

## ABSTRACT

### Optimization Problems from Genome Sequence Rearrangement

Devin Henson

A problem of great interest is the problem of computing the evolutionary distance between two organisms using their genomic data. We model a genome as a permutation with the genes as elements. Given two permutations, we survey the known results for the shortest sequence of rearrangement operations that transforms one permutation into the other. Most results come in the form of approximation algorithms which sort a given permutation within a small multiplicative constant of the minimal sorting distance. The approximation algorithms given are helpful due to their low computational complexity. More specifically, the algorithms to be examined in this paper run in polynomial time. The rearrangement operations we will consider are transpositions and reversals within a permutation. The transposition material comes from papers by I. Elias and T. Hartman, as well as T. Hartman and R. Shamir. The reversals material comes from papers by S. Hannenhalli and P. Pevzner, and A. Bergeron et al.

Dissertation Director: Dr. László Székely

# Optimization Problems from Genome Sequence Rearrangement

by

Devin Henson

Bachelor of Science  
College of Charleston, 2004

---

Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in the

Department of Mathematics

College of Arts and Sciences

University of South Carolina

2006

---

Department of Mathematics  
Director of Thesis

---

Department of Mathematics  
2<sup>nd</sup> Reader

---

Dean of The Graduate School

## DEDICATION

- To Jesus Christ, the Creator and Almighty God.

- I also dedicate this to my wife Kathy, whose patient love enabled me to complete this work. Also, to Mom, Dad, Caity, and Spencer for their constant love and support all these years.

“...and out of the ground the Lord formed every beast of the field and bird of the air.”

## ACKNOWLEDGEMENTS

I give my deepest thanks to Dr. László Székely for his help and guidance throughout this process. Without his help this would never have been possible. I would like to thank Dr. Daniel Dix for his helpfulness and insight. In addition, I would like to thank my friend Yiting Yang for his help in understanding this material and for letting me ask him a multitude of questions. Lastly, thanks to Dr. George McNulty for his clear explanation of the intricacies of  $\text{\LaTeX}$ .

## ABSTRACT

A problem of great interest is the problem of computing the evolutionary distance between two organisms using their genomic data. We model a genome as a permutation with the genes as elements. Given two permutations, we survey the known results for the shortest sequence of rearrangement operations that transforms one permutation into the other. Most results come in the form of approximation algorithms which sort a given permutation within a small multiplicative constant of the minimal sorting distance. The approximation algorithms given are helpful due to their low computational complexity. More specifically, the algorithms to be examined in this paper run in polynomial time. The rearrangement operations we will consider are transpositions and reversals within a permutation. The transposition material comes from papers by I. Elias and T. Hartman, as well as T. Hartman and R. Shamir. The reversals material comes from papers by S. Hannenhalli and P. Pevzner, and A. Bergeron et al.

# CONTENTS

Dedication	ii
Acknowledgements	iii
Abstract	iv
List of Figures	vi
Chapter 1 Introduction	1
Chapter 2 Sorting by transpositions	3
2.1. 1.5-approximation	3
2.2. 1.375-approximation	15
2.3. Difficultly computing exact transposition distance	19
Chapter 3 Sorting by reversals	20
3.1. Reversals	20
3.2. Closed form for reversal distance of signed permutations	22
3.3. Reversal distance without hurdles and fortresses	25
Chapter 4 Remaining issues in sorting genomic material	35
Bibliography	37

## LIST OF FIGURES

1	An example of a transposition on contiguous segments A and B for a linear (top) and circular (bottom) permutation.	5
2	Breakpoint graph for the circular permutation $\pi = (1, 3, 5, 2, 4)$	6
3	(a) Breakpoint graph for $\pi = (1, 3, 5, 2, 4)$ (b) Breakpoint graph of $\rho\pi$ after the transposition $\rho$ has acted on $\pi$ before positions 2,4, and 5 (which corresponds to the starred black edges)	7
4	$G(id)$ - The breakpoint graph for the identity permutation (on 8 elements).	8
5	How to split a cycle of length $k$ into a 3-cycle and a $(k - 2)$ -cycle. A dashed line indicates a path and $g$ represents a grey edge while $b$ represents a black edge.	10
6	The only two possible configurations of 3-cycles. The first one is oriented, while the second is unoriented	11
7	(a) Two intersecting three cycles. Here we see that the pair of black edges $(p, q)$ intersects with the pair of black edges $(r, s)$ , so by definition the cycles intersect (b) Two interleaving 3-cycles.	12
8	(a) A 2-transposition on a permutation that contains a 2-cycle (b) A $(0,2,2)$ -sequence of transpositions on two interleaving 3-cycles. (c) A $(0,2,2)$ -sequence of transpositions on three unoriented 3-cycles.	13

9	(a) A configuration, which is impossible as a breakpoint graph, but is a subgraph of (b). (b) A breakpoint graph which includes (a) as a subgraph. Note that there are no open gates in (b).	14
10	A 1.5-approximation algorithm for sorting by transpositions due to Hartman and Shamir [HS03].	16
11	Elias and Hartman's algorithm 1.375 – <i>Sort</i> .	18
12	Examples of sorting by reversals. (a)- unsigned permutation (b)-signed permutation	21
13	Hurdles and Non-hurdles	23
14	A fortress.	24
15	Elementary intervals for $P_1$ . Elementary intervals are shown with solid horizontal lines. The cycles the intervals generate are completed with vertical dotted lines.	26
16	Components in the permutation $P_2$ .	27
17	The tree $T_{P_2}$ associated with permutation $P_2$ .	29

# CHAPTER 1

## INTRODUCTION

There are certain events that can affect the genome of an organism, which could subsequently affect future offspring. Over a period of time, we can ask how close two organisms are under a distance of some sort. Recently it has become possible to map an organism's entire genome. While two beings will not have the same set of genes, the set of common genes may be large. The larger the overlapping, the closer the two organisms' distance. In these models we neglect the symmetric difference. Certain genomes are linear, and others are circular. We model them as permutations (circular permutations) of their gene set. Moreover, for most of the thesis, we assume that the two genomes whose genetic distance is under study, have the same gene set, and they are both represented as a permutation or circular permutation of the same gene set.

There are rearrangement operations that can occur which can rearrange the sequence of genes from one organism to be the same as that of another organism. These naturally occurring operations include transpositions, reversals, and insertions/deletions. The basic idea is the fewer the rearrangement operations it takes to transform one organism's genome into another, the closer is their evolutionary distance. It should be noted that the best we can do is to estimate the optimal number of rearrangement operations needed. For example, a segment of DNA could be reversed, then later reversed again in the same spot and these two operations that occurred would never be discoverable.

For these processes in which we would like to know the minimal number of steps needed to sort two permutations, known as a permutation's *minimal sorting*

*distance*, all we need to consider is the minimal distance between any given permutation and the identity permutation *id*. To see this, consider the minimal sorting distance between permutations  $\sigma$  and  $\pi$ . The minimal sorting sequence that sorts  $\sigma$  into  $\pi$  will also sort  $\pi^{-1}\sigma$  into *id* (afterwards just apply  $\pi$  to both permutations).

Therefore, for the rest of the paper we only consider the distance between a given permutation  $\pi$  and the identity permutation. This concept holds true for all of the evolutionary processes considered here. In the next section we delve more in depth into the complexity of sorting genetic material by transpositions.

## CHAPTER 2

### SORTING BY TRANSPOSITIONS

#### 2.1. 1.5-APPROXIMATION

One of the naturally occurring processes that rearranges genomic data within an organism's genetic makeup is the transposition of contiguous segments of DNA. For a transposition, a strand of genetic material is cut at certain points, known as break-points, and then the cut segments are interchanged without reversing direction.

The *evolutionary distance* of two strands of genomic data under a given operation is defined to be the minimal number of steps it would take to transform one sequence into the other using only the specified operation. These operations can be transposition, insertion, deletion, reversal, or a number of naturally occurring evolutionary processes. We will examine the best known estimates for the evolutionary distance between two segments of genetic material under the transposition operation. Later we will examine the evolutionary distance of other evolutionary processes.

To begin we need to take the given genetic data and represent it as a permutation. To do this we represent each gene on a strand of genetic material as a number, then the sequence of numbers gives us a permutation with which to work. The first to study the complexity of sorting a given permutation by transpositions were Bafna and Pevzner [BP98]. The complexity proved too difficult to completely solve the minimal sorting distance of two permutations, but it is not known if this problem is NP-hard. However, in 1998 Bafna and Pevzner did find a rather involved 1.5-approximation algorithm which ran in quadratic time. This means that if the minimal sorting distance is

$d$ , their algorithm does the sorting in at most  $1.5d$  steps. The search for a better or simpler approximation for the transposition distance continued until in 2004 Hartman and Shamir [HS03] gave another 1.5-approximation algorithm which was considerably simpler with a better running time of  $O(n^{3/2}\sqrt{\log n})$ , where  $n$  is the length of the permutation. Still, the search continued further for a better approximation algorithm until in 2005, Elias and Hartman [EH05] gave us our currently best known 1.375-approximation algorithm which uses some old results combined with some new ideas to improve the approximation ratio. Another important component of the 1.375-approximation algorithm is the use of a rigorous case analysis to independently test over 80,000 cases using a computer program.

