Sorting Signed Permutations by Transpositions and Reversals

by

Fei Zhang

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

University of Victoria

Supervisors: Dr. Ulrike Stege and Dr. Hausi A. Müller

ABSTRACT

Large scale comparative genetic mapping offers exciting prospects for understanding genomic evolution and has recently become of interest in computational molecular biology. The genome rearrangement problem is the computational problem of determining the smallest number of evolutionary events required to transform a given genome into another.

In this thesis, we study a specific variant of the genome rearrangement problem. We assume that every genome has exactly one linear chromosome, and that each gene is an oriented unit and appears exactly once per genome. Furthermore, our model allows only two kinds of evolutionary events: reversals and transpositions. The problem is equivalent to the problem of sorting signed permutations by transpositions and reversals.

We explore the characteristics of signed permutations and their sorting path. This exploration results in lower and upper bounds for a shortest sorting path. These bounds help us develop three approximation algorithms. We also prove that sorting by transpositions and reversals is at least as hard as the problem sorting by transpositions only — the complexity of which is unknown. In an effort to implement an algorithm to find an optimal sorting path of events, we designed four techniques to reduce the input size of the problem and thus achieve an improvement of the actual running time for any exhaustive algorithm.

Table of Contents

List of Figures and Tables

ACKNOWLEDGMENTS

I would like to thank everyone who supported me and offered me guidance throughout this research. In particular, the advice of Dr. Ulrike Stege, who inspired me to write this thesis and encouraged me to bring it to fruition, was much appreciated. I would like to thank Dr. Hausi Müller, who gave me guidance in thinking about research and life. Furthermore, without his financial support, most of my research work could not have been accomplished. I would also like to thank Allan Scott and Parissa Agah for their help with brainstorming and proof-reading.

I am also grateful for the support from the Department of Computer Science and the University of Victoria.

Finally, I wish to thank my wife, Fang Yang, for all her encouragement and support.

Chapter 1

Introduction

The study of genome rearrangements has recently become of interest in computational molecular biology. It is the problem of determining the smallest number of evolutionary events required to transform a given genome into another. There are various global rearrangement problems according to the different events and genome characteristics.

In this thesis, we study a specific case of the problem that allows only two kinds of evolutionary events: reversals and transpositions. Further we assume that every genome has exactly one linear chromosome, and that each gene is an oriented unit and appears exactly once per genome. The problem is equivalent to the problem of sorting signed permutations by transpositions and reversals.

The thesis has the following structure:

In Chapter 2 we introduce the genome rearrangement problem, review the existing literature on different variants of the genome rearrangement problem, and summarize the fundamental notions and existing results.

We study the problem of sorting by transpositions and reversals in Chapter 3. At the beginning of this chapter, we introduce some basic definitions and formalize the genome rearrangement problem. We also explore and prove some properties associated with a permutation and its *optimal sorting path*. Then we give lower and upper bounds for the length of such a shortest sorting path and consider the computational complexity analysis of the problem. Finally we describe three different approximation algorithms for this problem. We then present four techniques to reduce the input size of the problem. We give a rough estimate of the running-time improvement of our implementation by applying these techniques, compared to a basic exhaustive search.

In Chapter 4 we draw some conclusions and present ideas for the future work.

The major results of this thesis are as follows:

- We prove a conjecture stated in the literature [SM97], namely that no event in an optimal sorting path affects a sorted substring of the permutation to be sorted [Section 3.1.2].
- Using the concept of breakpoint graph, which is introduced by Bafna and Pevzner [BP96], we show under which circumstances a permutation can be partitioned, e.g., each part can be solved separately [Section 3.1.3].
- We classify all possible events occurring in an optimal sorting path for a permutation into six categories regarding their effects on the change of the number of the cycles in the corresponding breakpoint graph [Section 3.1.4].
- We prove that the computational problem of Sorting by Transpositions and Reversals is at least as hard as Sorting by Transpositions only [Section 3.2.3].
- We present four techniques to reduce problem instances [Section 3.3].

Chapter 2

Motivation and Background

2.1 Motivation

In order to understand evolution, biologists would like to know how all species on earth are related to each other. This problem has been studied for several centuries, using traditional methods from Morphology, Anatomy, Physiology, Paleontology, etc. These methods are based on structural characteristics of the organism (e.g., the construction of inner components of the organism and the connecting mode in the different hierarchy of the organism) [Eri03].

However, these techniques do not use any information of genetic materials. Crick [Cri70] proposed the central dogma of Molecular Biology: "biological information is encoded in DNA, transmitted by DNA replication, transcribed into RNA, and translated into proteins". He used this dogma to backup the idea that DNA and heredity are tightly related. That is, genetic code determines the phenotype and exterior biological traits have their genetic backgrounds. Therefore, it is widely believed that the evolution of genetic material is fundamental to the field of biology. Sequencing and analyses of whole genomes started a revolution in the understanding of evolution.

This study of evolution can detect significant information on the relationships between organisms by examining the history of organisms at the molecular level by comparing the order of bases of nucleic acids. Further it reveals evolutionary mechanisms and allows to effectively examining organisms for which the use of traditional methods has been problematic (such as in bacteria) [LX01]. Therefore, the study of evolution has entered a new era. Especially since Sanger and colleagues [SN77] developed the methods of DNA sequencing in 1977, the availability of sequenced whole genomes has turned this field into a hot topic.

Before the 1990s, most research conducted in molecular evolution used the so-called *edit distance* to measure the evolutionary distance between two species. The edit distance is the minimum number of *insertions*, *deletions*, and *substitutions* required to change one DNA sequence into another. Such an evolutionary event can insert (insertion), deletes (deletion), or replaces (substitution) one or more nucleotides in DNA. We call such events *local mutations*.

In the late 1980s, Palmer and his colleagues [PH88] compared the mitochondrial genomes of *Brassica oleracea* (cabbage) and *Brassica campestris* (turnip). While both genomes are very closely related (most of the genes are 99–99.9% identical) these molecules differ dramatically in their gene order [BP95]. That is, at the chromosome level, the major difference between cabbage and turnip is the order these genes are arranged in and not the contents of the genes. This discovery as well as many other studies [BP95, BP96, HP99] over the last decade shows that the genome rearrangement problem is essential to the understanding of molecular evolution in mitochondrial, chloroplast, viral and bacterial DNA.

In the genome rearrangement model, the evolution of a species consists of mutations that alter the gene order. Primarily, *reversals, transpositions,* and *trans-reversals* are studied. In a *reversal*, a segment of the genome is taken out and put back in reversed order (see Definition 4 in Section 3.1.1). In a *transposition*, a segment of the genomes is moved and placed back at another position in the genome (see Definition 3 in Section 3.1.1). Finally, a *trans-reversal* is the event that removes a segment of genes and places it at another position in reversed orientation in the genome. We call these kinds of events *global mutations*. There are also other global mutations, for example, *duplication* [SE00, TC00] and *translocation* [KR95, Han96]. In this thesis, we mainly focus on reversals and transpositions.

Recently the study of genome rearrangements has drawn a lot of attention. First, large volumes of genomic data on various organisms became available. For the first time a large scale study of evolutionary relations between species became possible by comparing the order in which common genes appear along their chromosomes. Second, changes in the gene order are much less frequent than local mutations. Thus using global mutations, we can deduce the evolutionary history of speciation more precisely and further backwards in time [PS00].

"However, one of the crucial assumptions in the study of genome rearrangements is the Molecular Clock Hypothesis. The Molecular Clock Hypothesis, postulated in 1965 by Emile Zuckerkandl and Linus Pauling [ZP65], asserts that the rate of evolutionary change of any specified protein was approximately constant over time and over different lineages. It has been applied to DNA sequence evolution also, particularly neutral evolution. Subsequent testing has shown that, while the MCH cannot be blindly assumed

to be true" [http://www.fact-index.com/m/mo/molecular_clock_hypothesis.html], i.e., not all genes, proteins and genomes "ticked" at the same rate across all species [Bri86, Behe90, Aya97], it does hold in many cases, and these can be tested for[Fit94, Tho82]. "Knowledge of approximately-constant rate of molecular evolution in particular sets of lineages facilitates establishing the dates of phylogenetic events. The molecular clock concept has been quite successful in the determination of relationship and ancestry among different taxonomic groups present on this planet by allowing to look backwards in time and to deduce certain evolutionary assumptions" [OL92].

The study of genome rearrangements can be done by comparing the gene orders in the studied species and then reconstructing the sequences of events that possibly have transformed the ancestral species into the contemporary species. To consider the more simple case having two species, the goal is to explain the lineages leading to their common ancestral species in the most parsimonious manner (i.e., by as few global mutations as possible) [PS00].

The study of genome rearrangements offers exciting prospects for understanding the evolution of the genome. It is a first step to investigate the distance between genomes and an efficient approach to assess the similarity between genomes in the sense of global mutations. We know that another key factor in genome evolution is local mutation which is handled by classic alignment algorithms. However, the study of genome rearrangements does not handle it. A combination of the two could be used to develop practical algorithms to analyze the evolutionary history of genomes. The evolutionary divergence of two genomes can be represented by a series of (possibly different kinds of) events, possibly involving arbitrarily long substrings of the chromosomes. When

investigating relationships among contemporary species, an arising key problem is to reconstruct a phylogenetic tree that minimizes the total evolutionary distance measured along the tree branches. Here, the study of genome rearrangements can provide a new model when studying the reconstruction of evolutionary trees of more than two species.

The potential of computational methods for analyzing genome rearrangements was first recognized by Sankoff *et al.* [SC90]. The problem of genome rearrangements takes as input two different orders of the same set of genes. If the events permitted are reversals and the input consists of signed permutations, the problem is solvable in Linear time [Bader99]. For unsigned permutations, however, the problem is NP-hard [Cap97]. The only other known polynomial time algorithm for the genome rearrangement problem considers translocations only [Chr98]. The latter problem is not discussed in this thesis. The *rearrangement distance* is the number of elementary events necessary to transform one linear ordering of genes into another. The corresponding rearrangement model prescribes which elementary events are permitted. We view the study of genome rearrangements as solving a "combinatorial puzzle" to find a shortest sequence of global mutations to transform one genome into another. "For genomes consisting of a small number of "conserved blocks", the most parsimonious scenarios may be found easily by brute-force algorithm. However, for genomes consisting of a large number of blocks, finding the optimal solution can be time consuming." [HP95]

As mentioned above, the complexity of this problem under the assumption that all global mutations are reversals has been well studied. Taking into account the direction of each gene, the objects discussed are signed permutations. For example, Hannenhalli *et al.* [HC95] showed that the Epstein-Barr virus (EBV) genome can be transformed into the

Human cytomegalovirus (CMV) genome with five reversals. They first selected the shared 25 genes and designated each of them with a letter and an arrow for gene direction as shown in Figure 2.1. Then they found that these 25 genes are preserved in 7 blocks depicted with different colors in Figure 2.1. Thus, the problem of finding the distance between EBV and CMV can be transformed into the problem of transforming permutation $(\overrightarrow{1},\overleftarrow{2},\overleftarrow{3},\overleftarrow{6},\overleftarrow{5},\overrightarrow{7},\overrightarrow{4})$ into permutation $(\overrightarrow{1},\overrightarrow{2},\overrightarrow{3},\overrightarrow{4},\overrightarrow{5},\overrightarrow{6},\overrightarrow{7})$.

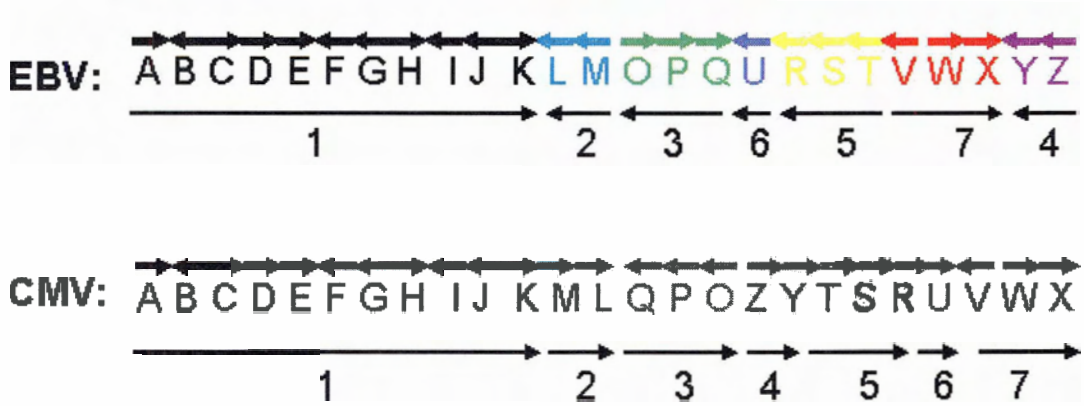

Figure 2.1 Transforming EBV to CMV corresponds to sorting $(\overrightarrow{1},\overleftarrow{2},\overleftarrow{3},\overleftarrow{6},\overleftarrow{5},\overrightarrow{7},\overrightarrow{4})$.

In this thesis, we use π and σ to denote permutations. The problem is viewed as transforming π to σ. For example, if $\pi = (\overrightarrow{5},\overleftarrow{1},\overrightarrow{4},\overleftarrow{3},\overleftarrow{2})$, and $\sigma = (\overrightarrow{4},\overleftarrow{1},\overleftarrow{5},\overrightarrow{3},\overleftarrow{2})$. We relabel the elements of the permutations such that σ is the identity $id = (\overrightarrow{1},\overrightarrow{2},\overrightarrow{3},\overrightarrow{4},\overrightarrow{5})$. Then π is changed respectively to $\pi' = (\overleftarrow{3},\overrightarrow{2},\overrightarrow{1},\overleftarrow{4},\overrightarrow{5})$. Therefore, finding the distance between π and σ is equivalent to finding the shortest sequence of events that sort π'. This substitution does not change the nature of the problem; it just makes the writing easier. In our model, a reversal event on a segment of a permutation inverts not only the order of

the segment, but also the signs of the elements in the segment. It is noted that the study of signed permutations is relevant to genome rearrangements, since genes are oriented in DNA sequences.

