

Gustavo Rodrigues Galvão

**Algorithms for Sorting by Reversals or  
Transpositions, with Application to Genome  
Rearrangement**

*Algoritmos para Problemas de Ordenação por  
Reversões ou Transposições, com Aplicações em  
Rearranjo de Genomas*

CAMPINAS

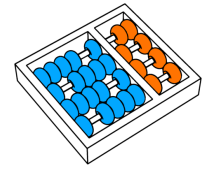
2015





University of Campinas  
Institute of Computing

*Universidade Estadual de Campinas*  
*Instituto de Computação*



Gustavo Rodrigues Galvão

**Algorithms for Sorting by Reversals or  
Transpositions, with Application to Genome  
Rearrangement**

Supervisor: Prof. Dr. Zanoni Dias  
*Orientador(a):*

***Algoritmos para Problemas de Ordenação por  
Reversões ou Transposições, com Aplicações em  
Rearranjo de Genomas***

PhD Thesis presented to the Graduate Program of the Institute of Computing of the University of Campinas to obtain a PhD degree in Computer Science.

*Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.*

THIS VOLUME CORRESPONDS TO THE FINAL VERSION OF THE THESIS DEFENDED BY GUSTAVO RODRIGUES GALVÃO, UNDER THE SUPERVISION OF PROF. DR. ZANONI DIAS.

*ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA POR GUSTAVO RODRIGUES GALVÃO, SOB ORIENTAÇÃO DE PROF. DR. ZANONI DIAS.*

---

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS  
2015

Agência de fomento: FAPESP  
Nº processo: 2014/04718-6

Ficha catalográfica  
Universidade Estadual de Campinas  
Biblioteca do Instituto de Matemática, Estatística e Computação Científica  
Maria Fabiana Bezerra Muller - CRB 8/6162

G139a Galvão, Gustavo Rodrigues, 1988-  
Algorithms for sorting by reversals or transpositions, with application to genome rearrangement / Gustavo Rodrigues Galvão. – Campinas, SP : [s.n.], 2015.

Orientador: Zanoni Dias.  
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Biologia computacional. 2. Algoritmos aproximados. 3. Filogenia - Matemática. I. Dias, Zanoni, 1975-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

#### Informações para Biblioteca Digital

**Título em outro idioma:** Algoritmos para problemas de ordenação por reversões ou transposições, com aplicações em rearranjo de genomas

**Palavras-chave em inglês:**

Computational biology

Approximation algorithms

Phylogeny - Mathematics

**Área de concentração:** Ciência da Computação

**Títuloção:** Doutor em Ciência da Computação

**Banca examinadora:**

Zanoni Dias [Orientador]

Maria Emília Machado Telles Walter

Yoshiko Wakabayashi

Eduardo Candido Xavier

Guilherme Pimentel Telles

**Data de defesa:** 02-10-2015

**Programa de Pós-Graduação:** Ciência da Computação

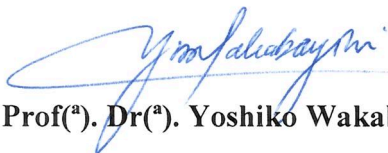
## TERMO DE APROVAÇÃO

Defesa de Tese de Doutorado em Ciência da Computação, apresentada pelo(a)  
Doutorando(a) **Gustavo Rodrigues Galvão**, aprovado(a) em **02 de outubro de 2015**, pela  
Banca examinadora composta pelos Professores Doutores:



**Prof<sup>(a)</sup>. Dr<sup>(a)</sup>. Maria Emília Machado Telles Walter**

**Titular**



**Prof<sup>(a)</sup>. Dr<sup>(a)</sup>. Yoshiko Wakabayashi**

**Titular**



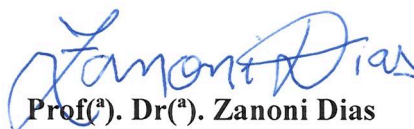
**Prof<sup>(a)</sup>. Dr<sup>(a)</sup>. Eduardo Candido Xavier**

**Titular**



**Prof<sup>(a)</sup>. Dr<sup>(a)</sup>. Guilherme Pimentel Telles**

**Titular**



**Prof<sup>(a)</sup>. Dr<sup>(a)</sup>. Zanoni Dias**

**Presidente(a)**



# Algorithms for Sorting by Reversals or Transpositions, with Application to Genome Rearrangement

Gustavo Rodrigues Galvão<sup>1</sup>

October 02, 2015

**Examiner Board/*Banca Examinadora*:**

- Prof. Dr. Zaroni Dias (Supervisor/*Orientador*)
- Prof. Dr. Maria Emília Machado Telles Walter  
Institute of Exact Sciences - UnB
- Prof. Dr. Yoshiko Wakabayashi  
Institute of Mathematics and Statistics - USP
- Prof. Dr. Eduardo Candido Xavier  
Institute of Computing - UNICAMP
- Prof. Dr. Guilherme Pimentel Telles  
Institute of Computing - UNICAMP
- Prof. Dr. Cristina Gomes Fernandes  
Institute of Mathematics and Statistics - USP (Substitute/*Suplente*)
- Prof. Dr. Cid Carvalho de Souza  
Institute of Computing - UNICAMP (Substitute/*Suplente*)
- Prof. Dr. Flávio Keidi Miyazawa  
Institute of Computing - UNICAMP (Substitute/*Suplente*)