We will forego the topics in the Bafna and Pevzner paper [BP98] and instead give a sketch of the ideas from the Hartman and Shamir paper [HS03], then also give a sketch of the ideas from the Elias and Hartman paper [EH05].

Let  $\pi=[\pi_1\dots\pi_n]$  be a permutation on  $n$  elements. A transposition acts on contiguous segments in a permutation. Two segments  $A=\pi_i\dots\pi_j$  and  $B=\pi_k\dots\pi_l$  are contiguous if  $j = k - 1$  or  $l = i - 1$ . A transposition  $\rho$  acting on  $\pi$  which transposes contiguous segments  $A$  and  $B$ , with  $j = k - 1$ , gives  $\rho\pi=[\pi_1\dots\pi_{i-1}\pi_k\dots\pi_l\pi_i\dots\pi_j\pi_{l+1}\dots\pi_n]$  (see Figure 1).

Note in the figures above, every transposition is uniquely determined once the positions where the segment is cut are determined. Therefore, a transposition  $\rho$  which cuts before positions  $i, j$ , and  $k$  may be defined as  $\rho(i, j, k)$ . It is a natural question whether sorting linear permutations is equivalent to sorting circular permutations. It turns out that, in fact, the two sortings are equivalent [HS03]. This is a pleasant fact since sorting circular permutations by transpositions is more convenient. Therefore, if we are given a linear permutation, the algorithms mentioned here transform the linear permutation to a circular one, sort the circular permutation, then mimick the sorting of the circular permutation on the original linear permutation.

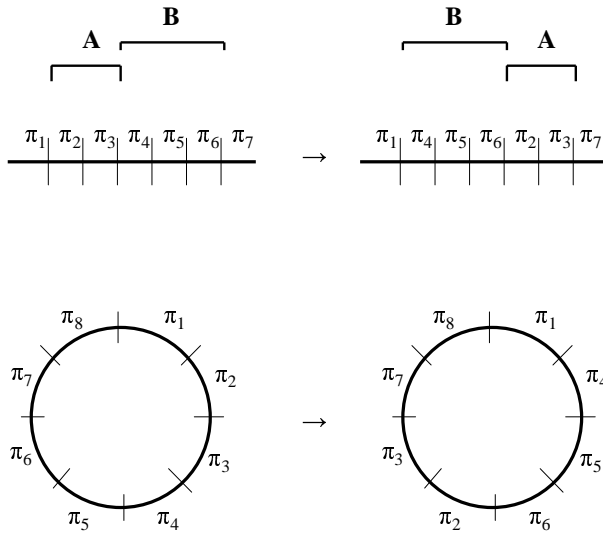


FIGURE 1. An example of a transposition on contiguous segments A and B for a linear (top) and circular (bottom) permutation.

The following useful theorem is due to Hartman and Shamir [HS03]:

**THEOREM 1.** *The problem of sorting linear permutations by transpositions is linearly equivalent to the problem of sorting circular permutations by transpositions.*

**PROOF.** Let  $\pi$  be a linear permutation with  $n$  elements. We can circularize  $\pi$  by adding one extra element  $x = \pi_{n+1}$  which encloses the permutation into a circular permutation. Call this new circular permutation  $\pi^c$ . Note that any transposition on a circular permutation involves three and only three segments, in which two of the segments are transposed. Therefore, any transposition on a circular permutation may be represented by the two segments that do not contain  $x$ . Hence, we can now define the optimal sequence of transpositions that sorts  $\pi^c$  in terms of transpositions such that no transposed segments contain  $x$ . The same sequence can be viewed as a sequence of transpositions on the linear permutation  $\pi$ , by ignoring  $x$ . Therefore we have  $d(\pi) \leq d(\pi^c)$ . On the other hand, any sequence of transpositions on  $\pi$  is also a sequence of transpositions on  $\pi^c$ , so  $d(\pi^c) \leq d(\pi)$ . Hence  $d(\pi) = d(\pi^c)$ . For the other direction, starting with a circular permutation we can linearize it by arbitrarily

$$\pi = (1,3,5,2,4) \rightarrow \text{Vertex set} = \{l_1, r_1, l_2, r_2, l_3, r_3, l_4, r_4, l_5, r_5\}$$

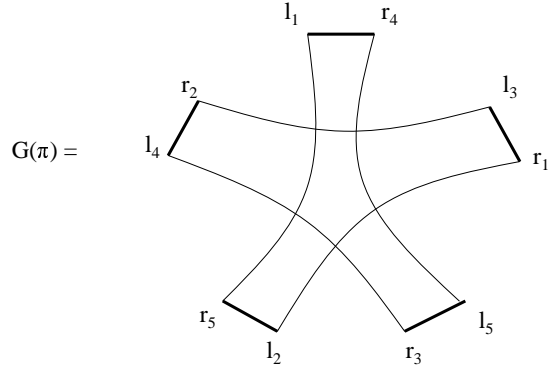


FIGURE 2. Breakpoint graph for the circular permutation  $\pi = (1, 3, 5, 2, 4)$

removing one element, which takes the place of  $x$  above. The same arguments imply that an optimal solution for the linear permutation translates to an optimal solution for the circular one.  $\square$

*Breakpoint Graph.* One of the reasons sorting an  $n$ -element circular permutation is more convenient is that we can usefully represent the circular permutation  $\pi$  by an undirected edge colored graph  $G(\pi)$  with  $2n$  vertices known as the breakpoint graph. All algorithms presented here actually work on the breakpoint graph  $G(\pi)$ ; then we use this information to sort  $\pi$ .

DEFINITION 1. A pair of elements  $(i, i+1)$  is called a *breakpoint* if  $|\pi_{i+1} - \pi_i| \neq 1$ .

Intuitively, a breakpoint in a permutation is a spot where two contiguous elements are not together like they would be in the identity permutation. We now show how to generate the breakpoint graph  $G(\pi)$  for a given permutation  $\pi$ .

DEFINITION 2. Let  $\pi = (\pi_1 \dots \pi_n)$  be an  $n$ -element circular permutation. The breakpoint graph  $G(\pi)$  is an edge-colored graph on  $2n$  vertices  $\{l_1, r_1, l_2, r_2, \dots, l_n, r_n\}$ . The edges are defined as follows:

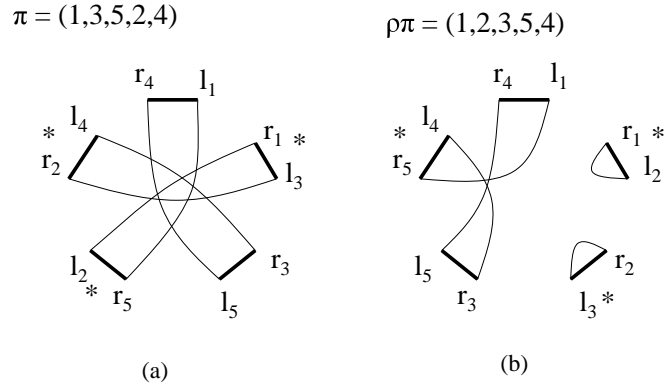


FIGURE 3. (a) Breakpoint graph for  $\pi = (1, 3, 5, 2, 4)$  (b) Breakpoint graph of  $\rho\pi$  after the transposition  $\rho$  has acted on  $\pi$  before positions 2,4, and 5 (which corresponds to the starred black edges)

**-Grey edges-**  $\forall i \in \{1, \dots, n\}$  connect  $r_i$  and  $l_{i+1}$  by a grey edge. (These edges represent the identity permutation).

**-Black edges-**  $\forall \pi_i, i \in \{1, \dots, n\}$  connect  $l_{\pi_i}$  and  $r_{\pi_{i-1}}$  by a black edge. (These edges represent the original permutation  $\pi$ ).

Note: By convention we put the black edges around the perimeter of a circle with the grey edges as chords through the middle of the circle.

Above we give an example of generating a breakpoint graph for the permutation  $\pi = (1, 3, 5, 2, 4)$  (see Figure 2). By the way we constructed  $G(\pi)$ , every vertex has degree 2 which means  $G(\pi)$  can be decomposed into cycles. We define  $c(\pi)$  to be the number of cycles in  $G(\pi)$ . For  $\pi$  in Figure 2, we have  $c(\pi) = 1$  since we have just one cycle of length 5 where length refers to the number of black edges present. Transpositions act *only* on black edges when viewed as an action occurring on  $G(\pi)$ . Each transposition which acts on  $G(\pi)$  corresponds to a transposition on the permutation  $\pi$ . To follow the action of a transposition on  $G(\pi)$ , we cut three black edges, perform the transposition around the circle, then reconnect the black edges appropriately (see Figure 3). Note that applying a transposition to a permutation can affect  $c(\pi)$ . This is actually our goal since if we notice that the identity's breakpoint

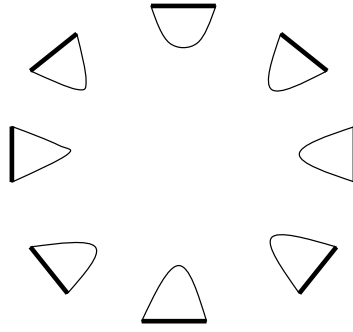


FIGURE 4.  $G(id)$  - The breakpoint graph for the identity permutation (on 8 elements).

graph,  $G(id)$  (see Figure 4) has  $n$  1-cycles, i.e.  $c(id) = n$ , our goal when sorting a permutation is to transform  $G(\pi)$  into a graph with all 1-cycles. Figure 3(a) shows  $c(\pi)=1$  and Figure 3(b) shows  $c(\rho\pi)=3$ .

