with $*$, the operation is a reversal.

A full listing of the starting sequences can be found in [2]. In the following Δc_{odd} (Δc_{even}) denotes the change in the number of odd (even) cycles after the application of the starting sequence.

Lemma 8. For Case 4, there is a $0_r 1_r 2_{tr}$ -sequence with $\Delta c_{\text{odd}} = 4$ and $\Delta c_{\text{even}} = -1$.

Proof. The sequence is described in Fig. 4. We have $\sum \Delta \sigma_i / \sum w_i = 2(w_r + w_t) / w_t(2w_r + w_t)$. This value varies from $4/3w_t$ (for $w_t : w_r = 1 : 1$) to $3/2w_t$ (for $w_t : w_r = 2 : 1$).

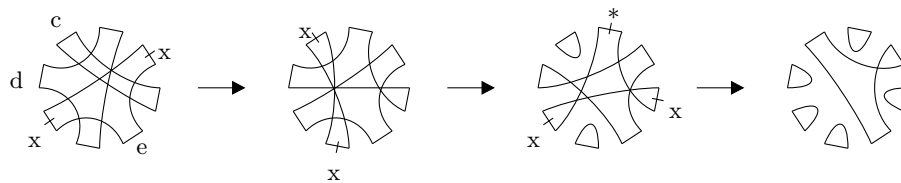


Fig. 4. Sequence for Case 4.

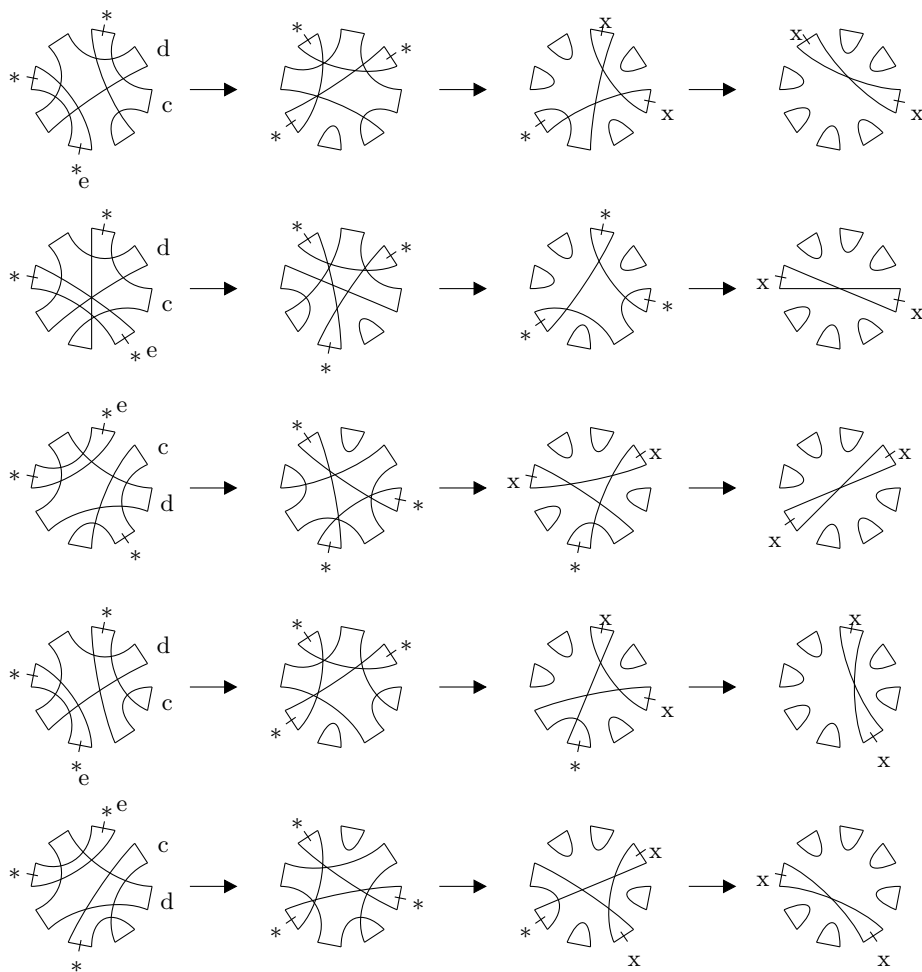


Fig. 5. The sequences for Case 6.