---

<sup>1</sup>Financial support: CAPES scholarship (process 01-P-01965-2012) 08/2012–07/2013, CNPq scholarship (process 142260/2014-2) 07/2014–08/2014, and FAPESP scholarship (process 2014/04718-6) 09/2014–09/2015.





# Abstract

During evolution, rearrangement events may alter the order and the orientation of the genes in a genome. The problem of finding the minimum sequence of rearrangements that transforms one genome into another is a well-studied problem that finds application in comparative genomics. Representing genomes as permutations, in which genes appear as elements, that problem can be reduced to the combinatorial problem of sorting a permutation using a minimum number of rearrangements. Such combinatorial problem, referred to as rearrangement sorting problem, varies according to the types of rearrangements considered.

In this thesis, we focus on two types of rearrangements: reversals and transpositions. Many variants of the rearrangement sorting problem involving these rearrangements have been tackled in the literature and, for most of them, the best known algorithms are approximations or heuristics. For this reason, we present a tool, called GRAAu, to aid in the evaluation of the results produced by these algorithms. In addition, we present a general heuristic that can be used to improve the solutions provided by any non-optimal algorithm.

Besides presenting GRAAu and the improvement heuristic, which have a more general appeal, we present contributions regarding specific variants of the rearrangement sorting problem. First, we consider the problem of sorting by transpositions and we present experimental and theoretical results regarding three approximation algorithms based on alternative approaches to the cycle graph, which is a standard tool for attacking the rearrangement sorting problem. Then, we turn our attention to variants involving short rearrangements. More precisely, we study five variants: (i) the problem of sorting a signed linear permutation by super short reversals, (ii) the problem of sorting a signed circular permutation by super short reversals, (iii) the problem of sorting a signed linear permutation by short reversals, (iv) the problem of sorting a signed linear permutation by super short rearrangements, and (v) the problem of sorting a signed linear permutation by short rearrangements. We present polynomial-time algorithms for problems (i), (ii) and (iv), a 5-approximation algorithm for problem (iii), and a 3-approximation algorithm for problem (v). We use the algorithm developed for problem (ii) to reconstruct the phylogeny of *Yersinia* genomes and compare the result with the phylogenies presented in previous works.



# Resumo

Ao longo da evolução, eventos de rearranjo podem alterar a ordem e a orientação dos genes de um genoma. O problema de calcular a menor sequência de rearranjos que transforma um genoma em outro é um problema bastante estudado que encontra aplicações em genômica comparativa. Representando genomas como permutações, nas quais os genes aparecem como elementos, esse problema pode ser reduzido ao problema combinatório de ordenar uma permutação utilizando o menor número de rearranjos. Tal problema combinatório, referido como problema da ordenação por rearranjo, varia de acordo com os tipos de rearranjo considerados.

Nesta tese, focamos nosso estudo em dois tipos de rearranjo: reversões e transposições. Muitas variações do problema da ordenação por rearranjo que envolvem esses rearranjos têm sido atacadas na literatura e, para a maior parte delas, os melhores algoritmos conhecidos são aproximações ou heurísticas. Em razão disso, apresentamos uma ferramenta, chamada GRAAu, que auxilia a avaliação dos resultados produzidos por esses algoritmos. Além disso, apresentamos uma heurística genérica que pode ser usada para melhorar as soluções produzidas por qualquer algoritmo não exato.

Além de apresentar o GRAAu e a heurística de melhoria, os quais possuem um apelo mais geral, apresentamos contribuições relacionadas às variações específicas do problema da ordenação por rearranjo. Primeiro, consideramos o problema da ordenação por transposições e apresentamos resultados teóricos e práticos relacionados a três algoritmos aproximados baseados em uma abordagem alternativa ao grafo de ciclos, que é uma ferramenta padrão para atacar o problema da ordenação por rearranjo. Depois, voltamos nossa atenção às variações que envolvem rearranjos curtos. Mais precisamente, estudamos cinco variações: (i) o problema de ordenar uma permutação linear com sinal por reversões super curtas, (ii) o problema de ordenar uma permutação circular com sinal por reversões super curtas, (iii) o problema de ordenar uma permutação linear com sinal por reversões curtas, (iv) o problema de ordenar uma permutação linear com sinal por rearranjos super curtos e (v) o problema de ordenar uma permutação linear com sinal por rearranjos curtos. Apresentamos algoritmos polinomiais para os problemas (i), (ii) and (iv), um algoritmo 5-aproximado para o problema (iii) e um algoritmo 3-aproximado para o problema (v). Usamos o algoritmo desenvolvido para o problema (ii) para reconstruir a filogenia de genomas de bactérias do gênero *Yersinia* e comparamos os resultados com as filogenias apresentadas em trabalhos anteriores.