*Transposition Distance.* It turns out instead of keeping track of the number of cycles in a breakpoint graph, we only need to keep track of the number of *odd* cycles. A cycle in  $G(\pi)$  is denoted as *odd* if it has an odd number of black edges. Define  $c_{odd}(\pi)$  to be the number of odd cycles in a breakpoint graph. We would like to increase the number of odd cycles with each transposition as fast as possible to get to  $n$  odd cycles (each with length 1), which would mean we would have the identity permutation. Since our goal is to increase the number of odd cycles as fast as possible, it is a natural question to ask what is the most number of odd cycles we can increase with a single transposition. The surprising result is that this is a fixed small number. Let  $\rho$  be a fixed transposition. We introduce the notation  $\Delta c(\pi, \rho)$  to indicate the increase in the number of cycles a transposition  $\rho$  induces. We define  $\Delta c(\pi, \rho)$  as

$$(1) \quad \Delta c(\pi, \rho) = c(\rho \cdot \pi) - c(\pi).$$

We use  $\Delta c(\pi, \rho)$  to calculate the number of new cycles created by the transposition  $\rho$ . The following lemma is due to Bafna and Pevzner [BP98].

LEMMA 1. For all permutations  $\pi$  and transpositions  $\rho$ , we have  $\Delta c(\pi, \rho) \in \{-2, 0, 2\}$  and  $\Delta c_{odd}(\pi, \rho) \in \{-2, 0, 2\}$ .  $\square$

Because of Lemma 1 is now of interest to us to examine what kinds of transpositions result in increasing the number of odd cycles by either -2, 0, 2. This leads us to the following definition:

DEFINITION 3. Given a transposition  $\rho$ , define a *k-transposition* (or a *k-move*) as a transposition on  $\pi$  such that  $\Delta c_{odd}(\pi, \rho) = k$ .

Intuitively, a *k-transposition* is a transposition which increases the number of odd cycles in the resulting permutation by *k*. Lemma 1 tells us that we only have -2, 0, and 2-transpositions. For example we see that in Figure 3 the transposition is a 2-transposition since the number of odd cycles is increased from 1 to 3. We now know that the most odd cycles any single transposition can add is two. If we could find a sequence of transpositions that added two odd cycles with each transposition, this would be optimal. However, this is not always possible. Hartman and Shamir [HS03] proved that there always exists a sequence of transpositions which has at least 2 out of every 3 transpositions being 2-transpositions. This gives us a  $\frac{3}{2}=1.5$  approximation, and this fact is the basis of their paper. We now examine how and why this is true.

Let  $n(\pi)$  be the number of black edges in  $G(\pi)$  and  $d(\pi)$  the minimal transposition distance. When  $\pi$  is completely sorted to the identity permutation, there will be  $n(\pi)$  odd cycles (all of length 1). We now give a lower bound for the minimal transposition distance.

LEMMA 2. (Bafna and Pevzner [BP98]) For all permutations  $\pi$ ,  $d(\pi) \geq \frac{n(\pi) - c_{odd}(\pi)}{2}$ .

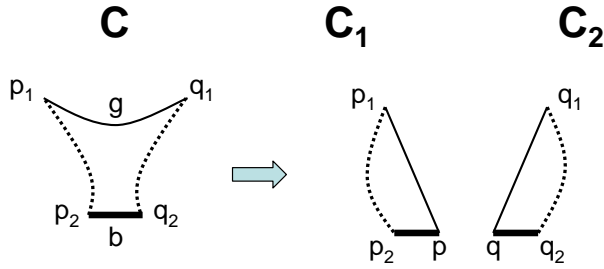


FIGURE 5. How to split a cycle of length  $k$  into a 3-cycle and a  $(k - 2)$ -cycle. A dashed line indicates a path and  $g$  represents a grey edge while  $b$  represents a black edge.

PROOF. (No proof was given in [BP98]) Create a set  $S$ , such that each element  $s$  in  $S$  is the number of transpositions in some sequence of transpositions which sorts the permutation  $\pi$ . If we consider the trivial sequence of transpositions that sorts  $\pi$  one element at a time, it is clear that there always exists a sequence of transpositions which sorts  $\pi$ . This implies that  $S$  is nonempty. As any nonempty set of natural numbers will have a smallest element, we know there always exists an optimal sequence of transpositions  $t_1, t_2, \dots, t_{d(\pi)}$  which transforms  $G(\pi)$  into  $G(id)$  in the least number of steps  $d(\pi)$ . Starting with  $c_{odd}(\pi)$ , by Lemma 1 the most odd cycles we can add with each transposition is two. Therefore optimally we would have a sequence of transpositions yielding 2 new odd cycles each giving  $c_{odd}(\pi) + (2 + 2 + 2 + \dots + 2) = n(\pi)$ . Since every transposition  $t_i$  may not always yield 2 new odd cycles, we have  $c_{odd}(\pi) + 2d(\pi) \geq n(\pi)$ .  $\square$

*Simple Permutations.* The length of a cycle in  $G(\pi)$  is the number of black edges present in the cycle and a cycle of length  $k$  is called a  $k$ -cycle. If  $k \leq 3$ , then the cycle is *short*; otherwise the cycle is long. A breakpoint graph is *simple* if it only contains short cycles. The algorithm for the 1.5-approximation is contingent upon the breakpoint graph being simple. It turns out that transforming a breakpoint graph into a simple breakpoint graph by inserting new elements and edges maintains the lower

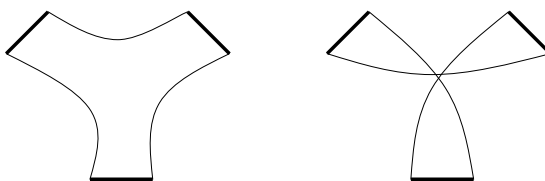


FIGURE 6. The only two possible configurations of 3-cycles. The first one is oriented, while the second is unoriented

bound of the transposition distance [HP99; LX01]. This simplifies matters since now all we have to do is convert 1,2, and 3-cycles into all 1-cycles, which is the identity permutation. We say that a transformation is *safe* if it maintains the same lower bound of Lemma 2, that is if  $n(\hat{\pi}) - c_{odd}(\hat{\pi}) = n(\pi) - c_{odd}(\pi)$ . If the transformation from  $\pi$  to  $\hat{\pi}$  is safe, then we say that  $\pi$  is *equivalent* to  $\hat{\pi}$ . We formulate the previous statements as a lemma:

LEMMA 3. (Lin and Xue [LX01]) Every permutation  $\pi$  can be transformed into a simple permutation  $\hat{\pi}$  by inserting new edges without changing the lower bound of  $d(\pi)$ .

PROOF. Let  $\pi$  be a permutation that contains a long cycle  $C$  of length  $k$ . Let  $b_1$  be a black edge in  $C$ . Denote by  $b_2$  and  $b_3$  the black edges that are connected to  $b_1$  via grey edges. Let  $g$  be the grey edge that is connected to  $b_2$  but not to  $b_1$ . Then we split  $C$  into a 3-cycle and a  $(k - 2)$ -cycle in  $\hat{\pi}$  using the method shown in Figure 5. In Figure 5, either  $C_1$  will be the 3-cycle and  $C_2$  will be the  $(k - 2)$ -cycle or vice versa. Clearly,  $n(\hat{\pi}) = n(\pi) + 1$ , and  $c_{odd}(\hat{\pi}) = c_{odd}(\pi) + 1$ , so the split is safe. This process can be repeated until a simple permutation is eventually obtained. Refer to Lin and Xue [LX01] for a more detailed explanation of the splitting process.  $\square$

Lemma 3 allows us the ability to convert  $\pi$  to a simple permutation  $\hat{\pi}$ , sort  $\hat{\pi}$ , then mimick the sorting of  $\hat{\pi}$  on  $\pi$ . In order to turn these 2-cycles and 3-cycles in  $G(\pi)$  into 1-cycles, we need the cycles to be configured in a certain way.

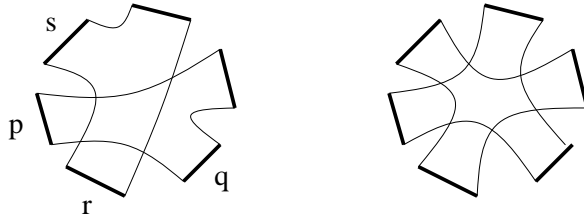


FIGURE 7. (a) Two intersecting three cycles. Here we see that the pair of black edges  $(p, q)$  intersects with the pair of black edges  $(r, s)$ , so by definition the cycles intersect (b) Two interleaving 3-cycles.

