

# $(1+\varepsilon)$ -approximation of Sorting by Reversals and Transpositions

Niklas Eriksen

*Dept. of Mathematics, Royal Institute of Technology  
S-100 44 Stockholm, Sweden*

*Key words:* permutations, sorting, genome rearrangements, reversals,  
transpositions, approximation algorithm

---

## 1 Introduction

This paper is concerned with the problem of sorting permutations using long range operations like inversions (reversing a segment) and transpositions (moving a segment). The problem comes from computational molecular biology, where the aim is to find a parsimonious rearrangement scenario that explains the difference in gene order between two genomes. In the late eighties, Palmer and Herbon [15] found that the number of such operations needed to transform the gene order of one genome into the other could be used as a measure of the evolutionary distance between two species.

The kinds of operations we consider are inversions, transpositions and inverted transpositions. Hannenhalli and Pevzner [13] showed that the problem of finding the minimal number of inversions needed to sort a signed permutation is solvable in polynomial time, and improved algorithms have subsequently been given by Berman and Hannenhalli [5], Kaplan et al. [14], Moret et al. [1] and Bergeron [4]. Caprara, on the other hand, showed that the corresponding problem for unsigned permutations is NP-hard [9]. In fact, it can not be approximated within 1.0008 [7]. The best known approximation is a 1.375-approximation by Berman et al. [6]. For transpositions no such sharp results are known, but the  $(3/2)$ -approximation algorithms of Bafna and Pevzner [2] and Christie [10] are worth mentioning.

Moving on to the combined problem, Gu et al. [12] gave a 2-approximation algorithm for the minimal number of operations needed to sort a signed permutation by inversions, transpositions and inverted transpositions. There are also some known heuristics. The computer program Derange II by Blanchette

and Sankoff [8] is built on a greedy algorithm which attempts to minimise the *weighted* sum of the number of operations. Varying the weights gives solutions with varying ratio between the numbers of inversions and transpositions. Using simulations, Eriksen et al. [11] found the optimal weights (i.e. introducing least bias) to be 1.0 for inversions and 2.0 for transpositions (including inverted transpositions), mirroring the fact that in the generic case, the optimal inversion will remove one breakpoint while the optimal transposition removes two breakpoints.

In the light of this, it is clear that an algorithm looking for the minimal number of operations will produce a solution heavily biased towards transpositions. Instead, we propose the following problem: find the  $\pi$ -sorting scenario  $s$  (i.e. transforming  $\pi$  to the identity) that minimises  $inv(s) + 2 trp(s)$ , where  $inv(s)$  and  $trp(s)$  are the numbers of inversions and transpositions (including inverted transpositions) in  $s$ , respectively.

We give a closed formula for this minimal weighted distance. Our formula is similar to the exact formula for the inversion case, given by Hannenhalli and Pevzner [13]. We also show how to obtain a polynomial time algorithm for computing this formula with an accuracy of  $(1 + \varepsilon)$ , for any  $\varepsilon > 0$ . As an example, we explicitly state a 7/6-approximation. We also argue that for most applications the algorithm performs much better than guaranteed.

## 2 Preliminaries

Here we present some useful definitions from Bafna and Pevzner [2] and Hannenhalli and Pevzner [13], as well as a couple of new ones.

In this paper, we work with **signed, circular permutations**. We adopt the convention of reading the circular permutations counterclockwise. We linearise the permutation by inverting both signs and reading direction if it contains -1 (inverting a complete genome will of course give an equivalent genome), then making a cut in front of 1 and finally adding  $n + 1$  last, where  $n$  is the length of the permutation. An example is shown in Figure 1.

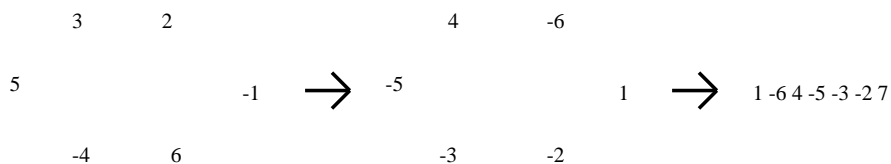


Figure 1. Transforming a circular permutation to a linear form.