For sorting signed permutations by reversals only, Bafna and Pevzner [BP96] designed a 1.5-approximation algorithm[1], and later Hannenhalli and Pevzner [HP99] presented a polynomial-time algorithm which finds the minimum number of reversals for a signed permutation. This is the first polynomial-time algorithm for a model of genome rearrangement problems. The problem of finding the reversal distance between two unsigned permutations is known to be NP-hard [Cap97]. Recently, interest has been directed towards the problem of sorting unsigned permutations by transpositions. Bafna and Pevzner [BP98] designed a 1.5-approximation for this problem; however the computational complexity is still unknown. Although in general sorting signed permutations by transpositions only is impossible (transpositions cannot change the direction of an element), if used combined with reversals, transpositions can reduce the number of rearrangement events required to sort a signed permutation. For example, sorting the signed permutation $\pi = (\overleftarrow{1}, \overleftarrow{4}, \overrightarrow{3}, \overrightarrow{2})$ requires exactly four reversals according to the algorithm by Hannenhalli and Pevzner [HP99] while one reversal and two transpositions suffice to sort π.

In the study of genome rearrangements, the first step is to model the problem. There are many variants of genome rearrangement problems depending on the events permitted. In this thesis, we consider reversals and transpositions only, exclude trans-

[1] k-approximation algorithm: An algorithm to solve an optimization problem that runs in polynomial time in the length of the input and outputs a solution that is guaranteed to be at most (or at least, as appropriate) k times the optimal solution [CRC].

reversal and other global mutations, and treat each gene as a directed unit (either oriented from left to right or from right to left). Further, we assume that no event applied on the genome breaks up a gene. We also suppose that each genome contains only one chromosome and every gene occurs exactly once per genome. We then can model a genome as a string (the chromosome) over a set of signed integers (the genes and their directions). Thus genomes can be modeled as permutations of signed integers. Because the events permitted—reversals and transpositions—cannot insert or delete any genes, we can describe the genome rearrangement problem as the problem of sorting signed permutations by reversals and transpositions. The task is to determine an optimal sorting path, i.e. an optimal sequence of global events that transform one permutation to another.

Besides their application in evolutionary biology, genome rearrangement problems are also interesting as algorithmic problems. Some of them have been shown to be NP-hard, some are polynomial-time solvable, and some are still of unknown computational complexity. Sometimes changing a problem's definition just a little can change an NP-hard problem to a problem in *P*, or vice-versa. Genome rearrangement problems can be viewed as combinatorial puzzles [PS00]. This intellectual challenge is also seen as a motivation.

2.2 Sorting Unsigned Permutations by Reversals

In an unsigned permutation, a reversal inverts the order of a substring of any length. Two examples of reversal events are shown in Figure 2.2. The *reversal distance* $rd(\pi)$ of a given permutation π is the length of an optimal sorting path of reversals for π. Sorting by reversals is the problem of finding a sequence of reversals of length $rd(\pi)$ that

sorts π. The problem of sorting unsigned permutations by reversals is the first variation studied.

5 7 1 6 3 8 2 4

5 7 8 3 6 1 2 4

5 4 8 7 6 3 2 1

1 2 3 6 7 8 4 5

Figure 2.2: Two examples of reversals in unsigned permutations. The reversals act on the underlined substrings.

Watterson *et al.* [WE82] describe a simple heuristic algorithm[2] for sorting by reversals. Given a permutation, this heuristic first moves 1 to the front of the permutation, then moves 2 to the correct position, and so on until the permutation is sorted. This algorithm requires at most n-1 reversals to sort any given permutation. However, this algorithm does not achieve any constant approximation bound.

Kececioglu and Sankoff [KS93] presented the first approximation algorithm for computing the reversal distance between two unsigned chromosomes. This is a greedy approximation that repeatedly applies a reversal that removes as many *breakpoints* (see Section 3.1.1 Definition 6) as possible from the permutation. Ties among reversals are resolved as follows. This algorithm removes at least one breakpoint with every reversal and thus yields a 2-approximation. The algorithm runs in $O(n^2)$ time and $O(n)$ space for permutations of n elements.

[2] A heuristic algorithm: An algorithm that usually, but not always, works or that gives nearly the right answer (from Paul E. Black http://hissa.nist.gov/~black/).

Bafna and Pevzner [BP96] improved the Kececioglu and Sankoff algorithm for signed and unsigned linear permutations. They introduced the elegant concept of the *breakpoint graph rG(π)*. Further they obtained a lower bound for the reversal distance *rd(π)* by considering cycles in the breakpoint graph.

$$n + 1 - C(\pi) \leq rd(\pi)$$

Here, n is the length of permutation π and $C(\pi)$ is the number of cycles in $rG(\pi)$. Based on this graph, an approximation algorithm is developed with a performance guarantee of 1.75. Further this yielded an approximation algorithm for signed permutation with a performance guarantee of 1.5.

Kececioglu and Sankoff [KS95] presented a branch-and-bound algorithm which is an application of maximum-weight matchings, shortest paths, and linear programming. The authors reported that their algorithm sorted random permutations of 30 elements in a few minutes of computer time.

Caprara, Lancia and Ng [CLN99] implemented a branch-and-bound algorithm for computing the reversal distance between two unsigned permutations. This algorithm performs well in practice, sorting random permutations containing up to 100 elements in a few minutes.

The general problem of sorting by reversals is NP-hard [Cap97]. He obtained this result by proving that determining the value $C(\pi)$ is NP-hard.

Since Bafna and Pevzner [BP96] presented the concept of a breakpoint graph r$G(\pi)$, many following results are based on this concept. However, Caprara [Cap99] showed that the lower bound of *rd(π)* based on r$G(\pi)$ is not always tight. He used this

result to explain the experimental performance of algorithms such as that of Caprara, Lancia and Ng [CLN99].

For sorting unsigned permutations by reversal*s*, only approximation and heuristic algorithms exist. The best known approximation is a 1.375-approximation by Berman *et al.* [BH01]. Further, the problem cannot be approximated within 1.0008 [BK99].

2.3 Sorting Signed Permutations by Reversals

In a signed permutation, a reversal not only inverts the order of a substring of any length but also changes the direction of elements in the substring. Two examples of reversal events are shown in Figure 2.3. The reversals act on the underlined substrings.

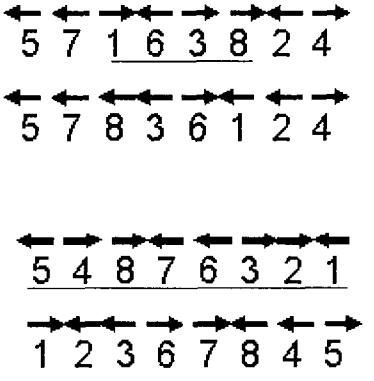

Figure 2.3: Two examples of reversal in signed permutations.

The reversal distance $rd(\pi)$ of a given signed permutation π is the length of an optimal sorting path of reversals that transforms π into the identity permutation. Sorting by reversal is the problem of finding a path of reversals of length $rd(\pi)$ that sorts π. For the problem of sorting signed permutations by reversals only, Bafna and Pevzner [BP96] first describe a 1.5-approximation algorithm.

Hannenhalli and Pevzner [HP99] presented the first polynomial-time algorithm to compute the reversal distance. They proved a conjecture of Kececioglu and Sankoff [KS95] namely that an optimal length sequence of reversals exists that sorts a permutation and does not break apart any strips of length three or more. They obtain a polynomial-time algorithm $O(n^4)$ for the problem sorting by reversals and achieved an $O(n^4)$-time algorithm for sorting signed permutations by reversals by first of all transforming the signed permutation into an equivalent unsigned permutation.

Berman and Hannenhalli [BH96] improved the $O(n^4)$-algorithm contained in [HP95] and produced an $O(n^2\alpha(n))$-algorithm to solve the problem.[3]

Kaplan, Shamir and Tarjan [KS97] present a quadratic time algorithm for finding the minimum number of reversals needed to sort a signed permutation. Their algorithm is faster than the previous algorithm of Hannenhalli and Pevzner [HP99] and its faster implementation by Berman and Hannenhalli [BH96]. The algorithm is conceptually simple and does not require special data structures. Furthermore, they prove that, if the length of the shortest sorting path is required only, this can be determined in $O(n\alpha(n))$ time by using Berman and Hannenhalli's method [BH96].

We further note that if one is interested only in the length of the shortest sequence, and does not require the sequence itself, then it could be calculated in linear time [BM01].

Kececioglu and Sankoff [KS95] gave the first approximation algorithm for the problem of determining the reversal distance between two signed circular permutations.

---

[3] $\alpha(n)$ is the inverse Ackerman function.

Meidanis, Walter and Dias [MW00] present a polynomial time algorithm solving this problem using that this problem is equivalent to a problem on signed linear permutations.

2.4 Sorting by Transpositions

A transposition swaps two adjacent substrings of any length. It has no effect on the direction of any element. We can also interpret a transposition as deleting a substring and subsequently inserting it at another location. In the problem of sorting by transpositions only, the permutation is unsigned. Two examples are shown in Figure 2.4. In each example, the transposition acts on the two underlined substrings.

5 7 1 6 3 8 2 4
1 6 3 5 7 8 2 4

5 4 8 7 6 3 2 1
5 4 8 7 6 2 3 1

Figure 2.4: Two examples of transposition events

The transposition distance $td(\pi)$ of a permutation π is the length of a shortest sequence of transpositions that transforms π into the identity sequence. Sorting by transpositions is the problem of finding a sequence of transpositions of length $td(\pi)$ that sorts π.

Bafna and Pevzner [BP98] analyzed the problem of determining the transposition distance between two unsigned linear chromosomes. They present a metric on a permutation that counts cycles in its breakpoint graph representation. Further they derived upper and lower bounds and presented a 1.5-approximation algorithm that runs in quadratic time.

Christie [Chr99] gave a somewhat simpler but slower $O(n^4)$ algorithm with the same approximation ratio. Hartman [Har03] further simplified Christie's algorithm to achieve an $O(n^2)$ running time.

Eriksson *et al.* [EE01] develop a heuristic that sorts any given permutation of n elements by at most $2n/3$ transpositions.

The computational complexity of the problem sorting by transpositions is an open problem. It is unknown whether the problem sorting by transpositions is NP-hard or solvable in polynomial time.

2.5 Sorting by Transpositions and Reversals

Sorting by reversal*s* and transposition*s* is the problem of finding an optimal path of reversals and transpositions that sorts a given signed permutation. Figure 2.5 shows that permutation $(\overleftarrow{3}, \overleftarrow{1}, \overrightarrow{4}, \overrightarrow{2})$ can be sorted using one reversal and one transposition.

Hannenhalli *et al.* [HC95] used exhaustive search to solve a particular instance of length 7 of sorting signed permutations by transpositions and reversals.

Walter *et al.* [WD98] investigated the problem of sorting an unsigned permutation and gave a 3-approximation algorithm. They also investigated a signed version of sorting by reversals and transpositions. For this problem they described a 2-approximate algorithm, and showed that the upper bound of the reversal and transposition distance is at least $[n/2]+2$.

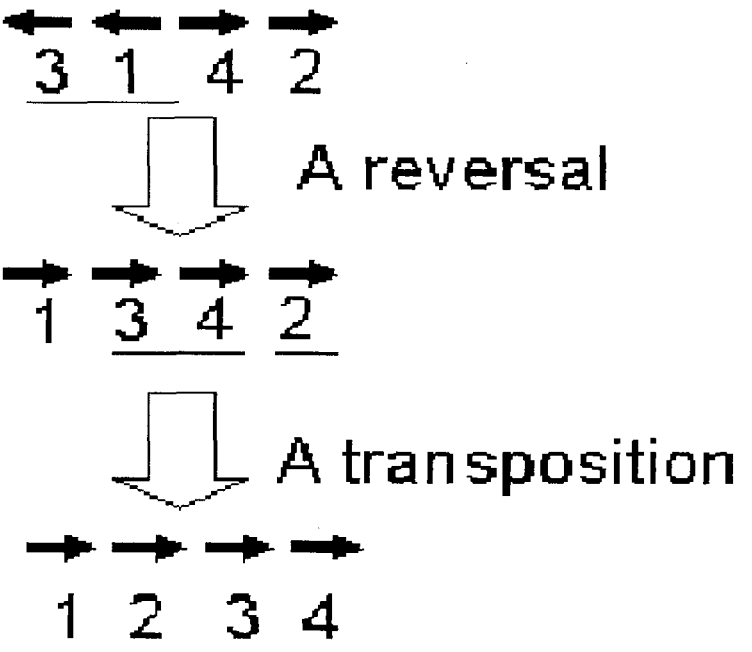

Figure 2.5: Sorting a permutation with transpositions and reversals

Gu, Peng and Sudborough [GP99] gave a lower bound of $\frac{brs(\pi) - C(\pi)}{2}$ for a solution of the problem of sorting signed permutation π by reversals, transpositions and trans-reversals. Here $brs(\pi)$ is the number of breakpoints of π and $C(\pi)$ is the number of cycles in $G(\pi)$. Based on this lower bound, they designed a 2-approximation algorithm, which runs in $O(n^2)$ time.

The computer program "Derange II" by Blanchette, Kunisawa and Sankoff [BK96] is based on a greedy algorithm which minimizes the number of events. However, the problem considered varies from the one stated above. They gave reversals and transpositions different weights. They also found that, when changing the weights signed to reversals and transpositions, the ratio between the numbers of inversions and transpositions is changed. Using simulations, Eriksen *et al.* [ED01] found that the optimal

weights (i.e., introducing least bias) are 1.0 for inversions and 2.0 for transpositions (including inverted transpositions).

Eriksen [Eri02] proposed the following problem: "find an optimal sorting path s of permutation π such that ($inv(s)$+2 $trp(s)$) is minimized, where $inv(s)$ and $trp(s)$ are the numbers of reversals and transpositions (including trans-reversals) in s, respectively. He obtained a polynomial-time approximation algorithm for computing this formula with an accuracy of $(1 + \varepsilon)$, for any $\varepsilon > 0$. He explicitly states a 7/6-approximation algorithm as an example and argues that for most applications the algorithm performs much better than guaranteed."

Lin and Xue [LX01] defined a third transposition event which not only swaps the two adjacent substrings but also inverts both of them. They established three problem models and gave a common lower bound and a 2-approximation algorithm for all three problems.