DEFINITION 4. A cycle is *oriented* if there is a transposition which cuts 3 of its black edges and yields a 2-transposition (see Def. 3), otherwise a cycle is called *unoriented*, that is it does not yield a 2-transposition.

For an oriented cycle we can find a transposition which acts on its edges and we can get 2 additional odd cycles. It turns out that there is only two possible configurations of 3-cycles which are shown in Figure 6. One of these configurations is oriented and the other is unoriented. The 1.5-approximation algorithm *Sort* attacks the sorting problem on a simple permutation by attacking the breakpoint graph depending on different cases of 2-cycles or 3-cycles which are oriented or unoriented, systematically so called *shattering* 3-cycles or 2-cycles into 1-cycles. A concept at the heart of the 1.5-approximation that helps us shatter these 2-cycles and 3-cycles is a specific sequence of transpositions. A  $(0, 2, 2)$  – *sequence* is a sequence of transpositions where the first is a 0-transposition (no new cycles added), then two 2-transpositions. Note that this adds 4 new cycles out of the most possible 6 new cycles in only three transpositions (see Lemma 1). This gives us our  $\frac{6}{4}=1.5$  approximation. We need one more concept before we can present the algorithm.

Two pair of black edges in a cycle are said to *intersect* if they alternate around the circumference of the circle the breakpoint graph is drawn around. We say that two cycles are *intersecting* if there are 2 pairs of black edges (one pair from each cycle) which intersect (see Figure 7(a)). We say that two 3-cycles are *interleaving* if

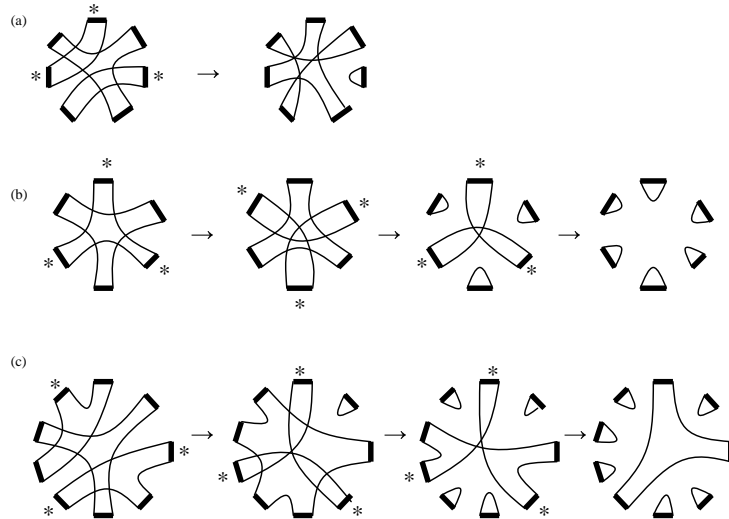


FIGURE 8. (a) A 2-transposition on a permutation that contains a 2-cycle (b) A  $(0,2,2)$ -sequence of transpositions on two interleaving 3-cycles. (c) A  $(0,2,2)$ -sequence of transpositions on three unoriented 3-cycles.

every pair of black edges from one of the 3-cycles intersects with some pair of black edges from the other 3-cycle (see Figure 7(b)).

We will now give a list of lemmas from various sources which prove that every step in the 1.5-approximation algorithm is able to be done. For proofs see the various sources. We give visual examples of the lemmas in Figure 8.

LEMMA 4. (Christie [Chr99]) If  $\pi$  contains a 2-cycle, then there exists a 2-transposition on  $\pi$ .  $\square$

LEMMA 5. (Hartman and Shamir [HS03]) Let  $\pi$  be a permutation that contains two interleaving unoriented 3-cycles. Then there exists a  $(0,2,2)$ -sequence of transpositions on  $\pi$ .  $\square$

LEMMA 6. (Hartman and Shamir [HS03]) Let  $\pi$  be a permutation that contains three unoriented 3-cycles  $C, D$ , and  $E$ , such that  $E$  is shattered by  $C$  and  $D$ . Then, there exists a  $(0,2,2)$ -sequence of transpositions on  $\pi$ .  $\square$

The Lemma 7 needs a bit of set up.

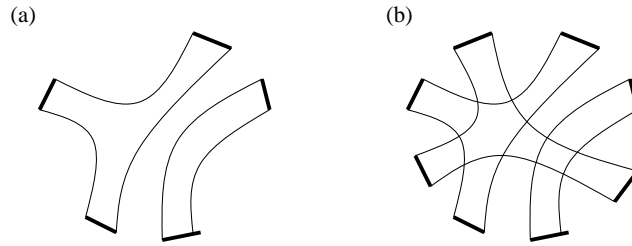


FIGURE 9. (a) A configuration, which is impossible as a breakpoint graph, but is a subgraph of (b). (b) A breakpoint graph which includes (a) as a subgraph. Note that there are no open gates in (b).

DEFINITION 5. A *configuration* of cycles is a subgraph of the breakpoint graph induced by one or more cycles.

For example Figure 6 shows the 2 possible configurations of 3-cycles. We say that two black edges in a cycle are *adjacent* if they are connected by a grey edge. It turns out that some of the unoriented configurations we have shown (see Figure 6) are possible as subgraphs embedded in a larger breakpoint graph, but are not possible as a breakpoint graph by itself. A pair of adjacent black edges which does not intersect with another cycle is called an *open gate*. Note that the component in Figure 9(a) has an open gate. A subgraph with no open gates is called *full* (see Figure 9(b)). Any graph with an open gate is not possible to be the breakpoint graph of a permutation. This subgraph could be embedded in a larger graph with no open gates which can be a breakpoint graph. These ideas are illustrated in Figure 9 and are condensed into our next lemma due to Bafna and Pevzner.

LEMMA 7. (Bafna and Pevzner [BP98]) A breakpoint graph cannot have an open gate.  $\square$

Figure 10 gives Hartman and Shamir's [HS03] 1.5-approximation algorithm to sort a permutation  $\pi$ . There are a few things to notice about their algorithm. In the first step, we transform our permutation  $\pi$  into a simple permutation, meaning

all cycles are short (of length 3 or less). We should note that no transposition we employ creates a long cycle, which would mean our permutation is no longer simple. Also, after we take care of all of the 2-cycles, no later transposition creates another 2-cycle, so we never have to revert back to previous steps. We now prove that this sorting algorithm is actually a 1.5-approximation.

**THEOREM 2.** (*Hartman and Shamir [HS03]*) *Algorithm Sort is a 1.5-approximation algorithm for sorting circular permutations.*

**PROOF.** Let  $alg(\hat{\pi})$  be the number of transpositions used in the algorithm *Sort* to sort  $\hat{\pi}$  into the identity permutation. We have the following string of inequalities,

$$alg(\hat{\pi}) \leq 1.5 \left( \frac{n(\hat{\pi}) - c_{odd}(\hat{\pi})}{2} \right) = 1.5 \left( \frac{n(\pi) - c_{odd}(\pi)}{2} \right) \leq 1.5 \cdot d(\pi)$$

Since the sorting algorithm *Sort* guarantees an increase of 4 odd cycles out of a maximally possible 6, the first inequality holds. The middle equality holds as  $\pi$  and  $\hat{\pi}$  are equivalent (see Lemma 3), that is  $n(\hat{\pi}) - c_{odd}(\hat{\pi}) = n(\pi) - c_{odd}(\pi)$ . The last inequality holds trivially by Lemma 2.  $\square$

It should be noted that the algorithm *Sort* can also sort linear permutations via Theorem 1. Next we will improve upon this 1.5-approximation with a 1.375-approximation by Elias and Hartman which uses a lot of the same lemmas, but improves the algorithm.

## 2.2. 1.375-APPROXIMATION

Elias and Hartman's 1.375-approximation for sorting by transpositions takes many of the ideas in Hartman and Shamir's 1.5-approximation paper and improves on some of them. Refer to Section 1 for basic definitions, terminology, lemmas, and theorems. A basic explanation of their idea is that they give an algorithm such that 8 out of 11 transpositions are 2-transpositions (they yield 2 new odd cycles each), giving us

### Algorithm Sort ( $\pi$ )

1. Transform permutation  $\pi$  into an equivalent simple permutation  $\Pi$  (Lemma 3).
2. While  $G(\Pi)$  contains a 2-cycle, apply a 2-transposition (Lemma 4).
3. While  $G(\Pi)$  contains a 3-cycle, do:
  - Pick an adjacent pair of black edges  $c$  from cycle  $C$ . If  $C$  is oriented - apply a 2-transposition.
  - Otherwise, pick an adjacent pair  $d$  (from cycle  $D$ ) that intersects with  $c$  (guaranteed by Lemma 7). If  $C$  and  $D$  are interleaving - apply a (0,2,2)-sequence (Lemma 5).
  - Otherwise, there is an adjacent pair  $c'$  in  $C$  that does not intersect with  $D$ . Pick an adjacent pair  $e$  (from cycle  $E$ ) that intersects with  $c'$  (guaranteed by Lemma 7). Cycle  $C$  is shattered by cycles  $D$  and  $E$ , thus, it is possible to apply a (0,2,2)-sequence (Lemma 6).
4. Mimic the sorting of  $\pi$  using the sorting of  $\Pi$ .