A **breakpoint** in a permutation is a pair of adjacent genes that are not adja-

cent in a given reference permutation. For instance, if we compare a genome to the identity permutation and consider the linearised version of the permutation, the pair  $(\pi_i, \pi_{i+1})$  is a breakpoint if and only if  $\pi_{i+1} - \pi_i \neq 1$ . For unsigned permutations, this would be written  $|\pi_{i+1} - \pi_i| \neq 1$ . As an example, we see that the permutation 1 -6 4 -5 -3 -2 7 contains 5 breakpoints.

The three operations we consider are inversions, transpositions and inverted transpositions. These are defined in Figure 2.

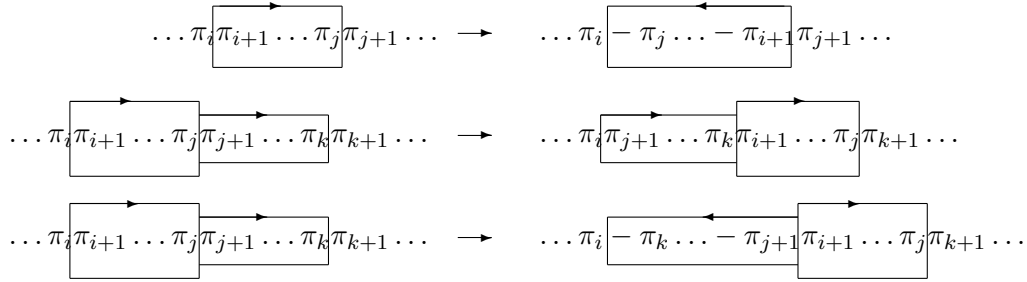


Figure 2. Definitions of inversion, transposition and inverted transposition on *signed* genomes. If we remove all signs, the definitions holds for *unsigned* genomes.

Following [2,13,14], we transform a signed, circular permutation  $\pi$  on  $n$  elements to an unsigned, circular permutation  $\pi'$  on  $2n$  elements as follows. Replace each element  $x$  in  $\pi$  by the pair  $(2x - 1, 2x)$  if  $x$  is positive and by the pair  $(-2x, -2x - 1)$  otherwise. An example can be viewed in Figure 3. Then, to each operation in  $\pi$  there is a corresponding operation in  $\pi'$ , where the cuts are placed after even positions. We also see that the number of breakpoints in  $\pi$  equals the number of breakpoints in  $\pi'$ .

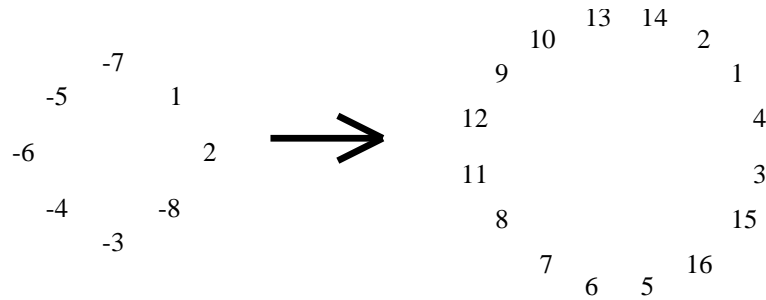


Figure 3. Transforming a signed permutation of length 8 to an unsigned permutation of length 16.

Define the **breakpoint graph** on  $\pi'$  by adding a black edge between  $\pi'_i$  and  $\pi'_{i+1}$  if there is a breakpoint between them and a grey edge between  $2i$  and  $2i+1$ , unless these are adjacent (Figure 4). These edges will then form **cycles**, with alternating edge colours. The **length** of a cycle is the number of black edges in it. Sometimes we will also draw black and grey edges between  $2i$  and  $2i+1$ , even though these are adjacent. We will then get cycles of length

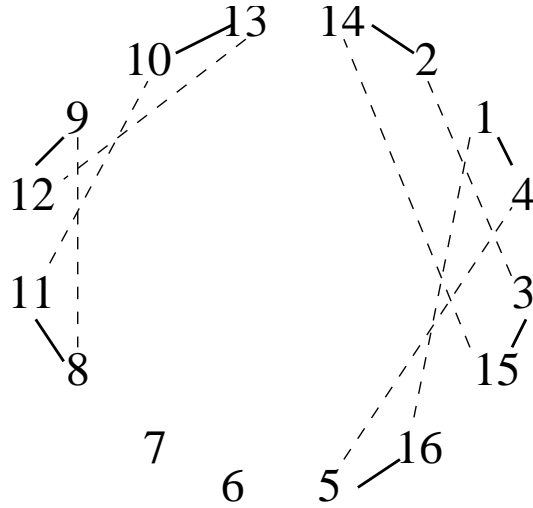


Figure 4. The breakpoint graph of a transformed permutation. It contains three cycles, two of length 2 and one of length 3. The latter constitutes one component, and the first two constitute another component. The first component is unoriented and the second is oriented. Both components have size 2.