Chapter 3

Sorting by Transpositions and Reversals

In Section 3.1, we introduce the necessary terminology, which contains most of the prerequisites for the thesis. We also explore some characteristics associated with a permutation and its optimal sorting path. In Section 3.2, we give lower and upper bounds for the length of an optimal sorting path of a permutation and perform a computational complexity analysis for the problem of finding the length of an optimal sorting path. We describe three approximation algorithms for this problem in Section 3.3. In an effort to implement an algorithm to find an optimal sorting path of events, we present four techniques to reduce the size of the input of the problem and thus achieve an improvement of the actual running time of a naive exhaustive algorithm.

The main results in this chapter are:

- No event in an optimal sorting path breaks sorted substrings (Section 3.1.2).
- Part of a permutation represented by a component in the breakpoint graph can be sorted separately (Section 3.1.3).
- A reversal increases the number of cycles by one (Section 3.1.4).
- There are six different possible events in an optimal sorting path regarding their effects on the change of the number of the cycles (Section 3.1.4).

- Sorting by transpositions and reversals is at least as hard as the problem of sorting by transpositions only (Section 3.2.2).
- We introduce four techniques to reduce the problem input (Section 3.3).

3.1 Basic Definitions and Problem Formulization

Comparative gene-order genomics is a relatively new discipline which seeks to apply computational methods to problems in the field of biology. The first step in such an application is modeling: i.e., the translation of a problem to the language of mathematics or computer science. In this process, different researchers model a problem following different criteria and may introduce new terms to facilitate their research. It is thus necessary to firmly state the notation used in this thesis, as nothing may yet be regarded as standard. Then we define our problem in two different versions—a decision version and an optimization version. In this section, we also explore some characteristics associated with a permutation and its optimal sorting path.

3.1.1 Basic definitions

Definition 1: A *signed permutation* $\pi = (\pi_1, \pi_2, \cdots, \pi_{n-1}, \pi_n)$ of length n, with $\pi_i = (\text{value}(\pi_i), \text{sign}(\pi_i))$, is defined as follows:

(1) $(\text{value}(\pi_1), \text{value}(\pi_2), \cdots, \text{value}(\pi_{i-1}), \text{value}(\pi_i))$ is a permutation over $\{1,2,\cdots,n\}$;

(2) For $1 \le i \le n$, $\text{sign}(\pi_i) \in \{\rightarrow, \leftarrow\}$.

For simplicity, for an element $\pi_i = (\text{value}(\pi_i), \text{sign}(\pi_i))$ in a signed permutation π, we write $\overrightarrow{value(\pi_i)}$ if $\text{sign}(\pi_i) = \rightarrow$ and $\overleftarrow{value(\pi_i)}$ if $\text{sign}(\pi_i) = \leftarrow$.

Definition 2: Let π be a signed permutation of length n. The *identity permutation* of π is defined as: $id(\pi) = (\vec{1}, \vec{2}, \cdots, \vec{n})$. We may write id for $id(\pi)$.

Definition 3: Let $\pi = (\pi_1, \pi_2, \cdots, \pi_{n-1}, \pi_n)$ be a signed permutation of length n. We define the event of a *transposition* e on π as follows. Let $e = [a, i, j]$ with $1 \le a < i < j \le n$.

Then $\pi \circ e = (\pi_1, \pi_2, \cdots, \pi_{a-1}, \pi_i, \pi_{i+1}, \cdots, \pi_{j-1}, \pi_a, \pi_{a+1}, \cdots, \pi_{i-1}, \pi_j, \pi_{j+1}, \cdots, \pi_n)$.

Definition 4: Let $\pi = (\pi_1, \pi_2, \cdots, \pi_{n-1}, \pi_n)$ be a signed permutation of length n. We define the event of a *reversal* e on π as follows. Let $e = [i, j]$ with $1 \le i < j \le n+1$. Then $\pi \circ e = (\pi_1, \cdots, \pi_{i-1}, (value(\pi_{j-1}), \overline{sign(\pi_{j-1})}), (value(\pi_{j-2}), \overline{sign(\pi_{j-2})}), \cdots, (value(\pi_i), \overline{sign(\pi_i)}), \pi_j, \cdots, \pi_n)$

where $\overline{sign(\pi_h)} = \begin{cases} \rightarrow & if\ sign(\pi_h) = \leftarrow \\ \leftarrow & if\ sign(\pi_h) = \rightarrow \end{cases}$.

Definition 5: Let π be a signed permutation of length n. We say that *the computational problem of sorting by transpositions and reversals (STR)* is the problem of determining a shortest sequence of events that transform π into the identity $id(\pi)$. Here each event is either a transposition or a reversal. We denote by $E(\pi)$ a shortest sequence of events (reversals and transpositions) that transforms π into $id(\pi)$, and $K_{min}(\pi)$ the length of $E(\pi)$. Then we define this problem in both the *decision version* and the *optimization version*:

Decision Version (STR-Dec):

Input: A signed permutation π of length n and a positive integer K.

Output: Is $K_{min}(\pi) \le K$?

Optimization Version (STR-Opt):

Input: A signed permutation π of length n.

Output: What is the $E(\pi)$?

Definition 6: Let π be a signed permutation of length n. With $0 \le i \le n$, $\pi_0 = \vec{0}$ and $\pi_{n+1} = \overrightarrow{n+1}$, we say that (π_i, π_{i+1}) is a *breakpoint* of π if any one of the following conditions is true:

(1) $|\ \text{value}(\pi_i) - \text{value}(\pi_{i+1})\ | \neq 1$;

(2) $\text{sign}(\pi_i) \neq \text{sign}(\pi_{i+1})$;

(3) $\text{value}(\pi_i) - \text{value}(\pi_{i+1}) = 1$ and $\text{sign}(\pi_i) = \text{sign}(\pi_{i+1}) = \rightarrow$;

(4) $\text{value}(\pi_{i+1}) - \text{value}(\pi_i) = 1$ and $\text{sign}(\pi_i) = \text{sign}(\pi_{i+1}) = \leftarrow$.

We denote the number of breakpoints of π by $brs(\pi)$. Note that $brs((id(\pi)) = 0$.

Definition 7: Let $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$ be a signed permutation of length n. A *strip* S of π is defined as follows:

(1) S is a substring of π, i.e., $S = (\pi_i, \pi_{i+1}, \cdots, \pi_j)$ for some $1 \le i \le j \le n$.

(2) S is breakpoint-free, i.e., S has only one element or for all $i \le h < j$, (π_h, π_{h+1}) is not a breakpoint of π.

(3) (π_{i-1}, π_i) and (π_j, π_{j+1}) are breakpoints.

Note that every element in the same strip S has the same direction, i.e., $\text{sign}(\pi_i) = \text{sign}(\pi_{i+1}) = \cdots = \text{sign}(\pi_{j-1}) = \text{sign}(\pi_j)$. We define a strip S as an *increasing strip* if the direction of its elements is $\rightarrow$, otherwise S is a *decreasing strip*.

Definition 8: Let $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$ be a signed permutation of length n. Let $S = (\pi_i, \pi_{i+1}, \cdots, \pi_j)$ be a strip π and let e be an event on π. We say *e breaks S* if:

(1) $e = [a, b, c]$ is a transposition and $\{a, b, c\} \cap \{i, i+1, \cdots, j\} \neq \emptyset$ or

(2) $e = [a, b]$ is a reversal and $\{a, b\} \cap \{i, i+1, \cdots, j\} \neq \emptyset$.

3.1.2 Simplified permutations

Theorem 3.1: Let π be a signed permutation of length n. Then there exists an optimal sorting path $E(\pi)$ such that for any $e \in E(\pi)$, e does not break any strip of π.

Proof: Note that permutation π has $(brs(\pi)-1)$ strips. We denote these $(brs(\pi)-1)$ strips of π with $S_1, S_2, \cdots, S_{brs(\pi)-1}$ (in order of occurrence from left to write). We further define a signed permutation σ of length $(brs(\pi)-1)$ as follows. For each strip S_i of π we define a symbol σ_i = (value(σ_i), sign(σ_i)). To determine value(σ_i), we sort $S_1, S_2, \cdots, S_{brs(\pi)-1}$ in increasing order of their smallest values (which is the first or last element of each S_i, depending on its direction). We set value(σ_i) = k if S_i is the k-th smallest element of S_1, $S_2, \cdots, S_{brs(\pi)-1}$. We set sign($\sigma_i$) = $\leftarrow$ if S_i is a decreasing strip and sign(σ_i) = $\rightarrow$ if S_i is an increasing strip. We call σ the *simplified permutation* of π.

Example 1: Let $\pi = (\overrightarrow{5}, \overrightarrow{6}, \overleftarrow{1}, \overleftarrow{9}, \overleftarrow{8}, \overleftarrow{7}, \overrightarrow{4}, \overleftarrow{3}, \overleftarrow{2})$. The set of breakpoints of π is $\{(\overrightarrow{0}, \overrightarrow{5}), (\overrightarrow{6}, \overleftarrow{1}), (\overleftarrow{1}, \overleftarrow{9}), (\overleftarrow{7}, \overrightarrow{4}), (\overrightarrow{4}, \overleftarrow{3}) (\overleftarrow{2}, \overrightarrow{10})\}$ with $brs(\pi) = 6$ and the five strips of π are $S_1 = (\overrightarrow{5}, \overrightarrow{6})$, $S_2 = (\overleftarrow{1})$, $S_3 = (\overleftarrow{9}, \overleftarrow{8}, \overleftarrow{7})$, $S_4 = (\overrightarrow{4})$, and $S_5 = (\overleftarrow{3}, \overleftarrow{2})$. Sorting these strips in increasing order yields $S_2 < S_5 < S_4 < S_1 < S_3$. Therefore $\sigma = (\overrightarrow{4}, \overleftarrow{1}, \overleftarrow{5}, \overrightarrow{3}, \overleftarrow{2})$.

We first observe that every sequence of events that sorts σ also sorts π, and thus sorting π does not require more events than sorting σ, i.e.,

$$K_{min}(\pi) \leq K_{min}(\sigma) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Using π and σ we construct a sequence σ^* as follows. We substitute each increasing strip S_i of π with a strip $S^*_i = (*, *, \cdots, *, \sigma_i)$ where $|S^*_i| = |S_i|$. Further we substitute each decreasing strip S_i of π with a strip $S^*_i = (\sigma_i, *, *, \cdots, *)$ where $|S^*_i| = |S_i|$. We call $\sigma^* = (S^*_1, S^*_2, \cdots, S^*_{brs(\pi)-1})$ the *mimic permutation* of π. Here $id(\sigma^*)$ is the same as $id(\sigma)$ if we ignore all wild characters * in σ^*.

Example 2: As in Example 1, $\pi = (\overrightarrow{5}, \overrightarrow{6}, \overleftarrow{1}, \overleftarrow{9}, \overleftarrow{8}, \overleftarrow{7}, \overrightarrow{4}, \overleftarrow{3}, \overleftarrow{2})$ and $\sigma = (\overrightarrow{4}, \overleftarrow{1}, \overleftarrow{5}, \overrightarrow{3}, \overleftarrow{2})$. We receive $\sigma^* = (*, \overrightarrow{4}, \overleftarrow{1}, \overleftarrow{5}, *, *, \overrightarrow{3}, \overleftarrow{2}, *)$.

We know that σ^* differs from π in the way that some elements of π are substituted with the wild character * in σ^*. So a sequence of events that sorts π can sort σ^* as well, i.e.

$$K_{min}(\sigma^*) \leq K_{min}(\pi) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \ (2)$$

We observe that σ differs from σ^* in that all wild characters of σ^* are removed. Every event on σ^* can be mimicked on σ in the following ways:

(I) If an event on σ^* is a reversal and the reversed segment consists of wild characters only, then the reversal will not be counted as an event in the path for sorting σ;

(II) If an event on σ^* is a transposition and one of the two segments that exchange their locations consists of wild characters only, then the transposition will not be counted as an event in the path for sorting σ;

(III) Except the two kinds of events in (I) and (II), all events on σ^* can be mimicked on σ by ignoring their wild characters.

Therefore, sorting σ does not require more events than sorting σ^* does, i.e.,

$$K_{min}(\sigma) \leq K_{min}(\sigma^*) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

From formula (2) and (3), we get:

$$K_{min}(\sigma) \leq K_{min}(\pi) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

We illustrate how to mimic every event on π to an event on σ in Table 3.1 step by step. In this particular example, the number of events used to sort σ is one less than that used to sort π.

Table 3.1: Every event in $E(\pi)$ is mimicked by an event in $E(\sigma)$.

events	π	σ^*	σ	events
	5 6 1 9 8 7 4 3 2	* 4 1 5 * * 3 2 *	4 1 5 3 2	
Reversal	5 4 7 8 9 1 6 3 2	* 3 * * 5 1 4 2 *	3 5 1 4 2	Reversal
Transposition	5 6 3 2 4 7 8 9 1	* 4 2 * 3 * * 5 1	4 2 3 5 1	Transposition
Reversal	5 4 2 3 6 7 8 9 1	* 3 * 2 4 * * 5 1	3 2 4 5 1	Reversal
Transposition	4 2 3 5 6 7 8 9 1	3 * 2 * 4 * * 5 1	3 2 4 5 1	
Transposition	1 4 2 3 5 6 7 8 9	1 3 * 2 * 4 * * 5	1 3 2 4 5	Transposition
Transposition	1 2 3 4 5 6 7 8 9	1 * 2 3 * 4 * * 5	1 2 3 4 5	Transposition

From Inequalities (1) and (4), we receive that $K_{min}(\sigma) = K_{min}(\pi)$, i.e., an optimal sorting path of π has the same length as an optimal sorting path of σ. We know that every sequence of events that sorts σ also sorts π. Thus an optimal sorting path of σ is an optimal sorting path of π as well, i.e., $E(\pi) = E(\sigma)$.

Because every strip of permutation σ is of length one, i.e., there is exactly one element between every two consecutive breakpoints in permutation σ, it is impossible to break any strips of σ in an optimal sorting path. Therefore none of the events in $E(\pi)$ interrupts any strip of π. This proves Theorem 3.1.