Lemma 9. For Case 6, there is a $0_t 2_t 2_t 1_r$ -sequence or a $0_t 2_t 2_{tr} 1_r$ -sequence with $\Delta c_{odd} = 6$ and $\Delta c_{even} = -1$.

Proof. There are five possible configurations. For all of them, a sequence is described in Fig. 5. The last operation of each sequence is a reversal that splits the last 2-cycle into adjacencies (the resulting configurations are not shown in the figure). Note that for these sequences, we have $\sum \Delta \sigma_i / \sum w_i = (4w_t + 2w_r) / w_t(3w_t + w_r)$. This value varies from $10/7w_t$ (for $w_t : w_r = 2 : 1$) to $3/2w_t$ (for $w_t : w_r = 1 : 1$).

4 Further Improvements

Our algorithm is implemented in C++ and it has time complexity $O(n^2)$. There are several possible improvements to the basic algorithm that can decrease its running time or its approximation ratio. Some of these improvements are:

- Using a special data structure described in [16], it is possible to find the different cases in sublinear time. The running time improves to $O(n^{3/2} \sqrt{\log n})$; cf. [15].
- Examining configurations with more cycles could improve the approximation ratio. Using this strategy, Elias and Hartman [10] recently succeeded in improving the performance ratio for sorting by transpositions from 1.5 to 1.375. It is highly expected that this strategy can also improve the performance ratio of sorting by weighted reversals, transpositions, and inverted transpositions.
- The algorithm can be combined with a greedy strategy: Instead of beginning with the first cycle in the reality-desire diagram, we start the search at each cycle in the diagram, and use a sequence with the best gain in score. This increases the running time by a factor of n , but the algorithm will find better sorting sequences, and changes in the weight function result in different sorting sequences.

References

1. D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8:483–491, 2001.
2. M. Bader. Sorting by weighted transpositions and reversals. Master’s thesis, University of Ulm, December 2005.
3. V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.
4. V. Bafna and P.A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.
5. A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. *Discrete Applied Mathematics*, 146(2):134–145, 2005.

6. A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proc. 15th Annual Symposium on Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 388–399. Springer-Verlag, 2004.
7. P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proc. of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2002.
8. M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:GC11–17, 1996.
9. A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *Journal on Discrete Mathematics*, 12:91–110, 1999.
10. I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. In *Proc. of 5th International Workshop on Algorithms in Bioinformatics*, volume 3692 of *Lecture Notes in Bioinformatics*, pages 204–215. Springer-Verlag, 2005.
11. N. Eriksen. $(1 + \epsilon)$ -approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517–529, 2002.
12. Q.-P. Gu, S. Peng, and I.H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.
13. S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Journal of the ACM*, 48:1–27, 1999.
14. T. Hartman. A simpler 1.5-approximation algorithm for sorting by transpositions. In *Proc. of the 14th Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 156–169. Springer-Verlag, 2003.
15. T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. In *Proc. of 4th International Workshop on Algorithms in Bioinformatics*, volume 3240 of *Lecture Notes in Bioinformatics*, pages 50–61. Springer-Verlag, 2004.
16. H. Kaplan and E. Verbin. Efficient data structures and a new randomized approach for sorting signed permutations by reversals. In *Proc. of 14th Symposium on Combinatorial Pattern Matching*, volume 2676 of *Lecture Notes in Computer Science*, pages 170–185. Springer-Verlag, 2003.
17. G.-H. Lin and G. Xue. Signed genome rearrangement by reversals and transpositions: Models and approximations. *Theoretical Computer Science*, 259(1-2):513–531, 2001.
18. J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, Boston, M.A., 1997.
19. E. Tannier and M.-F. Sagot. Sorting by reversals in subquadratic time. In *Proc. of the 15th Annual Symposium on Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2004.
20. M.E.T. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *Proc. of the Symposium on String Processing and Information Retrieval*, pages 96–102. IEEE Computer Society, 1998.