# Agradecimentos

Gostaria de agradecer aos meus pais, Fátima e Claudio, por sempre terem me apoiado.

À minha companheira, Ticiane, e à minha filha, Sofia, tanto pelo apoio e carinho, quanto pela paciência durante todo o tempo em que estive na Pós-Graduação. Foram inúmeras as vezes em que elas compreenderam que não poderia estar com elas devido ao meu *shigoto*.

Ao meu orientador, Prof. Zanoni Dias, pela oportunidade e confiança em mim depositada, pela orientação e colaboração durante o desenvolvimento desta tese e, acima de tudo, por sempre estar disposto a me ajudar. Eu tenho dúvidas se teria chegado até aqui caso não tivesse o orientador que tive. Muito obrigado, Zanoni.

Também gostaria de agradecer aos meus colegas Ulisses, Carla e Christian, assim como ao Prof. Orlando Lee, por terem colaborado com minha pesquisa.

Por fim, agradeço às agências de fomento CAPES (processo 01-P-01965-2012), CNPq (processo 142260/2014-2) e FAPESP (processo 2014/04718-6) pelo apoio financeiro.



# Contents

<b>Abstract</b>	<b>ix</b>
<b>Resumo</b>	<b>xi</b>
<b>Agradecimentos</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Permutations and Sorting Problems . . . . .	3
1.2 Evolution and Genome Rearrangement . . . . .	5
1.3 The Marriage Between Permutations and Genome Rearrangement . . . . .	7
1.4 Contributions and Organization . . . . .	8
<b>2 An Audit Tool for Genome Rearrangement Algorithms</b>	<b>11</b>
2.1 Introduction . . . . .	12
2.2 Background . . . . .	13
2.2.1 Modeling Genomes and Rearrangement Events . . . . .	13
2.2.2 Pairwise Genome Rearrangement Problem and Sorting . . . . .	14
2.2.3 Variants of the Rearrangement Sorting Problem . . . . .	15
2.3 Implementation . . . . .	18
2.3.1 Algorithm for Computing Rearrangement Distances . . . . .	18
2.3.2 Computing Rearrangement Distances . . . . .	23
2.3.3 Implementation of GRAAu . . . . .	36
2.4 Application of GRAAu . . . . .	38
2.4.1 Sorting by Prefix Reversals . . . . .	38
2.4.2 Sorting by Prefix Transpositions . . . . .	45
2.5 Conclusion . . . . .	53
<b>3 A General Heuristic for Genome Rearrangement Problems</b>	<b>57</b>
3.1 Introduction . . . . .	58
3.2 Background . . . . .	60
3.3 A General Heuristic . . . . .	62
3.4 Solution Database . . . . .	65
3.5 Experimental Results . . . . .	65

3.6	Conclusions . . . . .	80
<b>4</b>	<b>On Alternative Approaches for Approximating the Transposition Distance</b>	<b>81</b>
4.1	Introduction . . . . .	82
4.2	Preliminaries . . . . .	83
4.3	Algorithms . . . . .	85
4.3.1	Algorithm based on the breakpoint diagram . . . . .	85
4.3.2	Algorithm based on permutation codes . . . . .	86
4.3.3	Algorithm based on the longest increasing subsequence . . . . .	87
4.4	Computing Permutation Codes in $O(n \log n)$ Time . . . . .	91
4.5	Experimental Results and Discussion . . . . .	95
4.5.1	Experiments on small permutations . . . . .	95
4.5.2	Experiments on large permutations . . . . .	98
4.6	Conclusions . . . . .	102
<b>5</b>	<b>Sorting Signed Permutations by Short Operations</b>	<b>105</b>
5.1	Background . . . . .	106
5.2	Preliminaries . . . . .	108
5.3	Sorting by Bounded Signed Reversals . . . . .	110
5.3.1	The Vector Diagram . . . . .	110
5.3.2	Sorting by Signed Super Short Reversals . . . . .	112
5.3.3	Sorting by Signed Short Reversals . . . . .	114
5.4	Sorting by Bounded Operations . . . . .	121
5.4.1	The Permutation Graph . . . . .	121
5.4.2	Sorting by Signed Super Short Operations . . . . .	122
5.4.3	Sorting by Signed Short Operations . . . . .	126
5.5	Experimental Results . . . . .	131
5.6	Conclusions . . . . .	132
<b>6</b>	<b>Sorting Signed Circular Permutations by Super Short Reversals</b>	<b>135</b>
6.1	Introduction . . . . .	136
6.2	Sorting by Cyclic Super Short Reversals . . . . .	137
6.3	Sorting by Signed Cyclic Super Short Reversals . . . . .	140
6.4	Sorting Circular Permutations . . . . .	144
6.5	Experimental Results and Discussion . . . . .	144
6.6	Conclusions . . . . .	146
<b>7</b>	<b>Concluding Remarks</b>	<b>149</b>
	<b>Bibliography</b>	<b>153</b>