From Theorem 3.1, we conclude:

Corollary 3.1: Let π be a signed permutation of length n, and let $E(\pi)$ be an optimal sorting path that sorts π. Let e be an event that breaks at least one strip of π, and let $\pi' = \pi \circ e$. Then $e \notin E(\pi)$, i.e., $K_{min}(\pi) \leq K_{min}(\pi')$.

As a consequence of Theorem 3.1, from now on, we consider the problem of sorting a permutation π to be the problem of sorting its simplified permutation. That is, whenever we mention a signed permutation π of length n, we assume that π is a simplified permutation. Then $brs(\pi) = n+1$.

Following the methods described in [BP96], we transform a signed permutation π of length n into an unsigned permutation $\pi' = (\pi'_1, \pi'_2, \cdots, \pi'_{2n})$ of length $2n$. We replace each element π_i of π by the elements $\pi'_{2i} = (2\text{value}(\pi_i) - 1)$ and $\pi'_{2i+1} = 2\text{value}(\pi_i)$ if $\text{sign}(\pi_i) = \rightarrow$ and by the elements $\pi'_{2i} = 2\text{value}(\pi_i)$ and $\pi'_{2i+1} = (2\text{value}(\pi_i) - 1)$ otherwise. We call π' the *unsigned simplified permutation* of π. For example, for $\pi = (\overset{\rightarrow}{4}, \overset{\leftarrow}{1}, \overset{\leftarrow}{5}, \overset{\rightarrow}{3}, \overset{\leftarrow}{2})$, we receive $\pi' = (7, 8, 2, 1, 10, 9, 5, 6, 4, 3)$. We claim that sorting π is equivalent to the problem of sorting π' when we treat $2i-1$ and $2i$ of π' as a unit, i.e., there is no breakpoint between $2i-1$ and $2i$. Note that for each event on π, there is a corresponding event on π' and every optimal sorting path that sorts π also sorts π'.

3.1.3 Breakpoint graph

For a signed permutation π of length n, we use its unsigned simplified permutation π' to construct the so-called *breakpoint graph* $G(\pi) = (V, E)$ (where V is $G(\pi)$'s vertex set and E is a collection of edges, $V \subseteq E \times E$) of permutation π, which is received as follows. We first extend the permutation π' to the unsigned simplified permutation π" by adding 0 before the first element of π' and $(2n+1)$ after the last element, i.e., $\pi"_0 = 0$, $\pi"_1 = \pi'_1$, $\pi"_2 = \pi'_2$, ..., $\pi"_{2n} = \pi'_{2n}$, $\pi"_{2n+1} = 2n+1$, where n is the length of permutation π. A pair $(\pi"_i, \pi"_{i+1})$ of unsigned simplified permutation π" is defined to be a *breakpoint* if and only if $|\pi"_{i+1} - \pi"_i| \neq 1$ where $0 \leq i \leq 2n$.

We define the breakpoint graph $G(\pi) = (V, E)$ as follows.

Let $V=\{\pi"_0, \pi"_1, ..., \pi"_{2n+1}\}$, and let $E = E_R \cup E_D$ with

(1) for all i, $0 \leq 1 \leq 2n+1$, edge $(\pi"_i, \pi"_{i+1}) \in E_R$ if $(\pi"_i, \pi"_{i+1})$ is a breakpoint in π".

(2) for all i, $0 \leq 1 \leq 2n+1$, edge $(2i, 2i+1) \in E_D$.

For any signed permutation π, the diagram representing $G(\pi)$ can be arranged in a circle as illustrated in the example in Figure 3.1. Here reality edges go along the circumference and desire edges are represented by chords. Edge set E forms cycles of alternating edge types. The length of such a cycle is determined by the number of its reality edges. If a cycle contains exactly k reality edges, we call it a *k-cycle*. If elements $2i$ and $2i+1$ are consecutive in π", the diagram has cycles of length one at the places where we do not have a breakpoint in π (we do not need to touch these 1-cycles when sorting π), which is depicted as a red cycle in Figure 3.1. The decomposition of $G(\pi)$ into cycles

is unique and by $C(\pi)$ we denote the number of cycles in $G(\pi)$. As in Figure 3.1, $C(\pi) =$ 4.

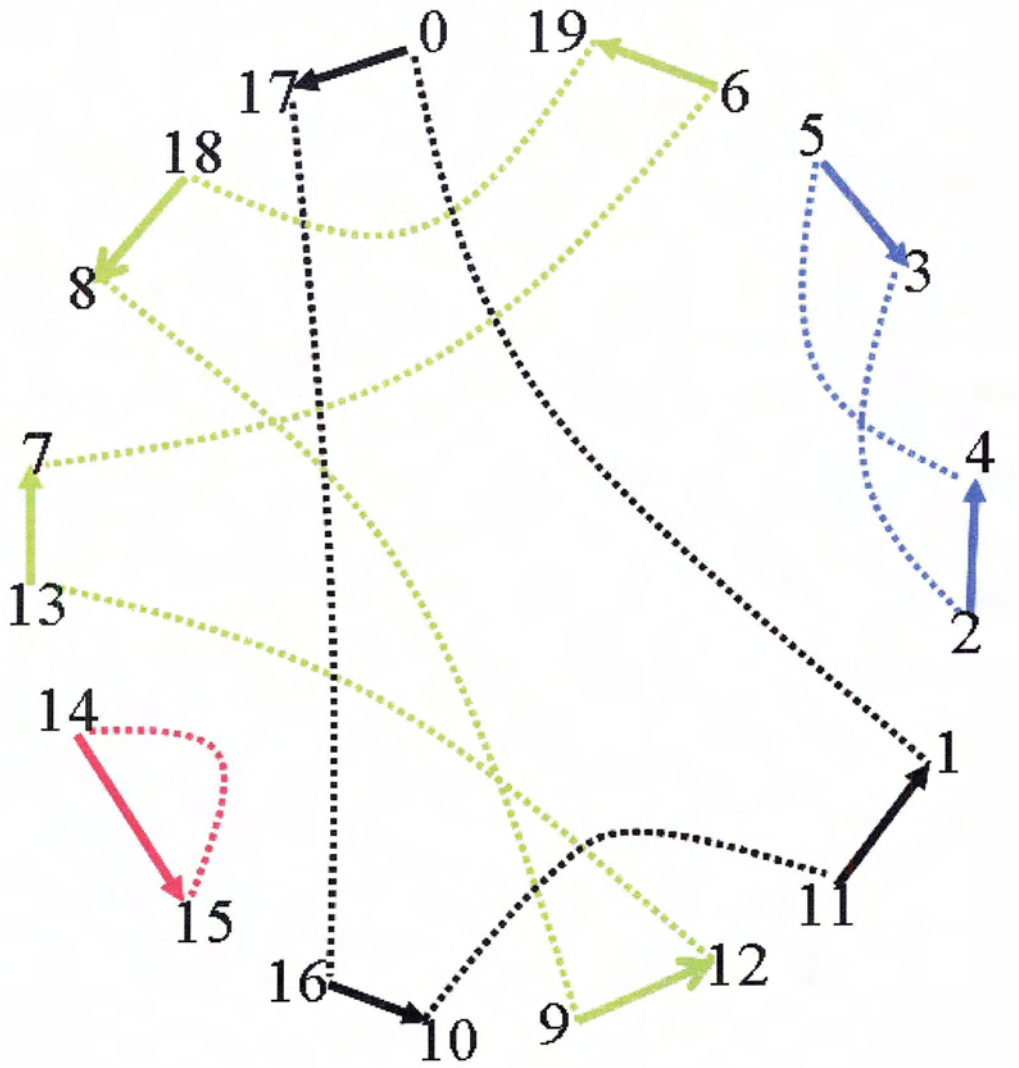

Figure 3.1: The breakpoint graph $G(\pi)$ of $\pi = \overset{\rightarrow}{9}\ \overset{\leftarrow}{4}\ \overset{\rightarrow}{7}\ \overset{\rightarrow}{8}\ \overset{\leftarrow}{5}\ \overset{\leftarrow}{6}\ \overset{\rightarrow}{1}\ \overset{\leftarrow}{2}\ \overset{\rightarrow}{3}$. "$\pi$" = (0, 17, 18, 8, 7, 13, 14, 15, 16, 10, 9, 12, 11, 1, 2, 4, 3, 5, 6, 19). Reality edges are shown with straight arrowed lines; desire edges are dotted curves. It contains four cycles: Black: C_1 is a 3-cycle; Green: C_2 is a 4-cycle; Red: C_3 is a 1-cycle and Blue: C_4 is a 2-cycle. There are three components: C_1 and C_2 constitute a component, and C_4 and C_3 constitute a component each.

We say that two reality edges of a cycle are *convergent* if, when we traverse the cycle, one reality edge is traversed in clockwise order and the other one is traversed in counterclockwise order. Otherwise, we say they are *divergent*. If we label all reality edges in their appearance order from left to right in permutation π by assigning labels from 1 to $brs(\pi)+1$, we can denote a cycle by a bracket notation containing a sequence of reality edges starting from the smallest reality edge following the traversal. We say a cycle is *open* if every pair of two reality edges of the cycle is divergent and the pairs are listed in increasing order in bracket notation. For example, in Figure 3.1, we have four cycles $C_1 = [1, 5, 7]$, $C_2 = [2, 6, 3, 10]$, $C_3 = [4]$ and $C_4 = [8, 9]$. Only C_1 is an open cycle. Note that an open cycle cannot constitute a component.

Consider two cycles C_m and C_n. If we cannot draw a straight line through the circle such that the elements of C_m are on one side of the line and the elements of C_n are on the other side (i.e., C_m and C_n cross each other), then we call these two cycles *inseparable*. If two cycles are inseparable, we say they belong to the same *component*. Otherwise, we say two cycles are *separable* and belong to different components. For example, in Figure 3.1, C_1 and C_2 are inseparable and belong to the same component while C_1 and C_3 are separable and belong to different components. 1-cycle builds a single component. 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 $C_1, C_2, \cdots, C_j$ such that, for all $1 \leq i \leq j\text{-}1$, C_i and C_{i+1} are inseparable. In this situation, all j cycles belong to the same component. Therefore, each cycle belongs to exactly one component and a component may contain more than one cycle.

We observe that sorting π is the process that transforms $G(\pi)$ to $G(id)$, which consists of $(n+1)$ 1-cycles. For example, sorting π in Figure 3.1 corresponds to transforming the graph in Figure 3.2 a) to the graph in Figure 3.2 b). Note that, when we transform graph $G(\pi)$ to $G(id)$, we can ignore the vertex values. Therefore, in the graphic view, events needed to transform $G(\pi)$ to $G(id)$ do not depend on the actual permutation π, but only depend on the shape of cycles, the way these cycles consist of components, and the way components make up the graph.

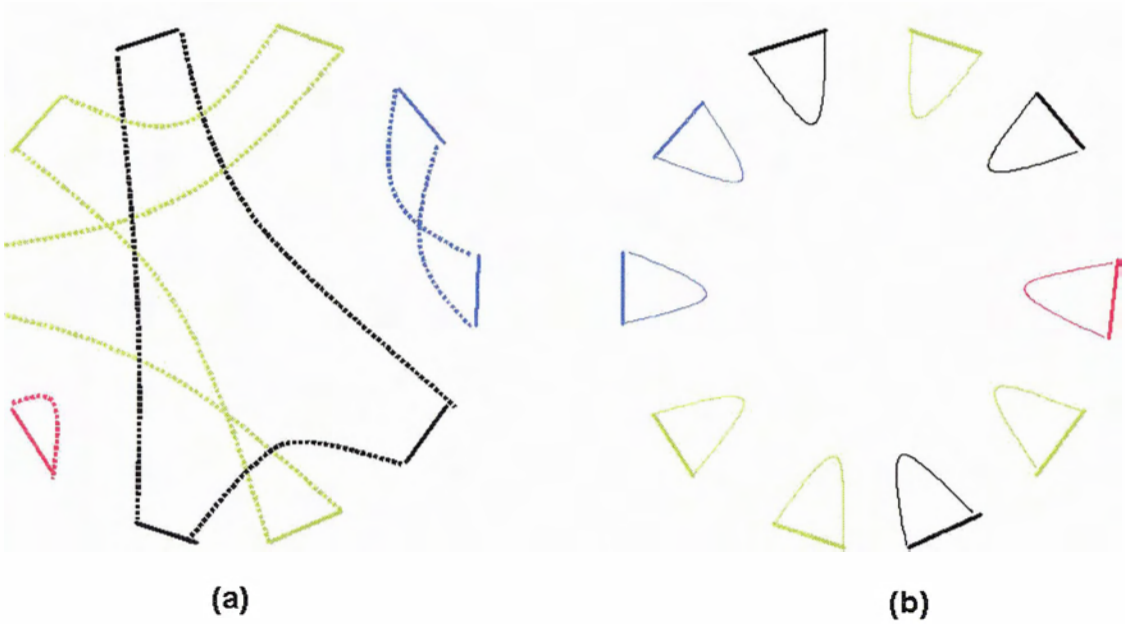

Figure 3.2: Sorting the permutation in Figure 3.1 is the process of transforming graph a) to graph b).

We next draw a connection between each component of $G(\pi)$ with all components of $G(id)$ as follows. A component in $G(\pi)$ corresponds to a series of consecutive components in $G(id)$. That means that each component in $G(\pi)$ represents a sorted substring of identity. For example, the blue 2-cycle in Figure 3.2 a) corresponds to the two adjacent blue 1-cycles in Figure 3.2 b). We know that a sorted substring is a single entity by applying the simplified permutation technique described in the previous section.

Therefore, we build a one-to-one relationship between a component of $G(\pi)$ and a certain consecutive part of the identity, and we can sort the permutation by sorting each component into a series of consecutive 1-cycles independently. This can be viewed as a type of divide-and-conquer algorithm. To do so, we regard components of $G(\pi)$ as permutations, by identifying them with one of the shortest permutations whose breakpoint graphs consist of this component only. We call this process the *instantiation* of a component. For example, the 2-cycle component in Figure 3.1 can be identified with the permutation $(\overrightarrow{6}, \overleftarrow{2}, \overrightarrow{5}, \overleftarrow{3}, \overleftarrow{4}, \overrightarrow{1})$ (shown in Figure 3.3 a) and the component of C_4 can be identified with the permutation $(\overleftarrow{1})$. We do not need to do anything for the component of C_3. So the problem of sorting permutation $(\overrightarrow{9}, \overrightarrow{2}, \overrightarrow{6}, \overrightarrow{7}, \overrightarrow{8}, \overleftarrow{1}, \overrightarrow{3}, \overrightarrow{4}, \overrightarrow{5})$ is transformed to the problem of sorting $(\overrightarrow{6}, \overleftarrow{2}, \overrightarrow{5}, \overleftarrow{3}, \overleftarrow{4}, \overrightarrow{1})$ and that of sorting $(\overleftarrow{1})$.