FIGURE 10. A 1.5-approximation algorithm for sorting by transpositions due to Hartman and Shamir [HS03].

an  $\frac{11}{8}=1.375$ -approximation. There is a little new terminology needed before we can present the algorithm.

Recall a *configuration* of cycles is a subgraph of the breakpoint graph induced by one or more cycles. We say that a configuration is *connected* if for any two cycles  $c_1$  and  $c_k$  there is a sequence of cycles  $c_2, \dots, c_k$  such that  $c_i$  intersects with  $c_{i+1}$  for  $i \in \{1, k-1\}$ . A good example of this is in Figure 9. Note that Figure 9(a) is not a connected configuration and that Figure 9(b) is a connected configuration.

**DEFINITION 6.** A *component* is a configuration within a breakpoint graph which is maximal in the sense that no cycle in the configuration intersects with any cycle outside the configuration.

Components play an important part in solving sorting problems with other operations, such as reversals. With this new terminology we see that Lemma 7 actually implies that all breakpoint graphs are components. For example in Figure 7(a) we see that by Lemma 7, there must be another cycle since we see an open gate. This implies that the configuration shown is not maximal. We reveal cycle after cycle until

there are no more open gates. We then have the maximally connected breakpoint graph, i.e. the breakpoint graph is a component.

In the Hartman and Shamir 1.5-approximation paper, they extensively used the  $(0,2,2)$ -sequence of transpositions. This sequence is a series of three transpositions, where the first is a 0-move, the second and third are 2-moves. Since there were only three transpositions, the terminology just given was sufficient. However, for the Elias and Hartman 1.375-approximation paper, we will be dealing with much longer sequences of transpositions. Therefore our terminology must adapt. Define an  $(x, y)$ -sequence of transpositions as a sequence of  $x$  transpositions, such that at least  $y$  of them are 2-moves. Therefore under the old terminology our  $(0,2,2)$ -sequence now becomes a  $(3,2)$ -sequence. We say that a configuration *has* an  $(x, y)$ -sequence if it is possible to apply such a sequence on the configuration.

Combining this new terminology and various previous lemmas, we state this useful corollary.

**COROLLARY 1.** *For every permutation that has an oriented cycle and contains at least three 3-cycles, there exists a  $(4,3)$ -sequence.  $\square$*

This corollary was the basis for all of the 1.5-approximation papers. Define a  $k$ -permutation to be a permutation which only has cycles of length  $k$ . We know by Lemma 4 that there is a 2-transposition for any 2-cycle and trivially 1-cycle are done. Since 1-cycles and 2-cycles can be handled optimally, to get a 1.375-approximation we need to show that for all 3-permutations there always exists an  $(x, y)$ -sequence such that  $x \leq 11$  and  $\frac{x}{y} \leq \frac{11}{8}$ . This means that for every configuration containing at least 8 cycles, there is a sequence of transpositions such that 8 of the 11 are 2-transpositions, giving a 1.375-approximation. We call such a sequence of transposition an  $\frac{11}{8}$ -sequence.

Algorithm 1.375-Sort ( $\pi$ )

1. Transform permutation  $\pi$  into an equivalent simple permutation  $\Pi$  (Lemma 3).
2. If there is a (2,2)-sequence, apply it.
3. While  $G(\Pi)$  contains a 2-cycle, apply a 2-move (Lemma 4).
4. While  $G(\Pi)$  contains at least 8 cycles of length 3, apply an  $\frac{11}{8}$ -sequence (Corollary 2).
5. While  $G(\Pi)$  contains a 3-cycle, apply a (3,2)-sequence (Corollary 1).
6. Mimic the sorting of  $\pi$  using the sorting of  $\Pi$ .

FIGURE 11. Elias and Hartman's algorithm 1.375 – *Sort*.

One pleasant fact is that by Corollary 1, if a configuration has an oriented cycle, then there is a  $\frac{11}{8}$ -sequence. This is because Corollary 1 implies there is a (4,3)-sequence and a (4,3)-sequence satisfies the criteria for an  $\frac{11}{8}$ -sequence. This greatly reduces our analysis because Corollary 1 enables us to only consider unoriented configurations (meaning *all* cycles in the configuration are unoriented).

Elias and Hartman use extensive case analysis in order to prove the above statements. Case analysis proofs are becoming more frequent. The most well known of which is probably the proof of the Four Colors Theorem. Elias and Hartman's case analysis uses a computer to verify some 80,000 possible cases of unoriented configurations and also comes with a verification program (see [http://www.sbc.su.se/~isaac/SBT1375\\_proof/](http://www.sbc.su.se/~isaac/SBT1375_proof/)).

Their result which is the backbone of the 1.375-approximation algorithm is the following corollary:

COROLLARY 2. *Every 3-permutation with at least 8 cycles has an  $\frac{11}{8}$ -sequence.*  $\square$

Figure 11 shows Elias and Hartman's approximation algorithm 1.375 – *Sort*. After we have a simple permutation (Step 1), there are only 1,2, and 3-cycles. Whenever there is a 2-cycle, we have a 2-transposition (Step 3). Elias and Hartman's case analysis has shown that after all of the 2-cycles are gone and we only have a 3-cycles

left (making it a 3-permutation), we can always find an  $\frac{11}{8}$ -sequence, giving us our  $\frac{11}{8}=1.375$ -approximation (Step 4).

### 2.3. DIFFICULTY COMPUTING EXACT TRANSPOSITION DISTANCE

For some evolutionary processes, we have results for the exact evolutionary distances. These processes include reversals (on signed permutations), insertions/deletions, and others. However, every attempt to solve the transposition distance has resulted in only an approximation. It is worth a look to examine what the difficulty is when trying to compute transposition distance.

The key is found at the very first step in both algorithms given earlier. For other evolutionary processes, we are able to handle the permutation as a whole, without breaking it into simpler pieces. Since a transposition deeply affects the structure of the permutation, every attempt so far consists of breaking the permutation down into simpler, more manageable segments. This is what is done in Lemma 3 (which is the basis for Step 1 in both approximation algorithms). However, as soon as we transform a breakpoint graph into a simple breakpoint graph, the transposition distances are affected. It turns out to not matter for an approximation because for an approximation the lower bounds for the transposition distance are preserved. Any two breakpoint graphs which maintain the same lower bound are *equivalent* which is the only restriction for the previously given approximation algorithms. However, as soon as we transform a permutation into a simple permutation, the exact transposition distance can be different for the original permutation versus the transposition distance for derived simple permutation.

It is not currently known if sorting by transpositions is NP-hard. If we are to solve the transposition distance problem, permutations will need to be considered as a whole unit, rather than simplifying the permutation, which immediately creates an approximation to the original transposition distance.

## CHAPTER 3

### SORTING BY REVERSALS

#### 3.1. REVERSALS

A *reversal* is an evolutionary process in which a segment of genomic material is cut out, reversed, and then inserted back in the same place, except backwards (see Figure 12(a)). A more rigorous definition of reversals will be given later. Of all the evolutionary processes, reversals (or inversions) may be the most frequently occurring. Some genomes, such as many plant mitochondrial DNA, are thought to evolve only through reversals [PH88].

In this chapter  $d(\pi)$  will refer to reversal distance and all permutations  $\pi$  are circular. Reversal distance is the minimal number of reversals it takes to transform a given permutation  $\pi$  into the identity permutation,  $id$ . A trivial lower bound on the reversal distance can be computed by observing that each reversal can remove at most two *breakpoints* (see Definition 1). This fact is obvious and yields the lower bound on  $d(\pi)$  given by

$$d(\pi) \geq \left\lceil \frac{b(\pi)}{2} \right\rceil$$

where  $b(\pi)$  is the number of breakpoints in  $\pi$  [Har04]. The first attempts to solve the reversals problem yielded a series of better and better approximation algorithms. The first of these approximation algorithms was a greedy algorithm given by Kececioglu and Sankoff [KS95] in 1995. This approximation was subsequently improved by Bafna and Pevzner [BP96] to a 1.75-approximation in 1996. The approximation ratio was

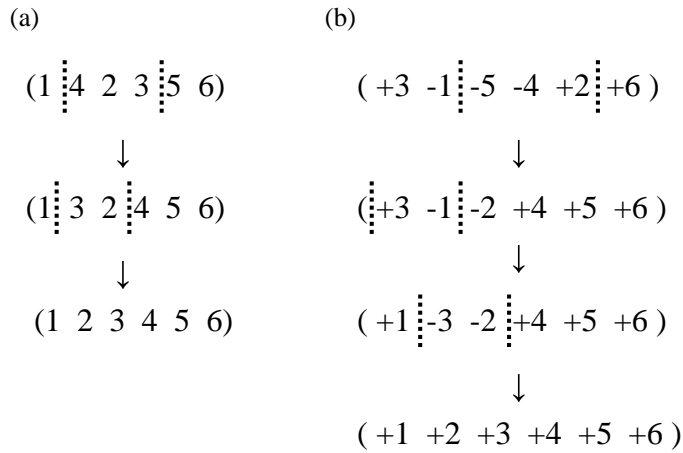


FIGURE 12. Examples of sorting by reversals. (a)- unsigned permutation (b)-signed permutation

improved again in 1998 by Christie [Chr98] to 1.5 and then to our current best estimate, a 1.375-approximation, given by Berman et al. [BHK02] in 2002. A big breakthrough came in 1997 when Caprara [Cap97] proved that sorting an unsigned permutation is NP-hard. In 1999, Berman and Karpinski [BK99] prove not only that sorting an unsigned permutation is NP-hard, but that even any sorting algorithm with an approximation ratio less than 1.008 will be NP-hard. The good news is that the more realistic situation occurring in nature is that most genomic sequences have position as well as orientation. That is, our permutations we work with will be *signed*, meaning that each element in the permutation will have in addition either a + or a -. When a reversal is applied to a permutation, the elements in the affected segment are reversed, and in addition whatever sign that is attached to each element is changed (see Figure 12(b)).