# List of Tables

2.1	Rearrangement models and values of $n$ considered in the computation of the rearrangement distances. . . . .	24
2.2	Exact values and bounds for the diameter. . . . .	25
2.3	Signed prefix reversal distance distribution in $S_n^\pm$ . . . . .	26
2.4	Signed reversal and transposition distance distribution in $S_n^\pm$ . . . . .	27
2.5	Signed reversal, transposition, and signed transreversal (type A) distance distribution in $S_n^\pm$ . . . . .	27
2.6	Transposition and signed transreversal (types A and B) distance distribution in $S_n^\pm$ . . . . .	27
2.7	Reversal distance distribution in $S_n$ . . . . .	28
2.8	Transposition distance distribution in $S_n$ . . . . .	29
2.9	Reversals and transposition distance distribution in $S_n$ . . . . .	29
2.10	Prefix reversal distance distribution in $S_n$ . . . . .	30
2.11	Prefix transposition distance distribution in $S_n$ . . . . .	31
2.12	Prefix reversal and prefix transposition distance distribution in $S_n$ . . . . .	32
2.13	Diameter, traversal diameter, and longevity of $S_n^\pm$ . . . . .	33
2.14	Diameter, traversal diameter, and longevity of $S_n$ . . . . .	34
2.15	Conjectures regarding $D(n)$ , $T(n)$ , and $L(n)$ . . . . .	35
2.16	Results from the audit of Algorithm 2. . . . .	41
2.17	Results from the audit of Algorithm 3. . . . .	42
2.18	Results obtained from the audit of Algorithm 4. . . . .	47
2.19	Results obtained from the audit of Algorithm 5. . . . .	47
3.1	List of algorithms used in this paper. . . . .	66
3.2	Average distance for each approximation algorithm for the sorting by transpositions problem. We highlight the best results produced both with and without our heuristic. . . . .	77
3.3	Percentage of instances in which each program yielded the best results for the sorting by transpositions problem. Columns do not add up to 100% because of ties. Best results highlighted. . . . .	78
3.4	Summary of the results produced for each algorithm. . . . .	79
4.1	Results obtained from the audit of the implementation of Walter, Dias, and Meidanis' algorithm. . . . .	96

4.2	Results obtained from the audit of the implementation of Benoît-Gagné and Hamel's algorithm. . . . .	96
4.3	Results obtained from the audit of the implementation of Algorithm 3, which is a constrained version of Guyer, Heath, and Vergara's heuristic. . . . .	97
4.4	Permutations $\pi^m$ of size $3m + 1$ , $m \in \{5, 6, 7\}$ , for which $\frac{p(\pi^m)}{d(\pi^m)} = \frac{3m}{m+1}$ . Note that $d(\pi^m) \geq \frac{b(\pi^m)}{3} \geq m + 1$ . . . . .	97
5.1	Results obtained from the audit of the implementation of Algorithm 15. . . . .	132
5.2	Results obtained from the audit of the implementation of Algorithm 18. . . . .	132
6.1	Matrix of the super short reversal distances among the signed circular permutations which represent the <i>Yersinia</i> genomes. . . . .	145