Since we could sort a permutation by sorting its components independently, we have another characteristic of an event in an optimal sorting path:

Theorem 3.2: Let π be a signed permutation of length n. Then there exists an optimal sorting path $E(\pi)$ such that each event of $E(\pi)$ only acts on the reality edges in at most one component of $G(\pi)$.

During the instantiation of a component, we disregard the particular permutation it is part of and we only care about the shape of the component and the way the component is constructed. Therefore, a component can be instantiated into different permutations. For example, the 2-cycle component in Figure 3.1 can also be treated as

$(\overleftarrow{4,6,5,1,2,3})$ (shown in Figure 3.3 b) as long as they have the same decomposition of breakpoint graph.

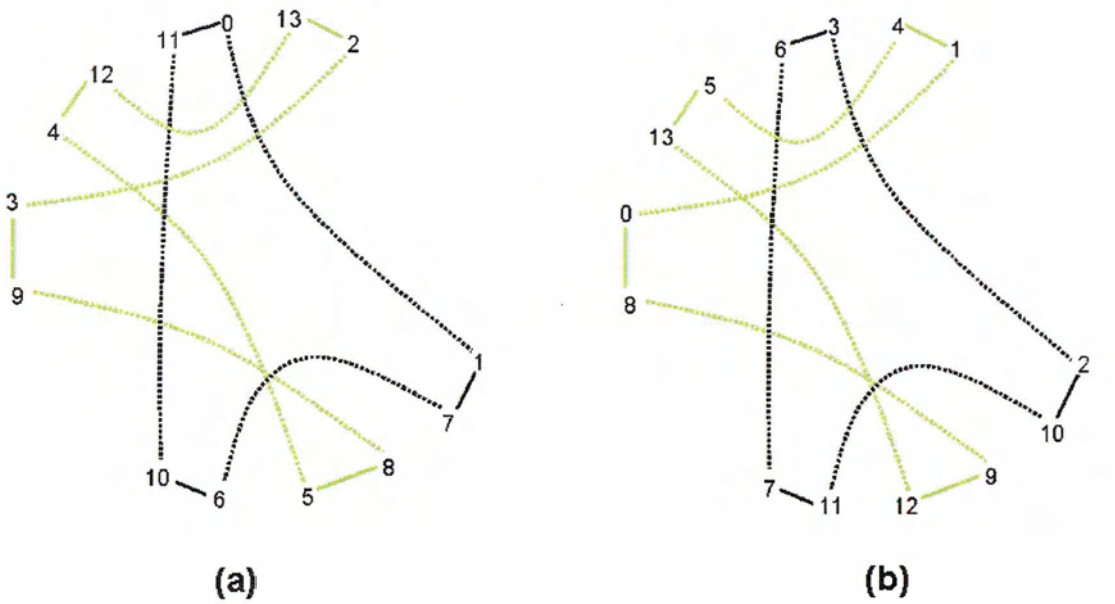

Figure 3.3: Different instantiation of a component

3.1.4 Possible events in an optimal sorting path

Note that the breakpoint graph $G(id)$ of the identity is the only one having $n+1$ cycles. So any sequence of events sorting π must change the number of cycles from $C(\pi)$ to $n+1$. For a permutation π and an event e, we denote by $\Delta C(\pi, e)$ the difference between $C(\pi \cdot e)$ and $C(\pi)$. $\Delta C(\pi, e)$ is also called the gain in the number of cycles due to event e applied to π. We say event e is a *k-move* if $\Delta C(\pi,e) = k$.

A reversal acts on two reality edges. Figure 3.4 shows all three possibilities of a reversal according to different combinations of the two reality edges in a breakpoint graph. The two reality edges may belong to two cycles as presented in Case 2, or may

belong to one cycle as in Case 1 and Case 3 of Figure 3.4. In Case 1 the two reality edges are convergent while they are divergent in Case 3. Note that the reversal is a 1-move in Case 1, while it is a (-1)-move in Case 2 and a 0-move in Case 3.

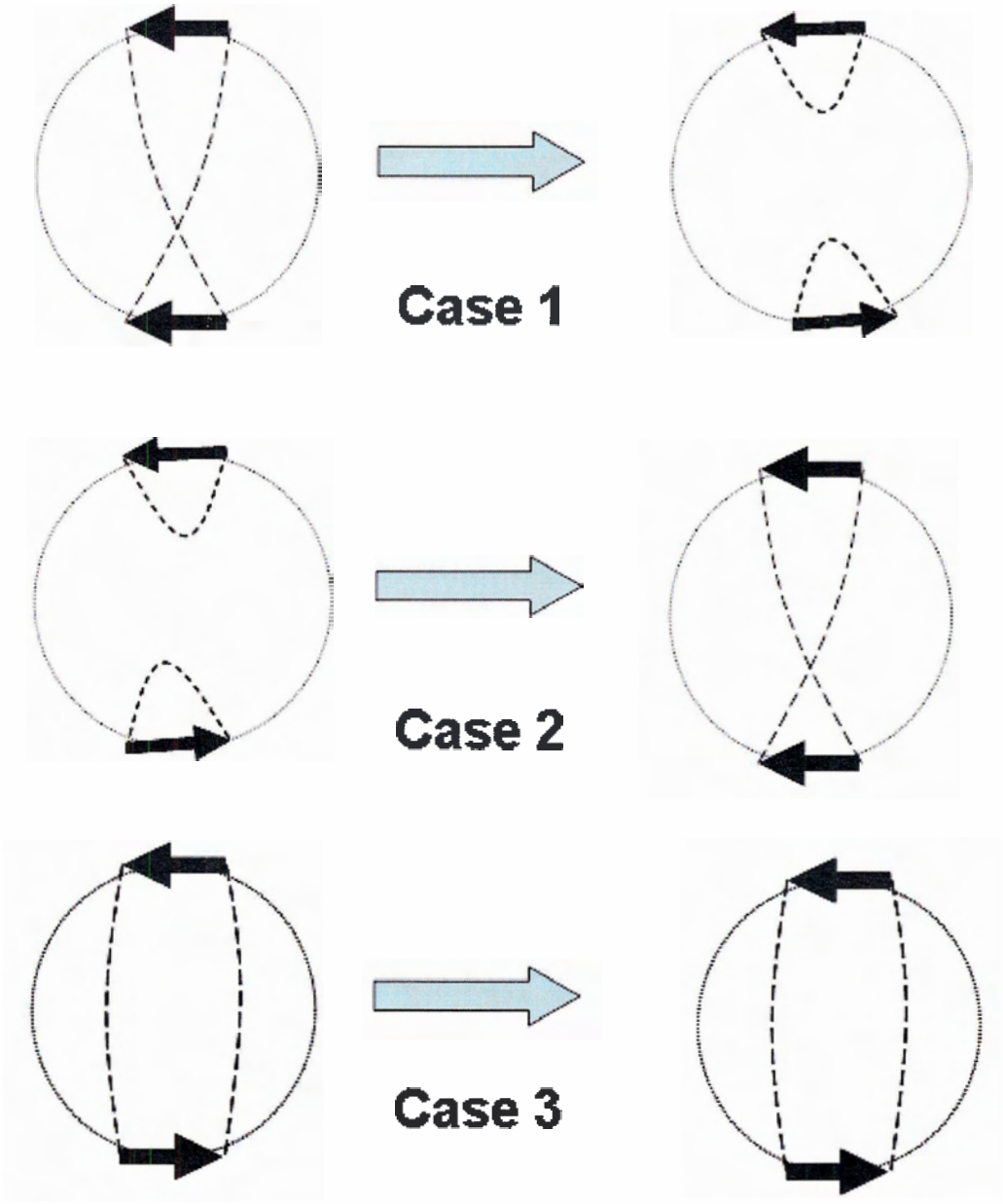

Figure 3.4: All possibilities of a reversal on a signed permutation. Only affected cycles are shown. A dashed line indicates a path formed by one or more desire and reality edges. The arrows represent reality edges and their directions when traversing counterclockwise. The reversal acts on the two reality edges of the left graph in each case.

A conjecture mentioned by [Chr98] says that an optimal sorting path does not contain an event that decreases the number of cycles, i.e., no negative-move is contained in an optimal sorting path. Therefore, assuming the correctness of this conjecture, the reversal in Case 2 never happens in an optimal sorting path. Furthermore, we conclude:

Claim 3.1: The 0-move reversal in Case 3 does not exist in an optimal sorting path of events that sorts π.

Proof: As depicted in the left-upper corner of the breakpoint graph in Figure 3.5, arrow BA and arrow CD are two divergent reality edges. Therefore, there must be at least one reality edge XY on arc AC and ZW on arc DB. Reality edges XY and ZW can be divergent or convergent. We prove that 0-move reversals are not necessary when they are divergent, while we do not show the proof when they are convergent because the proofs under the two situations are similar.

We suppose that the 0-move reversals are contained in $E(\pi)$, then after the 0-move reversal acting on reality edges BA and CD, we have the right-upper corner breakpoint graph of Figure 3.5, where reality edges are DA and CB now. Furthermore, arc DB and reality edges ZW have changed their direction. To sort the permutation we need a reversal on the reality edges XY and ZW. This reversal will make reality edges DA and CB become convergent (shown in the right-middle breakpoint graph). Therefore, we need another reversal on the reality edges of DA and CB and the result is showed in the bottom breakpoint graph. In sum, we need three reversals sorting the left-upper corner breakpoint graph to the bottom breakpoint graph in Figure 3.5 by allowing 0-move reversal. However, we only need two events (transpositions) sorting the left-upper corner breakpoint graph to the bottom breakpoint graph when we do not allow 0-move reversals.

The whole process is shown on the left side in Figure 3.5. Therefore, whenever a path has 0-move reversals, it can not be an optimal sorting path. This proves Claim 3.1.

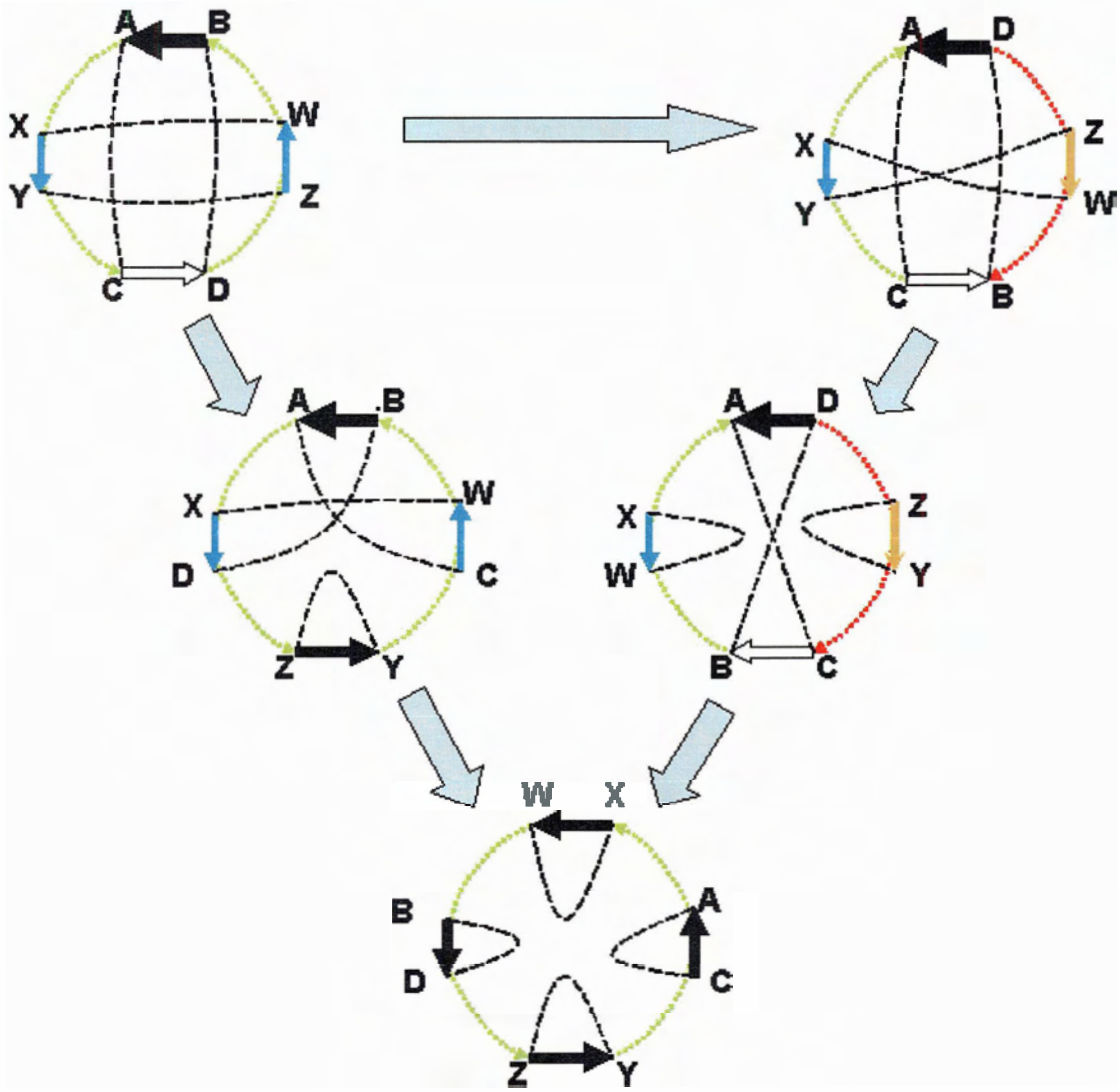

Figure 3.5: A path with a 0-move reversal compared to a path without 0-move reversals. Only affected cycles are shown. All arrows are reality edges. A black dashed line indicates a path formed by one or more desire and/or reality edges.