All of the approximation algorithms mentioned above are for *unsigned* permutations. Also it is NP-hard to sort unsigned permutations, not signed permutations. A huge breakthrough for sorting signed permutations came in 1995 when Hannenhalli and Pevzner [HP99] gave a polynomial-time algorithm which solved the sorting by reversals problem for signed permutations. Recall that  $c(\pi)$  is the number of cycles

in the *breakpoint graph* (see Definition 2) and that  $\Delta c(\pi, \rho) = c(\rho \cdot \pi) - c(\pi)$ . Hannenhalli and Pevzner found a result similar to that of Lemma 1 except for reversals:

LEMMA 8. [KS94] For all permutations  $\pi$  and reversals  $\rho$ , we have  $\Delta c(\pi, \rho) \in \{-1, 0, 1\}$ .  $\square$

This implies that each reversal modifies the number of cycles by at most one. Since the identity permutation is the only one with  $n$  cycles (all 1-cycles), we have that

$$(2) \quad d(\pi) \geq b(\pi) - c(\pi)$$

However, the question is how much greater. Hannenhalli and Pevzner amazingly gave a closed form for the reversal distance for a signed permutation

$$d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi).$$

Their result is of great historical importance, but has been substantially simplified, as we will show in Section 3.3.

### 3.2. CLOSED FORM FOR REVERSAL DISTANCE OF SIGNED PERMUTATIONS

In 1999, Hannenhalli and Pevzner proved that sorting a signed permutation by reversals is possible in polynomial time. In addition they found a closed form for the reversal distance for a signed permutation. For reference the equality is

$$(3) \quad d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi)$$

where  $h(\pi)$  is the number of *hurdles* in  $\pi$  and  $f(\pi)$  is 1 if  $\pi$  is a particular type of permutation known as a *fortress*, otherwise  $f(\pi)$  is 0. For this section we only

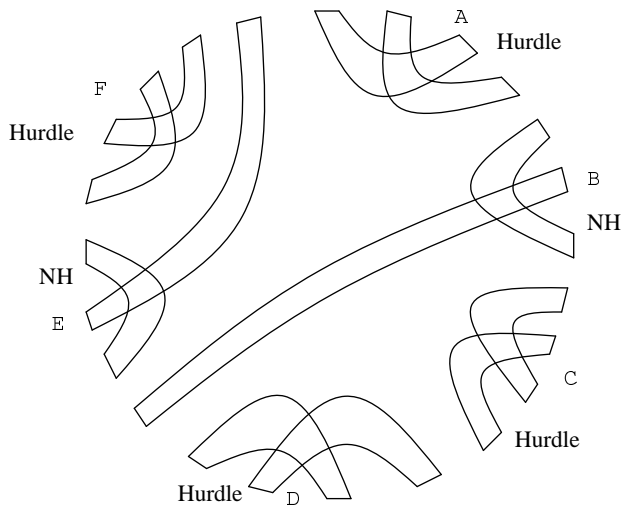


FIGURE 13. Hurdles and Non-hurdles

give a very brief explanation of the reversal distance for signed permutation found by Hannenhalli and Pevzner [HP99]. We now begin to give a basic break down of the equality.

We use the same definition for a breakpoint graph  $G(\pi)$  as in Section 1 to draw the breakpoint graph for a permutation  $\pi$ . A cycle in  $G(\pi)$  is *good* if it has two grey edges (see Definition 2) that cross in the breakpoint graph. Otherwise the cycle is *bad*.

DEFINITION 7. A component is *good* if it contains at least one good cycle. Otherwise it is a *bad component*.

If we have a good component, it is known that there is a reversal to increase the number of odd cycles by 1, which is optimal. This concept of a good component is similar to that of an oriented cycle (see Definition 4) for transpositions. Since we can handle good components optimally, we only need to consider bad components. Component A *separates* components B and C if any edge drawn from B to C would have to intersect with some edge from component A. Notice that for example in Figure 13, component E separates A and F.

Bad components are separated into *hurdles* and *non – hurdles*:

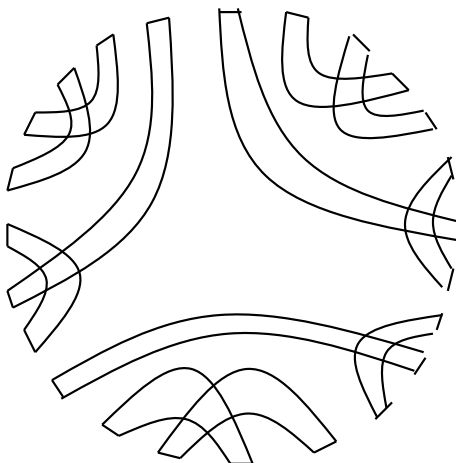


FIGURE 14. A fortress.

DEFINITION 8. A *hurdle* is a bad component that does not separate any pair of bad components. Likewise, a *non-hurdle* is a bad component that separates two bad components.

An example of a hurdle is component A in Figure 13. Notice that component A does not separate any two of the other bad components.

It turns out that in the optimal sequence of reversals which sorts a permutation  $\pi$ , we need exactly one extra reversal for each hurdle to sort  $\pi$ . This means that (2) is sharp if there are no hurdles. In addition, this explains the need for  $h(\pi)$  extra reversals to compute  $d(\pi)$  in (3).

The only term left undefined in (3) is *fortress* which needs some set up to define. Under the classification of all hurdles, there are two types. One type of hurdle is a *simple hurdle*, the other is a *super hurdle*. We say that component X *protects* nonhurdle Y if the removal of X would cause Y to become a hurdle. Knowing this, we say that

DEFINITION 9. A *super hurdle* is a hurdle which protects a non-hurdle. Likewise, a *simple hurdle* is a hurdle which does not protect a non-hurdle.

For example, in Figure 13 component F protects E from becoming a hurdle; that is, if we removed component F, then E would become a hurdle. Therefore, we say that component F is a super hurdle. We are now ready to define a fortress:

DEFINITION 10. A *fortress* is a permutation where there are an odd number of hurdles and all of them are super hurdles.

Fortresses require a single extra reversal to be sorted. Therefore, if  $\pi$  is not a fortress,  $f(\pi) = 0$  and there is no need for an extra reversal. If  $\pi$  is a fortress, then there is one extra reversal needed, thus  $f(\pi) = 1$  and (3) stands. Figure 14 is the smallest fortress possible and it contains 3 superhurdles. Even though we now have a closed form for the reversal distance, clearly the terms hurdle and fortress are quite cumbersome. In the following section, we examine a paper in which the authors have again computed the reversal distance, but without using the terms hurdle and fortress.

### 3.3. REVERSAL DISTANCE WITHOUT HURDLES AND FORTRESSES

In this section, our purpose will be to give an overview of a paper with the same name by Bergeron et al. [BMS04] which simplifies the results of Hannenhalli and Pevzner. It should be noted that, for this section, we are not implying that we can only handle permutations without hurdles or fortresses. Instead, "without hurdles and fortresses" means we can compute the reversal distance on any signed permutation without using the notion of a hurdle or fortress.

The main idea of the Bergeron et al. paper is that they prove equality (3) found by Hannenhalli and Pevzner, except they have found a way around the somewhat cumbersome terms *hurdles* and *fortresses*. They use simple known results on trees, yet still arrive at the same conclusions. We begin with some needed definitions:

$$P_1 = (0 \ 5 \ 6 \ -3 \ -4 \ 2 \ -1 \ 7)$$

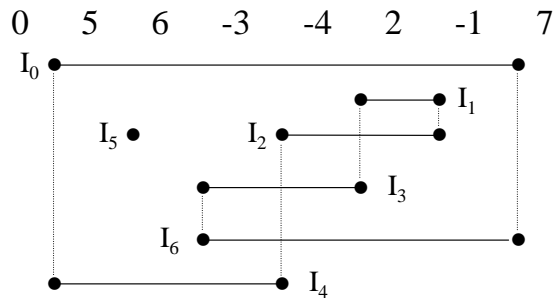


FIGURE 15. Elementary intervals for  $P_1$ . Elementary intervals are shown with solid horizontal lines. The cycles the intervals generate are completed with vertical dotted lines.