one at the places where we do not have a breakpoint in  $\pi$  (these cycles will be referred to as **short cycles**). A cycle is **oriented** if, when we traverse it, at least one black edge is traversed clockwise and at least one black edge is traversed counterclockwise. Otherwise, the cycle is **unoriented**.

Consider two cycles  $c_1$  and  $c_2$ . If we can not draw a straight line through the circle such that the elements of  $c_1$  are on one side of the line and the elements of  $c_2$  are on the other side, then these two cycles are **inseparable**. This relation is extended to an equivalence relation by saying that  $c_1$  and  $c_j$  are in the same **component** if there is a sequence of cycles  $c_1, c_2, \dots, c_j$  such that, for all  $1 \leq i \leq j - 1$ ,  $c_i$  and  $c_{i+1}$  are inseparable.

A component is **oriented** if at least one of its cycles is oriented and **unoriented** otherwise. If there is an interval on the circle, which contains an unoriented component, but no other unoriented components, then this component is a **hurdle**. If we cannot remove a hurdle without creating a second hurdle upon its removal (this is the case if there is an unoriented component, which is not a hurdle, that stretches over an interval that contains the previously mentioned hurdle, but no other hurdles), then the hurdle is called a **super hurdle** (Figure 5). If we have an odd number of super hurdles and no other hurdles, the permutation is known as a **fortress**.

We should observe that for components in the breakpoint graph, the operations needed to remove them do not depend on the actual numbers on the vertices. We could therefore treat the components as separate objects, disregarding the particular permutation they are part of. If we wish, we can also

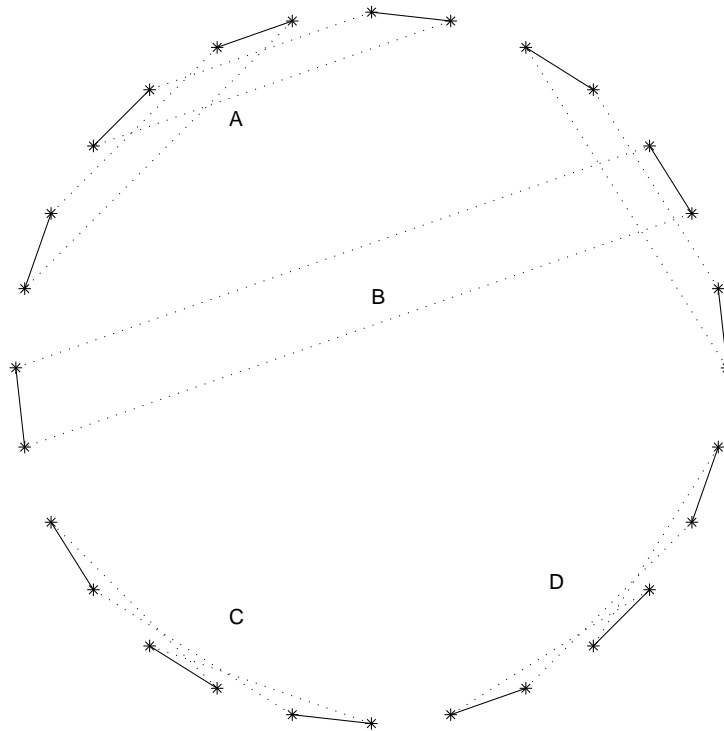


Figure 5. The breakpoint graph of the permutation 1 14 9 12 11 10 13 2 4 3 5 7 6 8 15. The graph contains four unoriented components, of which three (A, C and D) are hurdles. A is a super hurdle, since removing it will turn B into a hurdle. If we remove C, D will become a super hurdle (and vice versa).

regard the components as permutations, by identifying them with (one of) the shortest permutations whose breakpoint graphs consist of this component only. For example, the 2-cycle component in Figure 4 can be identified with the permutation 1 -3 -4 2 5.