As a conclusion for the three cases of a reversal in Figure 3.4, there is only one possible reversal in an optimal sorting path which is depicted in Case 1 of Figure 3.4. We obtain Theorem 3.3:

Theorem 3.3: Assuming Christie's conjecture holds, and then a reversal in an optimal sorting path acts on two convergent reality edges in the same cycle and increases the number of cycles by one.

A transposition acts on three reality edges. Figure 3.6 shows all seven possibilities of a transposition according to different combinations of the three reality edges in a breakpoint graph. The three reality edges may belong to three cycles as presented in Case 4, two cycles as presented in Case 2 and 7, or one cycle as presented in other cases in Figure 3.6. We say, in Case 3, the three reality edges are parts of a *full oriented cycle*. We assume that an optimal sorting path does not contain the transposition in Case 2 and 4 from Christie's conjecture [Chr98] because it is a negative move.

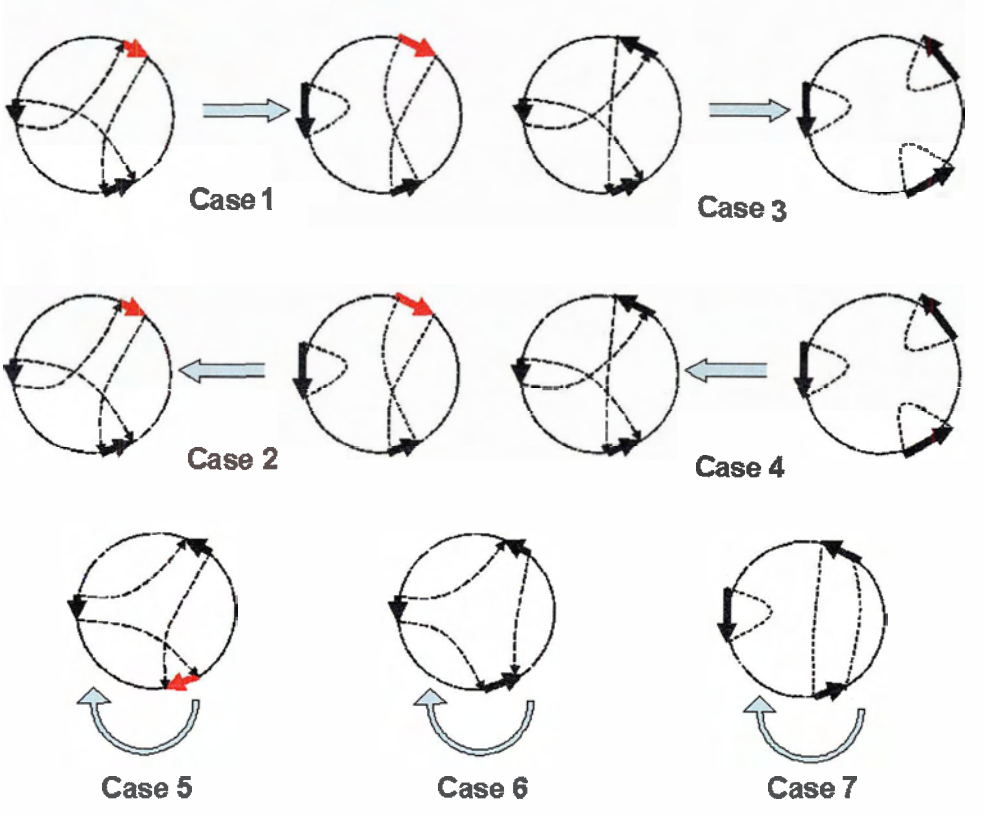

Figure 3.6: All possibilities of a transposition on a signed permutation. Only affected cycles are shown. A dashed line indicates a path formed by one or more desire and reality edges. Arrows represent reality edges and their directions when traverse the cycle.

Therefore, there are six types of events in an optimal sorting path which are summarized in Table 3.2. When taking into account the effect of each type of event on the changes of the number of breakpoints, denoted as $\Delta brs(\pi,e)$, each type of event may have more than one subtype of events. For example, the only allowed type of reversal in case 1 of Figure 3.4 has three subtypes because it may reduce the number of breakpoints by 0, 1 or 2. We list all possible subtypes of events for the six types of events and then we receive all 13 possible subtypes of events that may appear in an optimal sorting path.

Table 3.2: all possible types of events in an optimal sorting path

Event	Description of reality edges	$\Delta C(\pi,e)$	$\Delta brs(\pi,e)$
reversal	Two reality edges are convergent.	1	0
			1
			2
transposition	Three reality edges in the same cycle, one of them is divergent from the other two.	1	0
			1
transposition	Three reality edges in the same cycle, one of them is divergent from the other two.	0	0
transposition	Three reality edges are in the same cycle with same direction and are listed in increasing order in cycle bracket notation.	0	0
transposition	Three reality edges are in the same cycle with same direction, but are not listed in increasing order in cycle bracket notation.	2	0
			1
			2
			3
transposition	Three reality edges in two cycles. The two in a same cycle is convergent.	0	0
			1

3.2 Decision Version

In this part, we discuss the decision version of the problem which is defined as follows:

Input: A signed permutation π of length n and a positive integer k.

Output: Is $K_{min}(\pi) \leq k$?

By finding the lower and upper bounds of $K_{min}(\pi)$, we answer this question by dividing the problem into two cases based on whether k is bigger than the computed lower bound and smaller than the computed upper bound. If k is less than the lower bound or greater than the upper bound, the answer is no. The time complexity is $O(n^2)$, which is the time needed to compute the lower and upper bounds. If k is in the bounds range, we prove that the problem is at least as hard as sorting by transpositions only. We achieve this conclusion by analysing a signed permutation whose elements have all the same direction, namely $\rightarrow$.

3.2.1 Lower bounds

Claim 3.2: Let π be a simplified signed permutation of length n. Let e be a transposition or a reversal on π, and let $\pi' = \pi \circ e$. Then $0 \leq brs(\pi) - brs(\pi') \leq 3$.

Proof: Because π is a simplified permutation, no event on π can increase the number of breakpoints, i.e., $0 \leq brs(\pi) - brs(\pi')$.

If e is a transposition, we let $e = [a, i, j]$ with $1 \leq a < i < j \leq n$. From Definition 4 we know that e acts on three reality edges—(π_{a-1}, π_a), (π_{i-1}, π_i) and (π_{j-1}, π_j). After the

execution of e, we receive three new reality edges—(π_{a-1}, π_i), (π_{j-1}, π_a) and (π_{i-1}, π_j) in π'. If there are desire edges among the latter three pairs of vertices, i.e., some of the latter three reality edges do not correspond to any breakpoints, then e decreases the number of breakpoints. Therefore e can decrease by at most three breakpoints, i.e., $brs(\pi) - brs(\pi') \leq 3$. When e is a reversal, we can prove $brs(\pi) - brs(\pi') \leq 2$ in a similar way.

Theorem 3.4: Let π be a signed permutation of length n, Then $\frac{n+1}{3} \leq K_{min}(\pi)$.

Proof: From Claim 3.2, we know that an event can decrease the number of breakpoints by at most three. Further we know that permutation π has $n+1$ breakpoints. This yields a lower bound of $\frac{n+1}{3}$.

From Theorem 3.4, we obtain a trivial lower bound of the length of $E(\pi)$. However, not all permutations allow transpositions that reduce the number of breakpoints by 3, so the bound is not tight. We use the breakpoint graph $G(\pi)$ to obtain an improved lower bound. From Section 3.1.3 we know that an event can increase the number of cycles by at most two. And a sequence of events that sorts π must increase the number of cycles from $C(\pi)$ to $n+1$. These facts immediately lead to a better lower bound:

Theorem 3.5: Let π be a simplified signed permutation of length n, Then

$$\frac{n+1-C(\pi)}{2} \leq K_{min}(\pi).$$

We say that the lower bound in Theorem 3.5 is better that the one in Theorem 3.4 because it builds a relationship with the value of $C(\pi)$. Furthermore, we observe that in most of cases, the value of $C(\pi)$ is small and the lower bound is close to half of n, while the lower bound in Theorem 3.4 is static and close to one-third of n.

3.2.2 Upper bounds

Theorem 3.6: Let π be a signed permutation of length n. Then $K_{min}(\pi) \leq n$.

Proof: Setubal and Meidanis [SM97] proved that, as long as a permutation has elements with direction $\leftarrow$, there exists a reversal that reduces at least one breakpoint.

We state, in Claim 3.3, that there always exists a transposition that reduces at least one breakpoint when all elements of a permutation have direction $\rightarrow$.

Claim 3.3: When all elements of a permutation have direction $\rightarrow$, there exists a transposition that reduces at least one breakpoint.

Proof: Let us consider two elements $\vec{k}$ and $\overrightarrow{k-1}$ of a permutation. We show the case in which $\vec{k}$ precedes $\overrightarrow{k-1}$:

$$\ldots \cdot \vec{k} \cdot \ldots . \cdot \overrightarrow{k-1} \cdot \ldots$$

and the case in which $\overrightarrow{k-1}$ precedes $\vec{k}$:

$$\ldots \cdot \overrightarrow{k-1} \cdot \ldots \cdot \vec{k} \cdot \ldots .$$

In either case, each "·" represents a breakpoint. We can always move $\overrightarrow{k-1}$ to the left side of $\vec{k}$ which reduces at least one breakpoint.

Here we show that each event can reduce at least one breakpoint and so the upper bound is n+1. We next prove that the upper bound is n because the last event decreases by two breakpoints if it is a reversal or by three breakpoints if it is a transposition.

Theorem 3.7: Let π be a signed permutation of length n. Then $K_{min}(\pi) \leq n + 1 - C(\pi)$.

Proof: When traversing a cycle of $G(\pi)$, there are two possible directions of reality edges. From Theorem 3.3, we know that there always exists an event that increases the number of cycles by one whenever two reality edges are convergent.

When all reality edges are divergent in a cycle, the problem is transformed to the problem of sorting by transpositions only. Bafna and Pevzner [BP98] proved that sorting by transpositions only, the upper bound is $brs(\pi) - C(\pi)$.

Therefore, there always exists an event that increases the number of cycles by one. We proved that the upper bound is $n + 1 - C(\pi)$.

We know that, for sorting by transpositions only, the upper bound of an optimal sorting path is three quarters of the permutation length [BP98]. Meidanis *et al.* [MW02] hypothesize that sorting signed permutations by transpositions and reversals has the same upper bound. We *disprove* this by giving two counterexamples.

Conjecture [MW02]: Let π be a signed permutation of length n. Then $K_{min}(\pi) \leq \frac{3}{4}n$.

Counterexamples: Given permutation $(\overleftarrow{1}, \overleftarrow{2}, \overleftarrow{3}, \overleftarrow{4})$, we can easily determine that the length of its optimal sorting path is 4. But $\frac{3}{4}n = 3$. Based on the pattern of this permutation, we can construct more counterexamples. For example, the length of the optimal sorting path that sorts permutation $(\overleftarrow{5}, \overleftarrow{6}, \overleftarrow{7}, \overleftarrow{8}, \overleftarrow{1}, \overleftarrow{2}, \overleftarrow{3}, \overleftarrow{4})$ is 7, but $\frac{3}{4}n = 6$.

3.2.3 Permutation with same oriented blocks

Claim 3.4: Let π be a signed simplified permutation of length n. For every element π_i of π, let $sign(\pi) = \rightarrow$. Then there exists a shortest path that consists of transpositions only.

Proof: Suppose this claim is false. Then there exists a set U of counterexamples, $U = \{ (\pi, P) \mid \pi$ is a signed simplified permutation whose elements all have the same direction, and P is an optimal sorting path that sorts π and contains at least one reversal $\}$. From set

U, we pick up one element (π, P) as follows, π is a shortest permutation in U, and P has the shortest length for all the shortest permutations in U. This means that none of the permutations in U has fewer breakpoints than π and can be sorted with a path shorter than P. We let $P = e_1 e_2 \ldots e_k$. Then we prove that e_1 must be a reversal.

Suppose e_1 is not a reversal. Let $\pi' = \pi \cdot e_1$ and P' is an optimal sorting path sorting π'. There are two cases:

(1) π' is not in set U. Then P' may consist of transpositions only because π' is not a counterexample. We know $P = e_1 + P'$. Therefore P is a path with transpositions only as well, which conflicts with our assumption that P has at least one reversal.

(2) π' is in set U. Then P' has at least one reversal because π' is a counterexample. Furthermore, we know $|P| \leq |P'|$. We have $P = e_1 + P'$, i.e., $|P| = 1 + |P'|$. Then we have $1 + |P'| \leq |P'|$, which is impossible.

In either case, we arrive at a contradiction and then can prove that e_1 is a reversal.

Because all elements of π have the same direction from left to right, then all reality edges have the same direction from left to right. So e_1 must act on two divergent reality edges or two edges on two cycles. From Theorem 3.3, we know a reversal of an optimal sorting path can not act on two divergent reality edges or two edges on two cycles. Therefore, it is a contradiction, i.e., e_1 cannot be a reversal.

We prove that e_1 can be neither a reversal nor a transposition. That means P does not exist at all. Therefore such counterexamples do not exist. Finally, we prove our claim that permutations with one direction can be sorted with transpositions only.

3.2.4 Computational complexity

In the decision version, the computational complexity of the problem to answer whether $K_{min}(\pi)$ is smaller than a given integer K can be divided into two cases depending on whether $K_{min}(\pi)$ is in the range between the lower bound and upper bound.

We have to compute the lower and upper bounds before we can discuss the two cases. We know the time to compute the $C(\pi)$ is the dominant factor in determining the lower and upper bounds. To compute the value of $C(\pi)$, we need to construct the $G(\pi)$ which is $O(n^2)$ [BP96]. Therefore, we can compute lower and upper bounds in $O(n^2)$.

If K is out of the range between the lower and upper bound, we can answer the question whether $K_{min}(\pi)$ is smaller than K. If K is bigger than the upper bound, the answer is yes. If K is smaller than, or equal to, the lower bound, the answer is no.