# List of Figures

1.1	The Cayley graph $\Gamma(G,S)$ where $G$ is the set of all permutations on 4 elements and $S$ corresponds to the three different ways of swapping the first element of a permutation with any other element. This figure is based on a picture by Labarre [89, Figure 5.2]. . . . .	5
2.1	Average speed gain on the execution of the 32 bit implementation with multiple threads. The rearrangement model consisted of reversals only.	23
2.2	Breakpoint graph $G(\pi)$ of permutation $\pi = (4\ 2\ 1\ 3)$ . . . . .	39
2.3	Performance comparison between algorithms 2 and 3 based on the results provided by GRAAu. . . . .	43
2.4	Breakpoint graphs of the permutations produced by Algorithm 2 when sorting permutation $\pi = (1\ 7\ 8\ 2\ 4\ 3\ 9\ 5\ 6)$ . . . . .	44
2.5	Performance comparison between algorithms 4 and 5 based on the results provided by GRAAu. . . . .	48
3.1	Results regarding the algorithms for the problem of sorting by prefix reversals. . . . .	68
3.2	Results regarding the algorithms for the problem of sorting by prefix reversals and prefix transpositions. . . . .	69
3.3	Results regarding the algorithms for the problem of sorting by prefix reversals, prefix transpositions, suffix reversals and suffix transpositions.	70
3.4	Results regarding the algorithms for the problem of sorting by prefix reversals and suffix reversals. . . . .	71
3.5	Results regarding the algorithms for the problem of sorting by prefix transpositions and suffix transpositions. . . . .	72
3.6	Results regarding the algorithms for the problem of sorting by prefix transpositions. . . . .	73
3.7	Results regarding the algorithms for the problem of sorting by prefix transpositions. . . . .	74
3.8	Results regarding the algorithm for the problem of sorting by reversals and transpositions. . . . .	75
3.9	Results regarding the 5 algorithms for the problem of sorting by transpositions. . . . .	76

4.1	Comparison of Walter, Dias, and Meidanis' algorithm (WDM), Benoît-Gagné and Hamel's algorithm (BH), and the constrained version of Guyer, Heath, and Vergara's heuristic (GD) based on the results provided by GRAAu. . . . .	98
4.2	Comparison of Walter, Dias, and Meidanis' algorithm (WDM), Benoît-Gagné and Hamel's algorithm (BH), the constrained version of Guyer, Heath, and Vergara's heuristic (GD), Bafna and Pevzner's algorithm (BP), Elias and Hartman's algorithm (EH), and Dias and Dias' algorithms (DD (BP) and DD (EH)) based on the average distance. . . . .	99
4.3	Relative number of times each algorithm provided the best distance. Note that more than one algorithm can have provided the best distance.	100
4.4	Relative number of times each algorithm provided the best distance. Note that more than one algorithm can have provided the best distance.	101
5.1	Vector diagram of the signed permutation $\pi = (+3 -4 +6 -1 +5 -2)$ . Note that $\text{Vec}(\pi) = 14$ . . . . .	110
5.2	Permutation graph of the signed permutation $(+3 -4 +6 -1 +5 -2)$ . . . . .	122
6.1	Phylogeny of the <i>Yersinia</i> genomes based on the super short reversal distance of the signed circular permutations. . . . .	146

# Chapter 1

## Introduction

One of the challenges of modern science is to understand how species evolve. As evolution can be viewed as a branching process, whereby new species arise from changes occurring in living organisms, the study of the evolutionary history of a group of species is commonly made by analyzing trees whose nodes represent species and edges represent evolutionary relationships. Since these relationships are referred to as phylogeny, such trees are called phylogenetic trees.

Phylogenies can be inferred from different kinds of data, from geographic and ecological, through behavioral, morphological, and metabolic, to molecular data, such as DNA. Molecular data have the advantage of being exact and reproducible, at least within experimental error, not to mention fairly easy to obtain [63, Chapter 12]. Distance-based methods form one of the three large groups of methods to infer phylogenetic trees from molecular or sequence data [96, Chapter 5]. Such methods proceed in two steps. First, the evolutionary distance is computed for every sequence pair and this information is stored in a matrix of pairwise distances. Then, a phylogenetic tree is constructed from this matrix using a specific algorithm, such as *Neighbor-Joining* [108]. Note that, in order to complete the first step, we need some method to estimate the evolutionary distance between a sequence pair. Assuming the sequence data correspond to complete genomes, we can resort to the genome rearrangement approach [54] in order to estimate the evolutionary distance.

The genome of an organism is composed of chromosomes, which can be simplistically represented as a set of ordered and oriented genes. A genome rearrangement occurs when one or more chromosomes are broken into segments and rejoined in such a way that the order or the orientation of the genes is changed. Some examples of rearrangements include the following:

- **Reversal/Inversion:** a segment of a chromosome is reversed in such a way that the order and orientation of the genes within the segment is also reversed;
- **Transposition:** a segment of a chromosome is moved to a new location in the same chromosome. The orientation of the genes within the segment is not changed;

- **Inverted transposition/Transreversal:** similar to a transposition, but the orientation of the genes within the segment is changed;
- **Translocation:** an end segment of one chromosome is exchanged with an end segment of another;
- **Fusion:** two chromosomes are merged into one;
- **Fission:** a chromosome is splitted into two;
- **Deletion:** a segment of a chromosome is lost;
- **Duplication:** a copy of some chromosome region is inserted in the genome.