We say that an unoriented component is of **odd length** if all cycles in the component are of odd length. We let  $b(\pi)$ ,  $c(\pi)$  and  $h(\pi)$  denote the number of breakpoints, cycles (not counting cycles of length one) and hurdles in the breakpoint graph of a permutation  $\pi$ , respectively. For components  $t$ ,  $b(t)$  and  $c(t)$  are defined similarly. The **size** of a component  $t$  is given by  $s(t) = b(t) - c(t)$ . We also let  $c_s(\pi)$  denote the number of cycles in  $\pi$ , including the short ones, and  $f(\pi)$  is the characteristic function of the fortress property, i.e.  $f(\pi)$  is 1 if  $\pi$  is a fortress, and 0 otherwise.

### 3 Expanding the inversion formula

Let  $S_\pi^I$  denote the set of all scenarios transforming  $\pi$  into  $id$  using inversions only and let  $inv(s)$  denote the number of inversions in a scenario  $s$ . The inversion distance is defined as  $d_{Inv}(\pi) = \min_{s \in S_\pi^I} \{inv(s)\}$ . It has been shown in [13,14] that

$$d_{Inv}(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi),$$

where  $b(\pi), c(\pi), h(\pi)$  and  $f(\pi)$  have been defined in the previous paragraph.

In this paper, we define the distance between  $\pi$  and  $id$  by

$$d(\pi) = \min_{s \in S_\pi} \{inv(s) + 2 trp(s)\},$$

where  $S_\pi$  is the set of all scenarios transforming  $\pi$  into  $id$ , allowing both inversions and transpositions, and  $inv(s)$  and  $trp(s)$  is the number of inversions and transpositions in scenario  $s$ , respectively. Here, transpositions refer to both ordinary and inverted transpositions.

In order to give a formula for this distance, we need a few definitions.

**Definition 1** *Regard all components  $t$  as permutations and let  $d(t)$  be the distance between  $t$  and  $id$  as defined above for permutations. Consider the set  $S$  of components  $t$  such that  $d(t) > s(t) = b(t) - c(t)$  (when using inversions only, this is the set of unoriented components). We call this set the set of **strongly unoriented components**. If there is an interval on the circle that contains the component  $t \in S$ , but no other member of  $S$ , then  $t$  is a **strong hurdle**. **Strong super hurdles** and **strong fortresses** are defined in the same way as super hurdles and fortresses (just replace hurdle with strong hurdle).*

**Observation 2** *In any scenario, each inverted transposition can be replaced by two inversions, without affecting the objective function. This means that in calculating  $d(\pi)$ , we need not bother with inverted transpositions. Therefore, we will henceforth consider only inversions and ordinary transpositions.*

**Lemma 3** *Each strongly unoriented component is unoriented (in the inversion sense).*

**PROOF.** We know that for oriented components  $t$ ,  $d_{inv}(t) = b(t) - c(t)$  and for any permutation  $\pi$ , we have  $d(\pi) \leq d_{Inv}(\pi)$ . Regarding the component  $t$  as a permutation gives  $d(t) \leq d_{Inv}(t)$ . Thus, for strongly unoriented components we have  $d_{inv}(t) \geq d(t) > b(t) - c(t)$  and we can conclude that a strongly unoriented component can not be oriented.

**Theorem 4** *The distance  $d(\pi)$  defined above is given by*

$$d(\pi) = b(\pi) - c(\pi) + h_t(\pi) + f_t(\pi),$$

*or, equivalently (counting short cycles as well),*

$$d(\pi) = n - c_s(\pi) + h_t(\pi) + f_t(\pi),$$

*where  $h_t(\pi)$  is the number of strong hurdles in  $\pi$ ,  $f_t(\pi)$  is 1 if  $\pi$  is a strong fortress (and 0 otherwise) and  $n$  is the length of the permutation  $\pi$ .*

**PROOF.** First, let us agree that the two expressions are really equivalent. Let  $sc(\pi)$  denote the number of cycles of length one in  $\pi$ . Since we have a breakpoint exactly where we do not have a short cycle, we get

$$b(\pi) - c(\pi) = n - sc(\pi) - c(\pi) = n - c_s(\pi).$$

It is easy to see that  $d(\pi) \leq n - c_s(\pi) + h_t(\pi) + f_t(\pi)$ . If we treat the strong hurdles as in the inversion case, we need only  $h_t(\pi) + f_t(\pi)$  inversions to make all strongly unoriented components oriented. All oriented components can be removed efficiently using inversions, and the unoriented components which are not strongly unoriented can, by definition, be removed efficiently.