Secondly, when K is in the range of bounds, the problem is far from easy to answer. However, from the analysis of Section 3.2.3, we know that only transpositions are needed to sort a permutation whose elements have the sign $\rightarrow$. We also know that transpositions have no effect on the direction of each element, and then the problem of sorting a one-directional permutation is equivalent to the problem of sorting unsigned permutation by transpositions. Because the complexity of sorting by transpositions only is still unknown, the complexity of sorting a one-directional permutation is unknown. A one-directional permutation is a specific example of a signed permutation and the problem of sorting a one-directional permutation is a part of the problem of sorting by transpositions and reversals. Therefore we have Theorem 3.8:

Theorem 3.8: The complexity of the problem of sorting by transpositions and reversals is at least as hard as the problem of sorting by transpositions only.

3.3 Optimization Version

In this section, we discuss the optimization version of the problem which is defined as follows:

Input: A signed permutation π of length n.

Output: What is an $E(\pi)$?

As we know, except for the problem of sorting a signed permutation by reversals only, there are only approximation and heuristic algorithms known for all variants of the genome rearrangement problem. In Section 3.3.1, we discuss three approximation algorithms which are straightforwardly derived from the analysis of lower bounds and upper bounds of $K_{min}(\pi)$. However, we are interested in finding an algorithm that gives an exact optimal sorting path or its length. We know that the algorithm runs in time $O(n^n)$ according to the computational complexity analysis in Section 3.2. Our main objective is to improve this strategy to make the algorithm run in a more reasonable time useful for real-world applications.

From Section 3.3.2 to 3.3.5, we present four techniques which may be used to simplify a permutation and thus may be able to reduce the problem instances for sorting permutations. We denote *core permutations* as permutations to which we cannot apply these techniques. Our strategy is that, by using these techniques to simplify a non-core permutation, the permutation either is sorted or is reduced to a core permutation, i.e., using these techniques, the problem of sorting all permutations is transformed to the problem of sorting core permutations. All of these techniques are relatively computationally inexpensive and easy to perform with running times that do not exceed $O(n^2)$. Therefore, these techniques are valuable because they are helpful in sorting non-

core permutations which consists of the most part of our original problem. This is an application of kernelization—solving the easy cases first.

However, core permutations are the most difficult ones to be sorted and we do not have any better way than exhaustive search to sort them.

3.3.1 Approximation algorithms

Theorem 3.4 and Theorem 3.6 imply the following 3-approximation algorithm.

3-approximation algorithm:

Input: a signed simplified permutation π.
Output: a sequence f events sorting π.

```
While( |π| ≠ 0) Do
    find the element 1;
    If the direction of 1 is → do
        move element 1 to the left head;
    Else
        reverse the left segment ending at element 1;
    Simplify the permutation π;
End
```

Figure 3.7 The 3-approximation algorithm

We only need to make sure that every event applied decreases the number of breakpoint by at least one. It is a 3-approximation algorithm because, in the ideal situation, an event can decrease the number of breakpoint by three and this algorithm can only guarantee one breakpoint per event. We start from the smallest element 1, if the direction of 1 is $\rightarrow$, we use a transposition and move the single element to the left most end. If the direction of 1 is $\leftarrow$, we use a reversal move it to the left side by reversing the segment starting from beginning and ending with the element. We then simplify the yielded permutation

and repeat these steps until the identity is obtained. The times to find and to execute an event both are $O(n)$ and there are n events in worst-case. Therefore, the 3-approximation algorithm runs in $O(n^2)$.

Theorem 3.5 and Theorem 3.7 imply a 2-approximation algorithm. It is a 2-approximation algorithm because, in the ideal situation, an event can increase the number of cycles by two and this algorithm can only guarantee one cycle per event.

2-approximation algorithm:

Input: a signed simplified permutation π.
Output: a sequence f events sorting π.

```
Begin
   Construct G(π);
   While ( there is two convergent reality edges in a cycle) Do
         a reversal on these two reality edges is executed;
         simplify the permutation π;
         Construct G(π);
   TransSort(π);
End
```

Figure 3.8 The 2-approximation algorithm

First, we construct the breakpoint graph. Second, in each cycle, we pick two convergent reality edges and use a reversal until there are no convergent edges in a cycle. Then we adopt the algorithm TransSort(π) [BP98]. The time to construct the breakpoint graph is $O(n^2)$. The time to pick two reality edges and execute the reversal is $O(n)$ and there are n events in worst-case. The running time of TransSort(π) is $O(n^4)$. Therefore the 2-approximation algorithm runs in $O(n^4)$.

As we state in Section 3.2.2, three quarters of the permutation length is not the upper bound. Furthermore, we are not satisfied with the upper bound of $n+1-C(\pi)$. We

have a conjecture that $K_{min}(\pi) \leq \frac{3}{4}(n+1)+\frac{1}{4}C(\pi)$. This yields an approximation algorithm STR(π) that finds a path with at most $\frac{3}{4}(n+1)+\frac{1}{4}C(\pi)$ events, but the approximation of the path is uncertain for a given case. We believe STR(π) is better because the number of events is close to three quarters of the permutation length since $C(\pi)$ usually is small.

Algorithm STR(π):

Input: a signed simplified permutation π.
Output: a sequence f events sorting π.

```
1  Begin
2      Construct G(π);
3      While (G(π) contains a fully oriented cycle C) DO
4              a 2-transposition on C is executed;
5      If (there is two convergent reality edges in a cycle) DO
6              a reversal on these two reality edges is executed;
7              go to 2;
8      TransSort(π);
9  End
```

Figure 3.9 The algorithm STR(π)

We keep applying all 2-move transpositions first. Then for cycles with convergent reality edges, we use reversals to make all reality edges divergent or split them into different cycles. During this process, we always apply 2-move transpositions as soon as they are available. Finally, we have a breakpoint graph that consists of open cycles only. Then the problem is sorting by transpositions (SBT) only and we adopt the algorithm TransSort(π) [BP98]. The time to construct the breakpoint graph and to find a 2-move transposition both are $O(n^2)$. The time to find and execute a reversal that transforms the problem to SBT is $O(n)$. After the execution of a reversal, we go back to check whether a 2-move transposition is applicable. We do that at most n times. The running time of TransSort(π) is $O(n^4)$. Therefore this approximation algorithm runs in $O(n^4)$.

3.3.2 Input reduction by simplified permutation

This simple technique is based on the fact that the problem of sorting a given permutation is equivalent to the problem of sorting its simplified permutation. This reduces the length of a permutation from n to $brs(\pi)$-1. This rule is applied repeatedly after each event decreases the number of breakpoints when sorting π.

This technique is valuable. What we are interested in is an algorithm that finds the shortest path which sorts π. The algorithm inevitably runs at exponential time $O(n^n)$ which is horrible. However, any small reduction, even if it is tiny, in n can improve the running time significantly. Therefore, any effort to reduce n pays off handsomely.

This technique is applicable to a large range. For a given signed permutation with length n, there are $2^n n!$ possible permutations. Using combinatorics, we know that $(2^{n-1}n! - n! + 1)$ out of $2^n n!$ can be simplified using this technique. For example, for $n = 4$, there are 169 of 384 permutations that can be simplified using this technique at least once.

We would like to point out that this technique is of significant importance for biologists in their real life research. Molecular biologists find that the segment involved in global rearrangement usually consists of multiple genes and in most cases this segment is operated on as a unit. For example, Hannenhalli *et al.* [HC95] use this technique in their efforts to describe the evolution of the herpes virus. They located 30 conserved genes by comparing amino acid sequences encoded by ten herpes viruses. They also find these 30 genes comprise of seven conserved blocks that are rearranged in the genomes of different herpes viruses.

3.3.3 Input reduction by component permutation

From Theorem 3.2, we know that there exists an optimal sorting path that each event acts on a component of the breakpoint graph. We therefore can treat the components as separate permutations by identifying them with one of the shortest permutations whose breakpoint graphs consist of this component only. Therefore, the problem of sorting the permutation π can be divided into several small problems of sorting each component of $G(\pi)$.

It is hard to know whether this technique is applicable to a permutation before the construction of its breakpoint graph. It is also hard to count the number of permutations (among $2^n n!$) that can be simplified by using this technique. However, we can still see the value of this technique. Because, for an algorithm with an $O(n^n)$ running time, any reduction in n can improve the running time significantly.

3.3.4 Input reduction by high degree

From Table 3.2, taking into account both $\Delta C(\pi, e)$ and $\Delta brs(\pi, e)$, we get all the 13 possible types of events that may appear in an optimal sorting path. We know that if we can order the appearance of these types of events in an optimal sorting path, we can achieve a polynomial time algorithm. Although to order all of them in a row is far from easy and obviously now, we try to give an order to some of them.

We know that a transposition in Case 3 of Figure 3.6 is a 2-move. The three reality edges divide the circle into the three arcs X, Y and Z, which are depicted with different colors in Figure 3.10. The transposition decreases the number of breakpoints

from 3 to 0 depending on how many of the three paths connect the three reality edges directly. If each of the three paths connects two reality edges directly, then the transposition decreases three breakpoints as depicted in Figure 3.10a). If none of them connects two reality edges directly, then the transposition does not decrease any breakpoints as shown in Figure 3.10d). Figure 3.10b) and 3.10c) depict the other two situations that decrease the number of breakpoints by two and one, respectively.

The four cases in Figure 3.10 are all possibilities of a 2-move transposition. We can divide each case into more sub-situations when considering whether there are reality edges on arcs X, Y and Z. We believe, under some of these sub-situations, a 2-move transposition can be done before others. We describe our observation in two steps:

Step 1: Under three situations, a transposition has the highest priority that we can always implement it first if:

1) In Figure 3.10a), there is no reality edge on at least one of three arcs X, Y and Z;

2) In Figure 3.10b), there is no reality edge on arc Y; and

3) In Figure 3.10c), there is no reality edge on arc X and Y.

We designate a transposition under these three situations as a first priority transposition. Whenever we find a transposition under one of these situations, we can apply them directly because the implementation of them does not affect the shape of other cycles.

Step 2: Under the following two situations, a transposition has second priority if:

1) Other situations in Figure 3.10a), the three arcs X, Y and Z all have at least one reality edge.

2) Other situations in Figure 3.10b), there are reality edges on arc Y and we do not care whether there are reality edges on arc X and/or Z.

We designate a transposition under these two situations as a second priority transposition. However, if there is more than one transposition that falls into the category of the second priority transposition, the order to implement them is important. We cannot implement a second transposition whenever we find it and we need to know whether there are more second priority transpositions. If there are more, we need to do an exhaustive search for all of them to find which one need be done first.

. However, we have no knowledge about a 2-move transposition under other situations. So far, we believe that they have the third priority as all other events—1-move transpositions, 1-move reversals and 0-move transpositions. After we have finished all first and second priority transpositions, we have to use exhaustive search to decide which third priority transposition has to done first. We know during this process, the implementation of a third priority transposition may result in the generation of a first or second priority transposition. Then we can apply Step1 or Step 2 again, i.e., Step1 and Step 2 always have higher implementation priority and can be applied repeatedly.

This technique to reduce the input of our problem is valuable. Although 2-move transpositions in Steps 1 and 2 only represent a small ratio of all possible 2-move transpositions, their chances in an optimal sorting path is bigger than this ratio. A transposition in Steps 1 and 2 can increase the number of cycles by 2 and decrease the number of breakpoints by 2 or 3. That is why an optimal sorting path of sorting by transpositions and reversals is shorter than that of sorting by transpositions or reversals alone.

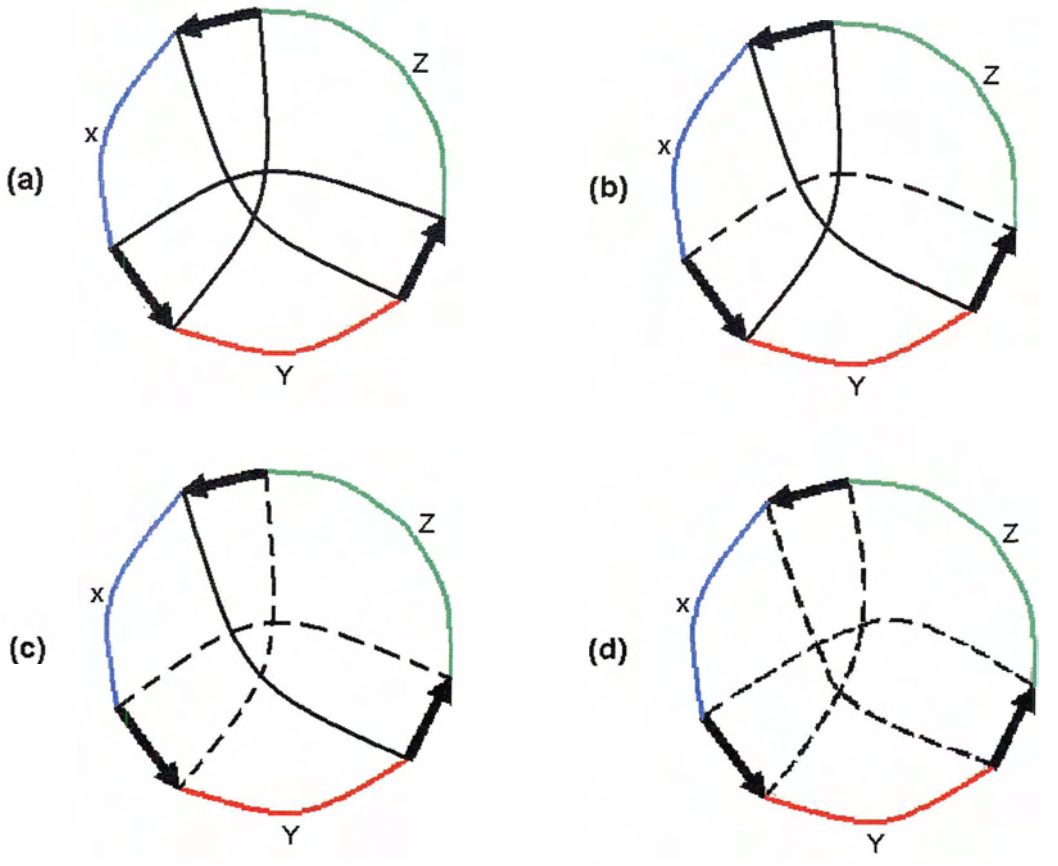

Figure 3.10: All possibilities of a 2-move transposition. A dashed line indicates a path that contains one or more reality edges. A real black line indicates a desire edge. a): $\Delta brs(\pi, p) = 3$; b): $\Delta brs(\pi, p) = 2$; c): $\Delta brs(\pi, p) = 1$; d): $\Delta brs(\pi, p) = 0$.