A *point*  $p \cdot q$  is defined as a pair of consecutive elements in a permutation. For example, let  $P_1 = (0 \ 5 \ 6 \ -3 \ -4 \ 2 \ -1 \ 7)$  be a circular permutation. We have 8 points on this permutation:  $0 \cdot 5$ ,  $5 \cdot 6$ ,  $6 \cdot -3$ , etc.

We say that a point is an *adjacency* if it is of the form  $i \cdot (i + 1)$  or  $-(i + 1) \cdot -i$ , otherwise it is a *breakpoint*. Our goal in sorting by reversals is to get rid of all breakpoints and create nothing but adjacencies, as the identity permutation  $id$  has no breakpoints.  $P_1$  has one adjacency,  $5 \cdot 6$ , and the rest are breakpoints.

It will be useful to define an interval within the permutation by its endpoints. An interval can be empty if its two endpoints are equal. A non-empty interval can be defined by its first and last elements, known as its *bounding elements*. The notation for a non-empty interval is  $(i..j)$  where  $i, j$  are the bounding elements.

A *reversal* of an interval on a signed permutation reverses the interval while also changing the sign of every element of the interval. For example a point  $p \cdot q$  in an interval which is reversed changes to  $-q \cdot -p$ .

We now define a particular type of interval which is fundamental to sorting by reversals.

$$P_2 = (0 \ -5 \ -3 \ -4 \ -2 \ -1 \ 6 \ 7 \ -12 \ 11 \ -10 \ 9 \ -8 \ 13)$$

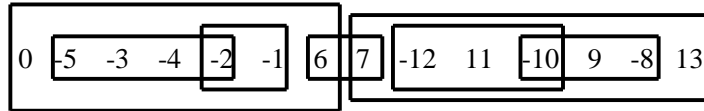


FIGURE 16. Components in the permutation  $P_2$ .

DEFINITION 11. For each pair of unsigned elements  $(k, k + 1)$ ,  $0 \leq k < n$ , define the *elementary interval*  $I_k$  associated to the pair to be the interval whose endpoints are:

- 1) The right point of  $k$ , if  $k$  is positive, otherwise its left point.
- 2) The left point of  $k + 1$ , if  $k + 1$  is positive, otherwise its right point.

Elements  $k$  and  $k + 1$  are called the *extremities* of the elementary interval.

Note that empty elementary intervals correspond to the adjacencies in the permutation. Figure 15 shows the elementary intervals in  $P_1$ . An *oriented* elementary interval has extremities of opposite sign, else the interval is *unoriented*. Notice that reversing an oriented elementary interval  $I_k$  creates an adjacency, either  $k \cdot k + 1$  or  $-(k + 1) \cdot -k$ . Also if we notice that by Definition 11, two elementary intervals will meet at each breakpoint, then a natural construction of cycles will occur. A *cycle* is a sequence of points such that two successive points are the endpoints of an elementary interval. Figure 15 shows  $P_1$  has two cycles. One cycle is empty, and the other contains 6 breakpoints. A second example for the reader to follow can be found in [BMS04].

Our goal is to increase the number of cycles with each reversal, since adjacencies correspond to empty cycles and *id* has  $n$  cycles. Recall Lemma 8 which says that a reversal increases the number of cycles by at most one. This construction makes it clear why Lemma 8 is true. For all points of the permutation, except the endpoints of the reversal, the elementary intervals which meet at those points will meet at the

same points after the reversal by Definition 11. This means that either 2 cycles will be merged (a (-1)-reversal), 2 cycles will be split (a 1-reversal), or the number of cycles will remain unchanged (a 0-reversal).

The elementary intervals and cycles previously defined are contained in objects known as *components*. Components are used to define a particular type of tree which allows us to sort permutation by reversals without the notion of a hurdle or fortress.

DEFINITION 12. Let  $P$  be a signed permutation on  $\{0, 1, \dots, n\}$ . A *component* of  $P$  is an interval from  $i$  to  $(i + j)$  or from  $-(i + j)$  to  $-i$  for some  $j > 0$ , whose set of unsigned elements are  $\{i, \dots, i + j\}$ , and that is not the union of two such intervals.

A component can be thought of as a subpermutation since it is a permutation embedded within another permutation. We give a visual interpretation of components within a permutation in Figure 16. Let

$$P_2 = (0 \ -5 \ -3 \ -4 \ -2 \ -1 \ 6 \ 7 \ -12 \ 11 \ -10 \ 9 \ -8 \ 13)$$

$P_2$  has 7 components, represented in Figure 16 by blocks around the elements each component contains. Notice that some components overlap, some are disjoint, and others are nested within another component. It turns out there is a fixed set of rules governing how components are related to one another.

PROPOSITION 1. Two components of a permutation are either disjoint, nested with different endpoints, or overlapping on a single element.  $\square$

Notice that we have all three scenarios in  $P_2$ . Two components which overlap on one element are said to be *linked*. Successive linked components form a *chain* and a chain that cannot be extended in either direction is *maximal*. It is possible for a single component to be maximal. It should be noted that if a component of a chain is nested within a larger component  $A$ , then all components in the chain are nested within  $A$ . Using this proposition, we can define a tree to be associated with every

$$P_2 = (0 \ -5 \ -3 \ -4 \ -2 \ -1 \ 6 \ 7 \ -12 \ 11 \ -10 \ 9 \ -8 \ 13)$$

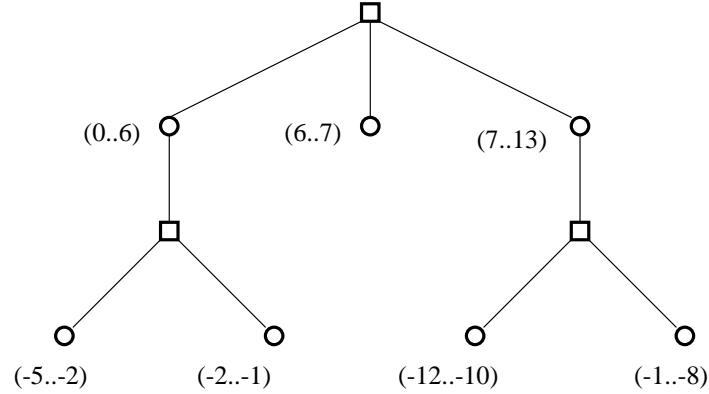


FIGURE 17. The tree  $T_{P_2}$  associated with permutation  $P_2$ .

permutation and using this associated tree, we can sort the permutation optimally using reversals. For now we define a forest, and then show through Proposition 2 that our forest is in fact a tree.

DEFINITION 13. Given a permutation  $P$  on  $\{0, 1, \dots, n\}$  and its components, define the forest  $T_P$  by the following construction:

- 1- Each component is represented by a round node.
- 2- Each maximal chain is represented by a square node whose (ordered) children are the round nodes that represent the components of this chain.
- 3- A square node is the child of the smallest component that contains this chain.

Figure 17 shows the forest  $T_{P_2}$  associated with permutation  $P_2$ .

PROPOSITION 2. [BMS04] If a permutation  $P$  begins with 0 and ends with  $n$ , then the resulting forest is a tree with a single square node as a root.

PROOF. (No proof was given in [BMS04]) If there is only one component in  $P$ , then we are done as the component is a single maximal chain. If there are two or more components, then one of the components must go from  $(0..i)$  for some  $i < n$ . and another component must go from  $(j..n)$  for some  $j < n$ . If  $j < i$ , we are done as

the two components are linked. Else,  $i < j$ . If  $j = i + 1$ , then  $(i..i+1)$  is a component and the three components form a maximally linked chain. If  $j > i + 1$ , then the set of elements between  $i$  and  $j$  are the elements  $\{i, i + 1, \dots, j\}$  only, which implies there is a sequence of linked components joining  $(0..i)$  and  $(j..n)$  and again, we have a maximally linked chain. All cases lead to a graph with a tree with a single square node as its root.  $\square$

DEFINITION 14. The *sign* of a point  $p \cdot q$  is positive if both  $p$  and  $q$  are positive, and negative if both are negative. A component is *unoriented* if it has one or more breakpoints and all of them have the same sign. Otherwise the component is oriented.

Similar to oriented components in the problem of sorting by transposition, we know that there is always a reversal to increase the number of cycles by the maximal number possible if we have an oriented component. This is because of the following proposition:

PROPOSITION 3. [BMS04] The endpoints of an elementary interval belong to the same component, thus all the points of a cycle belong to the same component.  $\square$

This proposition implies an oriented elementary interval will be contained within an oriented component, and as mentioned earlier reversing an oriented elementary interval creates an adjacency which is technically adding another cycle. Next we give a list of propositions from [BMS04] which tell what kind of reversals can be used to create oriented intervals in unoriented components. This will allow us to then reverse these oriented intervals to create additional cycles.

PROPOSITION 4. [BMS04] If a component  $C$  is unoriented, the reversal of an elementary interval whose endpoints belong to  $C$  orients  $C$ , and leaves the number of cycles of the permutation unchanged.  $\square$

This operation which orients a component, known as *cutting* a component, is actually not usually the optimal operation to use because there is a way to use a single reversal to orient more than one component at a time. We now show how to *merge* multiple components together and its effects on  $T_P$ .