We now need to show that we can not do better than the formula above. From Hannenhalli and Pevzner we know that we can not decrease  $n - c_s(\pi)$  by more than 1 using an inversion. The reason is that an inversion cuts in two places, leaving four loose ends to be tied up in pairs. This gives at most two cycles, and we began by splitting at least one. Similarly, a transposition will never decrease  $n - c_s(\pi)$  by more than 2, which is obtained by splitting a cycle into three cycles. The question is whether transpositions can help us to remove strong hurdles more efficiently than inversions.

Bafna and Pevzner have shown that applying a transposition can only change the number of cycles by 0 or  $\pm 2$ . There are thus three possible ways of applying a transposition. First, we can split a cycle into three parts ( $\Delta c_s = 2$ ). If we do this to a strong hurdle, at least one of the components we get must by definition remain a strong hurdle, since otherwise the original component could be removed efficiently. This gives  $\Delta h_t \geq 0$ . Second, we can let the transposition cut two cycles ( $\Delta c_s = 0$ ). To decrease the distance by three, we would have to decrease the number of strong hurdles by three which is clearly out of reach (only two strong hurdles may be affected by a transposition on two cycles). Finally, if we merge three cycles ( $\Delta c_s = -2$ ), we would need to remove five strong hurdles. This clearly is impossible.

It is conceivable that the fortress property could be removed by a transposition that reduce  $n - c_s(\pi) + h_t(\pi)$  by two and at the same time removes an odd number of strong super hurdles or adds a strong hurdle that is not a strong super hurdle. However, from the analysis above, we know that the transpositions that decrease  $n - c_s(\pi) + h_t(\pi)$  by two must decrease  $h_t(\pi)$  by an even number. We also found that when this was achieved, no other hurdles apart from those removed were affected. Hence, there are no transpositions that reduce  $n - c_s(\pi) + h_t(\pi) + f_t(\pi)$  by three.

We find that  $d(\pi) \geq n - c_s(\pi) + h_t(\pi) + f_t(\pi)$ , and in combination with the first inequality,  $d(\pi) = n - c_s(\pi) + h_t(\pi) + f_t(\pi)$ .

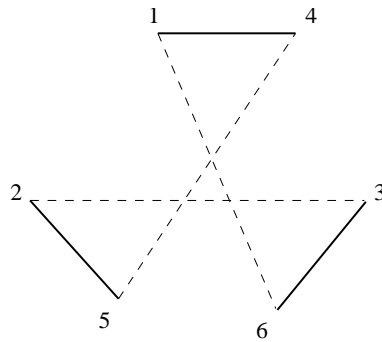


Figure 6. The breakpoint graph of a cycle of length three which can be removed by a single transposition.

### 3.1 The strong hurdles

Once we have identified all strongly unoriented components in a breakpoint graph, we are able to calculate the number of strong hurdles. We thus need to look into the question of determining which components are strongly unoriented.

From the lemma above, we found that all strongly unoriented components are unoriented. The converse is not true. One example of this is the unoriented cycle in Figure 6, which can be removed with a single transposition. However, many unoriented components are also strongly unoriented. Most of them are characterised by the following lemma.

**Lemma 5** *If an unoriented component contains a cycle of even length, then it is strongly unoriented.*

**PROOF.** Since the component is unoriented, applying an inversion to it will not increase the number of cycles. If we apply a transposition to it, it will

remain unoriented. Thus, the only way to remove it efficiently would be to apply a series of transpositions, all increasing the number of cycles by two.

Consider what happens if we split a cycle of even length into three cycles. The sum of the length of these three new cycles must equal the length of the original cycle, in particular it must be even. Three odd numbers never add to an even number, so we must still have at least one cycle of even length, which is shorter than the original cycle.

Eventually, the component must contain a cycle of length 2. There are no transpositions reducing  $b(t) - c(t)$  by 2 that can be applied to this cycle, and hence the component is strongly unoriented.

Concentrating on the unoriented components with cycles of odd lengths only, we find that some of these are strongly unoriented and some are not. For instance, there are two unoriented cycles of length three. One of them is the cycle in which we may remove three breakpoints (Figure 6) and the other one can be seen in Figure 7 (a). Note that this cycle can not be a component. This is, however, not true for the components in Figure 7 (b) and (c), which are the two smallest strongly unoriented components of odd length.