We can divide rearrangements into two classes: balanced and imbalanced [66]. The difference between them is that balanced rearrangements preserve the initial set of genes while imbalanced rearrangements do not. Note that the first six rearrangements above are balanced and the last two rearrangements are imbalanced.

Before we proceed, it should be noted that large-scale mutations, such as genome rearrangements, are not the only kind of mutation that affects the DNA: it may also be altered by point mutations. Basically, the DNA can be seen as long sequence of four types of nucleotides (A, T, C, and G). A point mutation changes the DNA by substituting, deleting, or adding one nucleotide. Therefore, one can also estimate the evolutionary distance between a sequence pair resorting to a point mutation approach. For a detailed presentation of this approach, the reader is referred to the book of Lemey, Salemi, and Vandamme [96].

Using the genome rearrangement approach, one estimates the evolutionary distance between two genomes by finding the rearrangement distance between them, which is the length of the shortest sequence of rearrangement operations that transforms one genome into the other. Assuming genomes consist of a single chromosome, share the same set of genes, and contain no duplicated genes, we can represent them as permutations of integers, where each integer corresponds to a gene. If, besides the order, the orientation of the genes is also considered, then each integer has a sign, + or  $-$ , and the permutation is called a signed permutation. Similarly, we also refer to a permutation as an unsigned permutation when its elements do not have signs. Moreover, if the genomes are circular, then the permutations are also circular; otherwise, they are linear.

By representing genomes as permutations, the problem of finding the shortest sequence of operations that transforms one genome into another can be reduced to the combinatorial problem of computing the minimum number of operations necessary to transform one permutation into another. By algebraic properties of permutations, this problem can be equivalently stated as the problem of computing the minimum number of operations necessary to transform one permutation into the identity permutation

$(+1 +2 \dots +n)$ . This problem is commonly referred to as the permutation sorting problem or as the rearrangement sorting problem.

Depending on the operations allowed to sort a permutation, we have a different variant of the permutation sorting problem. This thesis presents a collection of articles concerning variants that take into account two types of operations: reversals and transpositions. Since each article provides the necessary technical background for understanding its content, the next sections aim to provide a more historical background in order to give the readers a broader perspective. Specifically, Section 1.1 and Section 1.2 give an informal historical overview of the main concepts related to this thesis, whereas Section 1.3 discusses how they have been combined to form a new research field. The last section of this introductory chapter outlines the rest of this thesis.

## 1.1 Permutations and Sorting Problems

According to Smith [112, Chapter 13], the subject of permutations appeared first in China in the *I Ching*, or *Book of Changes*. The Hindus have given no attention to the subject until Baskara II took it up on his treatise on mathematics, *Lilavati*. Permutations started to gather more attention early in the Christian Era [112, Chapter 13]. In that time, there was a close relation between mathematics and the cabala, what “led to the belief in the mysticism of arrangements and hence to the study of permutations and combinations” [112, Chapter 13].

The first time the notion of permutations appeared in a printed book was in Pacioli’s *Suma*, where “he showed how to find the number of permutations of any number of persons sitting at a table” [112, Chapter 13]. Moreover, the first time the word “permutation” appeared in a printed book was in Jacques Bernoulli’s *Ars Conjectandi* [112, Chapter 13].

As we have seen so far, the concept of permutation denoted basically an arrangements of elements, therefore it was closely related to the theory of combinations. However, as Kiernan [85] pointed out, permutations were crucial to the evolution of algebra in the 18th century. In his paper “Réflexions sur la résolution algébrique des équations”, Joseph-Louis Lagrange gave a significant contribution to the problem of the solution of equations by radicals. He observed that important properties of an equation can be deduced by examining the effect produced by permutations of its roots [85]. Nevertheless, he was neither able to give a solution to equations of fifth degree (or higher) nor able to present a proof that no solution exist. Building upon Lagrange’s work, Paolo Ruffini and Niels Henrik Abel succeeded in proving that no solution exist, although Ruffini’s contemporaries disputed the conclusiveness of his results [85]. Ultimately, Évariste Galois gave the necessary and sufficient conditions for solving an equation by radicals [85].

Augustin-Louis Cauchy was the first to define a permutation as a bijective function

from a set to itself. Moreover, he introduced the two-row notation for permutations and many important terms, still in use today [85]. For this reason, we can attribute to him the study of permutations as a subject in its own right, giving rise to the theory of permutations (and subsequently to the group theory). Since then, many mathematicians have devoted their time to develop it. In particular, we would like to cite one of them: Arthur Cayley. To the best of our knowledge, the first solution to a permutation sorting problem was due to him. His paper “Note on the Theory of Permutations” [24] shows how to derive the minimum number of two-element swaps necessary to transform a given permutation into the identity permutation (referred by him as primitive arrangement). It is also due to him [25] a fundamental tool in group theory: the Cayley graph.