3.3.5 Input reduction by sorting from different directions

Here, let's go back to our original problem of sorting permutation π to another permutation σ rather than the problem sorting π to the identity. We know that, as depicted in Figure 3.8, if $e_1e_2...e_{k-1}e_k$ is an optimal sorting path sorting π to σ, $e_ke_{k-1}...e_2e_1$ is an optimal sorting path sorting σ to π. From Section 3.3.3, we know we can only partially order some events in an optimal sorting path. We believe the more we can do Step 1 and Step 2 described in the above section, the faster we can get an optimal sorting

path. To find this optimal sorting path, so far in our work, we only work from π to σ by treating σ as the identity permutation and changing π correspondingly. Our strategy is to apply this partial ordering on the other side as well. We try to apply Step1 and Step 2 on one side (sorting from π to σ) repeatedly until there are no more Step1 and Step 2. Then we may be able to apply Step1 and Step 2 on the other side (sorting σ to π). It will help us improve the time to find an optimal sorting path because Step 1 and Step 2 is less time consuming. Furthermore, when finished the Step 1 and Step 2 on the side from σ to π, we may be able to apply Step 1 and Step 2 on the side from π to σ again. That means the process of applying Step 1 and Step 2 may be able to applied recursively and alternatively between two ends.

Figure 3.8: An optimal sorting path sorting π to σ is the reverse of an optimal sorting path sorting σ to π

Chapter 4

Conclusions

4.1 Contributions

Large scale comparative genetic mapping offers exciting prospects for understanding genomic evolution; computational approaches provide efficient tools to attack these problems. This thesis presents the first Steps for computing the distance of genomes in the sense of the global mutations (reversals and transpositions). While the classical complexity of the problem sorting by transpositions is an open problem (NP-completeness is conjectured), the computational complexity of finding the transposition and reversal distance is shown to be at least as hard as the problem sorting by transpositions. We derived lower and upper bounds for the length of an optimal sorting path of events. The upper bound of sorting by transpositions can not be extended to our problem of sorting by transpositions and reversals.

There are no known polynomial algorithms for sorting by transpositions and reversals, and thus an exhaustive search algorithm is necessary to find the shortest path. Because the running time of an exhaustive search algorithm is extremely slow for long permutations, we developed four techniques to improve the running time of the exhaustive algorithm. These techniques have been implemented in our program. Our program is more efficient in running time. For example, for a randomly generated input permutation of length 10, our program is able to provide the result in milliseconds compared to days that are required by the "pure" exhaustive search algorithm without using these techniques. When the input data grows to $2^{20}20!$ permutations of length 20, our program computes the results in about 7 days. There is no measured running time of the same input data by a pure exhaustive search algorithm.

4.2 Future Work

4.2.1 Experimental study

We believe that techniques introduced in Section 3.3 can improve the running time of our program to find the exact optimal sorting path. Although we can find an optimal sorting path for a randomly given permutation of length 30 in seconds, we do not have large experimental studies to validate the achievement of the implementation of these techniques. It would be useful to evaluate how efficient these techniques are.

Furthermore, we should test our program with biological data, since we are developing an algorithm for future use in Biology. If our program can be used by

biologists in their work, we can get valuable feedback which will allow for more realistic implementation.

4.2.2 Further improvement of the running time

We conclude that the techniques used to improve the running time in this thesis are only applicable for the permutations with certain patterns. Unfortunately, these techniques are not applicable to the problem of sorting core permutations. We still need an exhaustive search for sorting core permutations. Furthermore, our program can only guarantee that sorting permutations of length 20 is fast. Obviously, it is too limited for realistic biological data and urges us to find more techniques for further running time speed-up. For example, any ordering among possible events will largely improve the running time because the computational complexity comes from the fact that we do not know which events should be chosen first. Furthermore, during the exhaustive search, the earlier we can terminate a search path, the shorter the running time will be. Although we do not discussed in this thesis, we did find some patterns that help us to terminate a path and we believe this is another promising way for further improvement.

4.2.3 Combine with other operations

In this thesis we consider reversals and transpositions as global rearrangements of genomes. However, there are more kinds of global mutations which make sense from a biological viewpoint. To model genome rearrangement adequately, we would like to combine all global mutations in our analysis. For example, another important global

rearrangement is duplication of segments of genomes. Duplication has a major role in genome evolution and has been observed in all organisms. Further research including reversals, transpositions and duplications will reflect the real life evolutionary history more precisely. The ideal algorithm is to handle all possible global mutations in the future.

Furthermore, our analysis does not include local mutations. We know that local mutations are another key factor in genome evolutions, especially in the evolution of closely related genomes. Therefore, the algorithms which can handle global and local mutations simultaneously could be used to analyze practical data.

4.2.4 Conjectures

We conclude this thesis with some conjectures and open problems.

Conjecture 1: For sorting signed permutation π, there is an optimal sorting path that contains no negative-move events.

Negative-move events refers to (-2)-move transpositions, (-1)-move transpositions and (-1)-move reversals. We make this conjecture because applying such events would appear to be a negative Step towards sorting π. However we have no insight as to how to prove this conjecture.

Conjecture 2: Let π be a signed permutation of length n, Then

$$K_{min}(\pi) \leq \frac{3}{4}(n+1)+\frac{1}{4}C(\pi)$$

From [BP98], we know, if all reality edges of $G(\pi)$ have the same direction $\rightarrow$, $K_{min}(\pi) \leq \frac{3}{4}(n+1-C(\pi))$. We observe that transforming a cycle that has convergent reality edges into cycle(s) with divergent reality edges only needs at most four reversals. Note that, from Theorem 3.3, each reversal increases the number of cycles by 1. Therefore $K_{min}(\pi) \leq \frac{3}{4}(n+1- (C(\pi) + 4C(\pi))) + 4C(\pi) = \frac{3}{4}(n+1)+\frac{1}{4}C(\pi)$. We need to further prove our observation.

References

[Aya97] F. J. Ayala. Vagaries of the Molecular Clock. *Proceedings of the National Academy of Science USA 94*, pp. 7776-7783, 1997.

[Behe90] M. J. Behe. Histone deletion mutants challenge the molecular clock hypothesis. *Trends in Biochemical Science* 15: 374-376, 1990.

[BH01] P. Berman, S. Hannenhalli, and M. Karpinki. 1.375-Approximation Algorithm for Sorting by Reversals. *Electronic Colloquium for Computational Complexity TR01-047*, 2001.

[BH96] P. Berman and S. Hannenhalli. Fast sorting by reversal. In *Proceedings of the Seventh Annual Symposium on Combinatorial Pattern Matching*, pp.168–185, 1996.

[BK96] M. Blanchette, T. Kunisawa, D. Sankoff. Parametric genome rearrangement. *Gene* 172, GC 11–17, 1996.

[BK99] *P.* Berman and M. Karpinski. On some tighter inapproximability results. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, LNCS 1644, 200–209, 1999.

[BM01] D. A. Bader, B. M. Moret, and M.Yan. A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study. *Journal of Computational Biology.* 8(5), pp. 483–491, 2001.

[BP95] V. Bafna and P. A. Pevzner. Sorting by reversals: genome rearrangements in plant and evolutionary history of X chromosome. *Molecular Biology Evolution* 12, pp. 239–246, 1995.

[BP96] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2), pp. 272–289, 1996.

[BP98] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2), pp. 224–240, 1998.

[Bri86] R. J. Britten. Rates of DNA sequence evolution differ between taxonomic groups. *Science* 231, pp. 1993–1398, 1986.

[Cap99] A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Mathematics* 12(1), pp. 91–110. 1999.

[Cap97] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology*, pp. 75–83, ACM Press, New York, 1997.

[Chr98] D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the Ninth Annual Symposium on Discrete Algorithms*, pp. 244–252. ACM Press, New York, 1998.

[Chr99] D. A. Christie. *Genome Rearrangement Problems*. PhD Thesis, University of Glasgow, 1999.

[CI01] David A. Christie, Robert W. Irving. Sorting Strings by Reversals and by Transpositions. *SIAM Journal on Discrete Mathematics*, 14(2), pp. 193–206, 2001.

[CLN99] A. Caprara, G. Lancia, and S. K. Ng. A column-generation based branch-and-bound algorithm for sorting by reversals. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 47, pp. 213–226, 1999.

[CRC] Algorithms and Theory of Computation Handbook, *CRC Press LLC*, pp. 32-39 and 34-17, 1999

[Cri70] F. Crick. Central Dogma of Molecular Biology. *Nature* 227, pp. 561–563, 1970.

[ED01] N. Eriksen, D. Dalevi, S. G. E. Andersson, and K. Eriksson. Gene order rearrangements with derange: weights and reliability. Submitted to *the Journal of Computational Biology*, 2001.

[EE01] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1–3), pp. 289–300, 2001.

[Eri02] N. Eriksen. (1+epsilon)-approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1), pp. 517–529, 2002.

[Eri03] N. Eriksen. *Combinatorial methods in comparative genomics.* PhD Thesis, Royal Institute of Technology, 2003.

[Fit94] W. M. Fitch. Molecular Clocks Are Better Than You Think. *Journal of General Physiology* 102(6), pp. A1-A1, 1994.

[GP99] Q.P. Gu, S. Peng, and H. Sudborough, A 2-approximation algorithm for genome rearrangements by reversals and transpositions, *Theoretical Computer Science*, 210(2), pp. 327–339, 1999.

[Han96] S. Hannenhalli. Polynomial algorithm for computing translocation distance between genomes. *Proceedings of CPM 1996*, pp. 168–185, 1996.

[Har03] T. Hartman. A simpler 1.5-approximation algorithm for sorting by transpositions. *Proceedings of CPM 2003*, pp. 156–169, 2003.

[HC95] S. Hannenhalli, C. Chappey, E. V. Koonin, and P. A. Pevzner. Genome sequence comparison and scenarios for gene rearrangements: a test case. *Genomics* 30, pp. 299–311, 1995.

[HP95] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithmic for genomic distance problem). In *Proceedings of 36th Annual IEEE Symposium on Foundations of Computer Science*, pp. 581–592, 1995.

[HP99] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals, Journal of the ACM (JACM), 46(1):1–27, 1999. First appeared in *Proceedings of the 27th Annual Symposium on Theory of Computing.* pp. 178–189, ACM Press, New York, 1995.

[KR95] J. D. Kececioglu and R. Ravi. Of mice and men: Evolutionary distances between genomes under translocations, *The Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 604–613, 1995.

[KS93] J. D. Kececioglu, D. Sankoff. Exact and approximation algorithms for the inversion distance between two chromosomes. *Proceedings of CPM 1993*, pp. 87–105. 1993.

[KS95] J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2). pp. 180–210, 1995.

[KS97] H. Kaplan, R. Shamir, and R. E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal Computing* 29(3). pp. 880–892, 1999. First appeared in *Proceedings of the Eighth Annual Symposium on Discrete Algorithms*. pp. 344–351, ACM Press, New York.1997.

[LX01] G. H. Lin and G. Xue. Signed genome rearrangements by reversals and transpositions: Models and approximations. *Theoretical Computer Science* 259. pp. 513–531, 2001.

[MW00] J.Meidanis, M. T. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. Manuscript, 2000.

[MW02] J.Meidanis, M. T. Walter, and Z. Dias. A lower bound on the reversal and transposition diameter. *Journal of Computational Biology.* 9(5). pp. 743–745, 2002

[OL92] C. O'h Uigin and W. H. Li. The molecular clock ticks regularly in muroid rodents and hamsters. *Journal of Molecular Biology and Evolution* 35, pp. 377–384, 1992.

[PH88] J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evol*ution 27. pp. 87–97, 1988.

[PS00] I. Pe'er and R. Shamir. Approximation Algorithms for the Median Problem in the Breakpoint Model, in Comparative Genomics, D. Sankoff and J.H. Nadeau, eds., Kluwer, Dordrecht, The Netherlands, 2000, pp. 225–241

[SC90] D. Sankoff, R. Cedergren, and Y. Abel. Genomic divergence through gene rearrangement. *Methods in Enzymology*, Volume 183, 1990

[SE00] D. Sankoff and N. El-Mabrouk. Duplication, rearrangement, and reconciliation. *Comparative Genomics*, pp. 537-550. Kluwer Academic Publishers, 2000.

[SM97] J. C. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*, International Thomson Publishing Inc, 1997.

[SN77] F. Sanger, S. Nicklen, and A.R. Coulson, DNA sequencing with chain terminating inhibitors, *Proceeding of National Academic Science. USA*, Vol. 74, pp. 5643–5667, 1977.

[TC00] E. R. Tillier and R. A. Collins. Genome rearrangement by replication-directed translocation. *Nature Genetics*, 26(2). pp. 134–135, 2000.

[Tho82] J. P. Thorp. The Molecular Clock hypothesis: Biochemical Evolution, Genetic Differentiation and Systematics. *Ann. Rev. Ecol. Syste.* 13, pp. 139-68, 1982.

[WC53] JD. Watson and FHC. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature* 171. pp. 737–38, 1953.

[WD98] M. E. Walter, Z. Dias, and J. Meidanis. Reversal and Transposition Distance of Linear Chromosomes, *Proceedings of SPIRE'98 – String Processing and Information Retrieval: A South American Symposium*, pp. 96–102, 1998.

[WD99] M. E. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes, *Proceedings of SPIRE'98 – String Processing and Information Retrieval: A South American Symposium*. pp. 96–102, 1999.

[WE82] G. A. Watterson, W. J. Ewens, T. E. Hall and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(31). pp. 1–7, 1982.

[ZP65] E. Zuckerkand and L. Pauling. Molecules as Documents of Evolutionary History. *Journal of Theoretic Biology* 8, pp. 357-66, 1965.