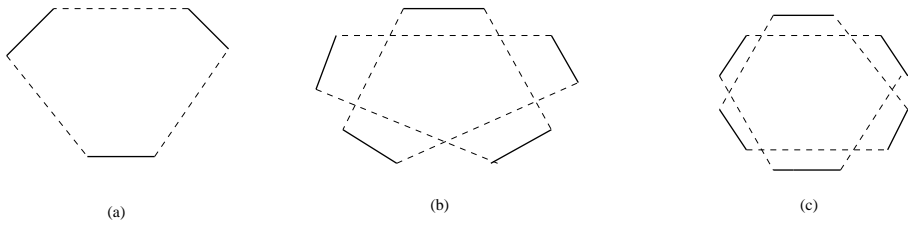


Figure 7. A cycle of length three which can not be removed by a transposition (a), the smallest strongly unoriented component of odd length (b) and the second smallest strongly unoriented component of odd length (c).

#### 4 The (7/6)-approximation and the $(1 + \varepsilon)$ -approximation of the distance

Even though we at this stage are unable to recognise all strongly unoriented components in an efficient manner, we are still able to approximate the distance reasonably well. We will first show that our identification of the two strongly unoriented components of size less than 6 that contain odd cycles exclusively will give us a 7/6-approximation (remember that the size of a component  $t$  was defined as  $b(t) - c(t)$ ). We will then show that if we have identified all odd strongly unoriented components of size less than  $k$ , we can make a  $(1 + \varepsilon)$ -approximation for  $\varepsilon = 1/k$ .

First we look at the case when we know for sure that  $\pi$  is not a strong fortress, and then we look at the case when  $\pi$  may be a strong fortress.

*4.1 If  $\pi$  is not a strong fortress, then we have a 7/6-approximation*

For all odd unoriented components with size less than 6, we are able to distinguish between those that are strongly unoriented and those that are not. In fact, by exhaustive search, we have found that the only strongly unoriented components in this set are the components in Figure 7 (b) and (c). Thus, the smallest components that may be wrongly deemed as strong hurdles are those of size 6.

Let  $h_u(\pi)$ ,  $c_u(\pi)$  and  $b_u(\pi)$  be the number of components, the number of cycles and the number of breakpoints among the odd unoriented components of size 6 or larger, respectively. It is clear that  $h_u(\pi) \leq c_u(\pi)$  and  $h_u(\pi) \leq \frac{b_u(\pi) - c_u(\pi)}{6}$ . Let  $b_o(\pi)$  and  $c_o(\pi)$  denote the number of breakpoints and cycles, respectively, among all other components (that is, the components that we know whether they are strongly unoriented or not). Also, let  $h_{none}(\pi)$  denote the number of hurdles we would have if none of the large odd unoriented components are strongly unoriented and let  $h_{all}(\pi)$  denote the number of hurdles we would have, if all of these are strongly unoriented. It follows that  $h_{none}(\pi) \leq h_t(\pi) \leq h_{all}(\pi) \leq h_{none}(\pi) + h_u(\pi)$ . This gives

$$\begin{aligned} d(\pi) &= b(\pi) - c(\pi) + h_t(\pi) \\ &\leq b_o(\pi) + b_u(\pi) - c_o(\pi) - c_u(\pi) + h_{all}(\pi) \\ &\leq b_o(\pi) - c_o(\pi) + b_u(\pi) - c_u(\pi) + h_{none}(\pi) + h_u(\pi) \\ &\leq b_o(\pi) - c_o(\pi) + b_u(\pi) - c_u(\pi) + h_{none}(\pi) + \frac{b_u(\pi) - c_u(\pi)}{6} \end{aligned}$$

and

$$\begin{aligned} d(\pi) &= b(\pi) - c(\pi) + h_t(\pi) \\ &\geq b_o(\pi) + b_u(\pi) - c_o(\pi) - c_u(\pi) + h_{none}(\pi) \end{aligned}$$

and hence (putting  $d_o(\pi) = b_o(\pi) - c_o(\pi) + h_{none}(\pi)$ )

$$d(\pi) \in \left[ d_o(\pi) + b_u(\pi) - c_u(\pi), d_o(\pi) + \frac{7(b_u(\pi) - c_u(\pi))}{6} \right].$$

In most situations,  $d_o(\pi)$  will be quite large compared to  $b_u(\pi) - c_u(\pi)$  and then the approximation is much better than 7/6. Thus, in practice we may use this algorithm to get a reliable value for  $d(\pi)$ .