PROPOSITION 5. If a reversal on a permutation  $P$  has its two endpoints in different components  $A$  and  $B$ , then only the components on the path from  $A$  and  $B$  in  $T_P$  are affected.

1- A component  $C$  is destroyed if and only if it contains either  $A$  or  $B$ , but not both.

2- If  $A$  or  $B$  is unoriented, any component  $C$  that contains both  $A$  and  $B$ , and that is on the path that joins  $A$  and  $B$ , will be oriented.

3- A new component  $C$  is created if and only if  $A$  and  $B$  are included in two components that are in the same chain. If either  $A$  or  $B$  is unoriented, then  $C$  will be oriented.  $\square$

By Proposition 5 we see that merging two unoriented components either destroys or orients all components on the path between the two components, without creating new unoriented components. Similar to the problem of sorting by transpositions, oriented components turn out to be easy to handle, and extra attention must be given to unoriented components. Define the *score* of a reversal as the number of oriented elementary intervals in the resulting permutation. The following theorem is due to Bergeron:

THEOREM 3. *The reversal of an oriented elementary interval of maximal score does not create new unoriented components.*  $\square$

Theorem 3 provides a very useful corollary which shows just how easy oriented components are to handle.

COROLLARY 3. *If a permutation  $P$  on  $n$  elements has no unoriented components and  $c$  cycles, then  $d(P) = n - c$ .*

PROOF. By equation (2) we have that  $d(P) \geq b(P) - c(P)$ . We know that an oriented reversal adds one new cycle. This implies that Theorem 3 guarantees there will always be enough oriented reversals to sort  $P$  since no new unoriented components will be created.  $\square$

We now have the tools to orient all unoriented components, therefore we turn our attention to choosing the optimal way to orient the unoriented components. To do this we use a *covering* on  $T_P$  to choose which reversals will orient all components optimally.

DEFINITION 15. A *cover*  $C$  of  $T_P$  is a collection of paths joining all the unoriented components of  $P$ , such that each terminal node of a path belongs to a unique path.

A path that contains two or more unoriented components is a *long path*, which corresponds to merging two components (see Proposition 5). A path containing a single component is a *short path*, which corresponds to cutting the component (see Proposition 4). The *cost* of a cover is defined as the sum of the costs of its paths, where:

- 1- The cost of a sort path is 1.
- 2- The cost of a long path is 2.

We define an *optimal cover* as a cover of minimal cost, with cost  $t$ . Each cover actually defines a set of reversals which orient all components in  $P$ . Trivially an optimal cover is the set of reversals which orients all the components in  $P$  optimally.

Using this terminology and known results on trees, we are now ready to give the main result for the reversal distance  $d(P)$  for a permutation on  $n$  elements without using the notion of hurdles or fortresses.

THEOREM 4. [BMS04] *If a permutation  $P$  on  $n$  elements has  $c$  cycles, and the associated tree  $T_P$  has minimal cost  $t$ , then*

$$d(P) = n - c + t.$$

PROOF. (Sketch from [BMS04]) We show  $d(P) \leq n - c + t$  and then  $d(P) \geq n - c + t$ . Let  $C$  be an optimal cover, then apply the  $m$  merges and  $q$  cuts induced by  $C$ . Note that since the cost of each merging is two and each cutting is one, we have  $t = 2m + q$ . After this merging and cutting, all components afterwards will be oriented (see Propositions 4 and 5) and the resulting permutation  $P'$  will now contain  $c - m$  cycles. However, since  $P'$  now has no unoriented components, Corollary 3 implies that  $d(P') = n - (c - m) = n - c + m$ . Since  $m + q$  reversals (merges + cuts) were applied to  $P$ , we have

$$d(P) \leq d(P') + (m + q) = n - c + 2m + q = n - c + t.$$

For the other direction assume any optimal sorting takes  $d(P)$  steps. By Lemma 8,  $d$  can be written as

$$(4) \quad d = s + m + q,$$

where  $s$  is the number of 1-reversals,  $m$  is the number of (-1)-reversals, and  $q$  is the number of 0-reversals. Since  $m$  reversals remove  $m$  cycles, and  $s$  reversals add  $s$  cycles, then  $c - m + s = n$ , which implies by (4) that  $d = n - c + m + (m + q)$  so

$$(5) \quad d = n - c + 2m + q.$$

Of the  $m$  reversals which merge components, let  $m_1$  be the number of long paths (which merge more than one component) and  $m_2$  be the number of short paths and let  $q'$  be the number of remaining unoriented components. We have that  $m_1 + m_2 \leq m$  and  $q' \leq q$ . Therefore since  $t \leq 2m_1 + m_2 + q'$ , we have

$$2m + q \leq 2m_1 + m_2 + q'$$

$$\Rightarrow n - c + 2m + q \leq n - c + 2m_1 + m_2 + q'$$

$$\Rightarrow d \geq n - c + 2m_1 + m_2 + q'$$

$$\Rightarrow d \geq n - c + t. \quad \square$$

We have computed the reversal distance for signed permutations without using the notion of a hurdle or fortress. In addition we have computed a close approximation for the transposition distance for unsigned permutations. In the next chapter we examine some of the remaining issues for sorting sequences of genomic material.

## CHAPTER 4

### REMAINING ISSUES IN SORTING GENOMIC MATERIAL

While much progress has been made in recent years in sorting genomic sequences using various naturally occurring operations, the majority of the work is left to be done. In this section we discuss some of the problems to be considered next.

A major issue is the fact that only individual operations have been investigated independently. For example, we have results on transposition distance and reversal distance, but in nature rarely does one of these operations account for all mutations in an organism's genetic material. We must learn how to find some way to account for multiple operations occurring simultaneously. There have been recent efforts to answer this question by S. Yancopoulos, O. Attie, R. Friedberg [YAF05] and others.

The next issue is that the mathematics used thus far weights the likelihood that an operation occurs between any given pair of elements equally. In nature certain segments of genomic material are more susceptible to mutation (or reordering) than others. Some sections of genomic material may code for an important protein and will not split there, and other segments may have duplications, which are known to be quite prone to mutation. Such spots where mutations are more likely to occur are known as *hot spots*. Places in which genes must be kept together to function properly are known as *operons*.

The third issue is the ultimate assumption that two organisms which have a small so called "evolutionary distance" implies that the organisms evolved from a close common ancestor. This is not a proven fact, only an assumption. Using rearrangement algorithms to create a phylogenetic tree does not prove the validity of the tree.

For example, if two automobiles have similar features, no one assumes one evolved into the other. Instead, it is in fact a greater implication for a common designer, being humans. The same holds true in nature. Two organisms, which are similar in genomic material, are a greater proponent for Intelligent Design. In addition, if the goal is to create a phylogenetic tree of life on earth, then the simplest and most basic life to first appear must be able to be created from non-life. With today's technology, abiogenesis is still impossible to be recreated in a lab. If the foundations of a phylogenetic history of earth do not stand up, neither will a phylogenetic tree tracing the recent evolutionary history of organisms on earth.

## BIBLIOGRAPHY

- [BHK02] P. Berman, S. Hannenhalli, and M. Karpinski. A 1.375-approximation algorithm for sorting by reversals. *Proc. of 10th European Symposium on Algorithms (ESA02)*, pages 200–210, 2002.
- [BK99] P. Berman and M. Karpinski. On some tighter inapproximability results. *Proceedings of the 26th ICALP*, pages 75–83, 1999.
- [BMS04] A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. *Combinatorial Pattern Matching: 15th Annual Symposium*, page 388, 2004.
- [BP96] V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Discrete Mathematics*, 25(2):272–289, 1996.
- [BP98] V. Bafna and P.A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 20(S2):224–240, May 1998.
- [Cap97] A. Caprara. Sorting by reversals is difficult. *Proceedings of the First International Conference on Computational Molecular Biology*, pages 75–83, 1997.
- [Chr98] D. A. Christie. A  $3/2$ -approximation algorithm for sorting by reversals. *Proc. ninth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 98)*, pages 244 – 252, 1998.
- [Chr99] D.A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1999.
- [EH05] Isaac Elias and Tzvika Hartman. A 1.375-approximation algorithm for sorting by transpositions. 2005.

- [Har04] Tzvika Hartman. *Combinatorial Algorithms for Genome Rearrangements and DNA Oligonucleotide Arrays*. PhD thesis, Weizmann Institute of Science, 2004.
- [HP99] S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46:1–27, 1999.
- [HS03] Tzvika Hartman and Ron Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Combinatorial Pattern Matching (CPM '03)*, 2676:156–169, 2003.
- [KS94] J. Kececioglu and D. Sankoff. Efficient bounds for oriented chromosome inversion distance. *Proceedings of CPM 94*, 807:307–325, 1994.
- [KS95] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals with application to genome rearrangement. *Algorithmica*, 13:180–210, 1995.
- [LX01] G.H. Lin and G. Xue. Signed genome rearrangements by reversals and transpositions: Models and approximations. *Theoretical Computer Science*, 259:513–531, 2001.
- [PH88] J. Palmer and L. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *J. Molecular Evolution*, 27:87–97, 1988.
- [YAF05] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21:3340 – 3346, 2005.