Generally speaking, a group is a structure consisting of a set of elements together with a binary operation, called the group operation. Moreover, a permutation group is a group whose elements are permutations and whose group operation is the product (or composition) of permutations. Given a permutation group  $G$  and a set  $S$  of group elements belonging to  $G$ , the Cayley graph  $\Gamma(G,S)$  associated with  $G$  and  $S$  is the graph whose vertices are the elements of  $G$  and whose edges connect two vertices such that the corresponding elements can be obtained from one another using an element of  $S$ . Usually, the Cayley graph is assumed to be a connected undirected graph by considering that: (i)  $S$  is a set of generators of  $G$ , that is, every permutation in  $G$  can be obtained by composing the permutations in  $S$ ; and (ii) if a permutation belongs to  $S$ , then its inverse also does. For instance, Figure 1.1 illustrates the Cayley graph  $\Gamma(G,S)$  where  $G$  is the set of all permutations on 4 elements and  $S$  corresponds to the three different ways of swapping the first element of a permutation with any other element.

It is important to note that Cayley [24] did not formulate the problem as a permutation sorting problem. In fact, such formulation is characteristic of the genome rearrangement literature. Before that, the problem was generally formulated as a minimum factorization problem, that is, given a set of generators and a single target permutation, the problem is to find the minimum-length generator sequence whose product is the target permutation. If the set of generators is part of the problem instance, then Even and Goldreich [51] have proved that the problem is NP-hard and Jerrum [80] has proved that the problem is PSPACE-complete. These two results explain why we study permutation sorting problems in a “variant-oriented” fashion: there is little hope that a general efficient solution can be found.

Another problem inspired by the notion that a permutation can be represented as a product of generators is the group diameter problem [7,45]: given a set  $S$  of generators of a permutation group  $G$ , find the length of the longest product of generators required to reach a group element. Note that this problem can be equivalently formulated as the problem of finding the diameter of the Cayley graph  $\Gamma(G,S)$  [6]. Diameter problems are also studied in the genome rearrangement literature. For a historical development



of this and other problems on Cayley graphs, the interested reader is referred to the survey by Konstantinova [87].

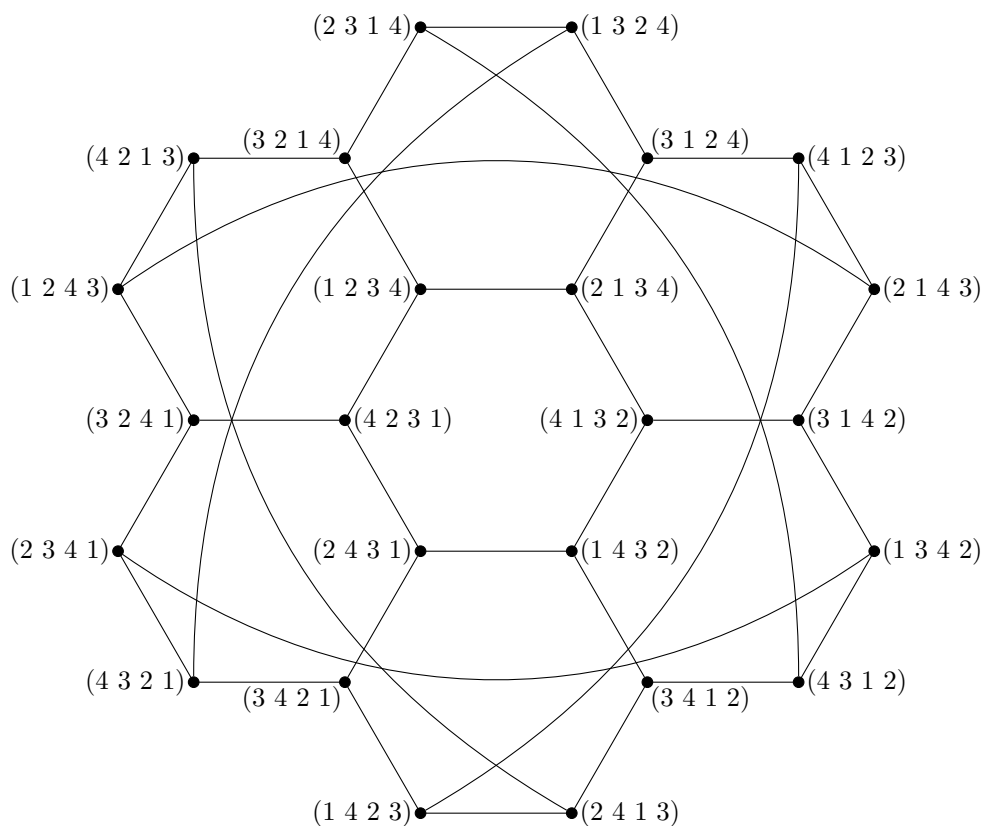


Figure 1.1: The Cayley graph  $\Gamma(G, S)$  where  $G$  is the set of all permutations on 4 elements and  $S$  corresponds to the three different ways of swapping the first element of a permutation with any other element. This figure is based on a picture by Labarre [89, Figure 5.2].