4.2 *If  $\pi$  may be a strong fortress, then we still have a  $7/6$ -approximation*

The analysis is similar to the one in the previous case. To simplify things a bit, we look at the worst case. The effect of  $\pi$  being a strong fortress is most significant if  $d(\pi)$  is small. We need an odd number of strong super hurdles, and no other strong hurdles, to make a strong fortress. It takes two strong hurdles to form a strong super hurdle, and one strong super hurdle can not exist by itself. Thus, we need at least six strongly unoriented components, arranged in pairs, covering disjoint intervals of the circle.

Let  $h_\pi(t)$  be 1 if the component  $t$  is a hurdle in  $\pi$  and 0 otherwise. We consider the case where we have six components, arranged such that we have three possible strong super hurdles. For each of these three pairs, there are three possible cases. If we know that we have a strong super hurdle, then we know that, for each component  $t$ ,  $b(t) - c(t) \geq 2$  (there are no components  $t$  with  $b(t) - c(t) < 2$ ). Thus, for the pair of components  $t$  and  $t'$ , we have  $b(t) - c(t) + h(t) + b(t') - c(t') + h(t') \geq 5$ . If we know that one of the two components is strongly unoriented, but we are not sure about the other, then we know that we have a strong hurdle and for the second component  $t'$ , we know that  $b(t') - c(t') \geq 6$ . Together this gives  $b(t) - c(t) + h(t) + b(t') - c(t') + h(t') \geq 9$ . Finally, if we are ignorant to whether any of the two components are strongly unoriented, we do not even know whether the pair constitutes a strong hurdle. Since both components fulfill  $b(t) - c(t) \geq 6$ , we get  $b(t) - c(t) + h(t) + b(t') - c(t') + h(t') \in [r, r + 1]$ , where  $r \geq 12$ . The worst cases is when we are totally ignorant for each of the three pairs and  $r = 12$ . In that case, we get

$$d(\pi) \in [3 \cdot 12, 3 \cdot 13 + 1] = [36, 40],$$

and since this is the worst case, we have a  $(10/9)$ -approximation, which is better than  $7/6$ . Again, this ratio will be significantly smaller in most applications.

4.3 *The  $(1 + \varepsilon)$ -approximation*

In order to improve on the  $(7/6)$ -approximation, we need to be able to identify strong hurdles among larger components. Since we have not yet found an easy way to do this, we content ourselves with creating a table of all unoriented components up to a certain size, which are not strongly unoriented. The table could be created using, for instance, an exhaustive search.

Given a table of all such components of size less than  $k$ , and a component  $t$  of size less than  $k$ , we will be able to tell if  $t$  is strongly unoriented or not. Thus, applying the same calculations as in the  $7/6$  case above, we find that (if  $\pi$  is

not a strong fortress)

$$d(\pi) \in \left[ d_o(\pi) + b_u(\pi) - c_u(\pi), d_o(\pi) + \frac{(k+1)(b_u(\pi) - c_u(\pi))}{k} \right],$$

or (if  $\pi$  may be a strong fortress, worst case (for  $k > 10$ , the worst case is different from the worst case for  $k = 6$ ))

$$d(\pi) \in [2k + 2 \cdot 5, 2k + 1 + 2 \cdot 5 + 1],$$

We clearly have

$$\lim_{k \rightarrow \infty} \frac{k+1}{k} = \lim_{k \rightarrow \infty} 1 + \frac{1}{k} = 1$$

and

$$\lim_{k \rightarrow \infty} \frac{2k+12}{2k+10} = \lim_{k \rightarrow \infty} 1 + \frac{1}{k+5} = 1.$$

## 5 The algorithm

We will now describe the  $(7/6)$ -approximation algorithm, which is easily generalised to the  $1 + \varepsilon$  case. First remove, by applying a sequence of optimal transpositions, all odd unoriented components of size less than 6, that are not strongly unoriented. This can be done as follows: Find a black edge such that its two adjacent grey edges are crossing. For these small components, this can always be done. Cut this black edge and the two black edges that are adjacent to the mentioned grey edges. This transposition will always reduce the distance by two, and for these components, we can always continue afterwards in a similar fashion. In the  $1 + \varepsilon$  case, we would have to use the table to find out which transpositions to use.

After removing these components, we can apply the inversion algorithm of Hannenhalli and Pevzner. The complexity of this algorithm is polynomial in the length of the original permutation, as is the first step of our algorithm, since identifying the unoriented components that are not strongly unoriented and removing them can be done in linear time. Computing the inversion distance can be done in linear time [1] and thus an approximation of the combined distance can also be computed in linear time.

To get a  $(1 + \varepsilon)$ -approximation, all we have to do is to tabulate all odd unoriented components of size  $\frac{1}{\varepsilon}$ , that are not strongly unoriented. We also need to tabulate, for each such component, a sequence of transpositions that will remove the component efficiently. It is clear that the algorithm is still polynomial, since looking up a component in the table is done in constant time (for each  $\varepsilon$ ), although the table will probably grow exponentially with  $1/\varepsilon$ .

## 6 Discussion

The algorithm presented here relies on the creation of a table of components that can be removed efficiently. Could this technique be used to find an algorithm for **any** similar sorting problem such as sorting by transpositions? In general, the answer is no. In this case, as for sorting with inversions, we know that if a component can not be removed efficiently, we need only one extra inversion. We also know that for components that can be removed efficiently, we can never improve on such a sorting by combining components. For sorting by transpositions, no such results are known and until they are, the table will need to include not only some of the components up to a certain size, but every permutation of every size.

The next step is obviously to examine if there is an easy way to distinguish all strongly unoriented components. For odd unoriented components, this property seems very elusive. It also seems hard to discover a useful sequence of transpositions that removes odd oriented components that are not strongly unoriented. However, investigations on small components have given very promising results. For cycles of length 7, we have the following result: If the cycle is not a strongly unoriented component, then no transposition that increase the number of cycles by two will give a strongly unoriented component. This appears to be the case for cycles of length 9 as well, but no fully exhaustive search has been conducted, due to limited computational resources.

If this pattern would hold, we could apply any sequence of breakpoint removing transpositions to a component, until we either have removed the component, or are unable to find any useful transpositions. In the first case, the component is clearly not strongly unoriented, and in the second case it would be strongly unoriented.

## 7 Acknowledgment

I wish to thank my advisor Kimmo Eriksson for valuable comments during the preparation of this paper.

Niklas Eriksen was supported by the Swedish Natural Science Research Council and the Swedish Foundation for Strategic Research.

## References

- [1] Bader, D. A., Moret, B. M. E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology* **8**, 5 (2001), 483–491
- [2] Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals. *Proceedings of the 34th IEEE symposium of the foundations of computer science* (1994), 148–157
- [3] Bafna, V., Pevzner, P.: Sorting by Transpositions. *SIAM Journal of Discrete Mathematics* **11** (1998), 224–240
- [4] Bergeron, A.: A Very Elementary Presentation of the Hannenhalli-Pevzner Theory. *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching* (2001), 106–117
- [5] Berman, P., Hannenhalli, S.: Fast sorting by reversals. *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching* (1996), 168–185
- [6] Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation Algorithm for Sorting by Reversals. *Electronic Colloquium for Computational Complexity* TR01-047 (2001)
- [7] Berman, P., Karpinski, M.: On some tighter inapproximability results. *Proceedings of the 26th International Colloquium on Automata, Languages and Programming, LNCS 1644* (1999), 200–209
- [8] Blanchette, M., Kunisawa, T., Sankoff, D.: Parametric genome rearrangement. *Gene* **172** (1996), GC 11–17
- [9] Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal of Discrete Mathematics* **12** (1999), 91–110
- [10] Christie, D. A.: Genome rearrangement problems. Ph. D. thesis (1998)
- [11] Eriksen, N., Dalevi, D., Andersson, S. G. E., Eriksson, K.: Gene order rearrangements with Derange: weights and reliability. Preprint
- [12] Gu, Q.-P., Peng, S., Sudborough, H.: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science* **210** (1999), 327–339
- [13] Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations with reversals). *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing* (1995), 178–189
- [14] Kaplan, H., Shamir, R., Tarjan, R. E.: Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Mathematics* (1997), 344–351
- [15] Palmer, J. D., Herbon, L. A.: Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution* **28** (1988), 87–97