## 1.2 Evolution and Genome Rearrangement

In the introduction of this thesis, we stated that evolution can be viewed as a branching process, whereby new species arise from changes occurring in living organisms. This concept dates back to Charles Darwin. In the introduction of his *On the Origin of Species* [34], Darwin writes:

Although much remains obscure, and will long remain obscure, I can entertain no doubt, after the most deliberate study and dispassionate judgment of which I am capable, that the view which most naturalists entertain, and which I formerly entertained – namely, that each species has been independently created – is erroneous. I am fully convinced that species are

not immutable; but that those belonging to what are called the same genera are lineal descendants of some other and generally extinct species, in the same manner as the acknowledged varieties of any one species are the descendants of that species.

Although Darwin's evolutionary theory gained scientific acceptance when it was published, it faced some difficulties. The most serious of them, according to Ayala [5], was the lack of an adequate theory of inheritance to account for the variations between and within the species. Coincidentally, about the time Darwin's *On the Origin of Species* was published, Gregor Mendel published a paper which formulated the basic laws of inheritance [5]. This work, however, was not known to Darwin.

The connection between Darwin's evolutionary theory and Mendel's theory of heredity was only made in the 1920s and 1930s through the work of several geneticists, giving rise to the Synthetic Theory of Evolution, also known as the Modern Synthesis of Evolutionary Theory [5]. A key author of this theory was Theodosius Dobzhansky. Among other things, Dobzhansky's *Genetics and the Origin of Species* [43] helped to popularize the synthetic theory to other biologists.

Another important early pioneer in genetic research was Alfred Sturtevant. Among his contributions to the field are the first genetic map [113] (*i.e.* an ordering of genes within a chromosome) and the discovery of rearrangements in *Drosophila* [114]. The essence of these contributions, namely the fact that genes are linearly arranged in the chromosome and that this arrangement can change by the occurrence of chromosomal mutations, laid out the idea of using the degree of disorder between the arrangement of genes as an estimate of the evolutionary distance between species. Sturtevant and Dobzhansky [115] were the first to employ it in order to perform the phylogenetic analysis of *Drosophila*. More specifically, they have constructed a phylogeny of different strains of *Drosophila pseudoobscura* using a rearrangement scenario consisting of reversals. In a posterior work [44], they write:

The linear arrangement of genes within chromosomes is constant from generation to generation in each line of descent. The degree of this stability is comparable to that of the gene structure; genes change by mutation, chromosomes change by the occurrence of chromosomal aberrations. Strains and races of the same species, as well as distinct species, may differ in gene arrangement. . . .

In the present article we shall report the results of comparisons of the gene arrangement in the chromosomes of strains of *D. pseudoobscura* coming from different geographical regions. . . . Moreover, as pointed out in our preliminary communication . . . , a comparison of the different gene arrangements in the same chromosome may, in certain cases, throw light on the historical relationships of these structures, and consequently on the history of the species as a whole.

According to Fertin *et al.* [54], a number of studies proposed rearrangement scenarios to explain chromosomal differences between relatively close species. Moreover, these studies were based on the parsimony principle, which tells one to choose scenarios involving as few rearrangements as possible. As observed by Fertin *et al.* [54], “this principle make the connection with combinatorial optimization possible, because optimization principle meets the parsimony criterion”.

## 1.3 The Marriage Between Permutations and Genome Rearrangement

One can say that a first encounter between permutations and genome rearrangements occurred in the work of Sturtevant and Novitski [116]. In this article, they write:

With the help of Prof. Morgan Ward, a beginning has been made in the study of the mathematical consequences of successive inversions. Complete catalogs have been prepared, showing all the possible different arrangements of 2, 3, 4, 5, and 6 loci, respectively, together with the minimum number of successive inversions required to change each arrangement into a single arbitrarily chosen one. Actually, numbers were used, and the required arbitrary sequence was the ordinal one (1, 2, 3, 4, etc.).

The engagement started with the work of Watterson *et al.* [126]. In this article, they introduced the chromosome inversion problem, which consists in finding the minimum number of inversions necessary to transform a chromosome into another. They also provided some heuristics which yield upper and lower bounds to the problem. Finally, the marriage was consummated in the work of Kececioglu and Sankoff [84]. In this article, they explicitly modeled chromosomes as permutations and restated the chromosome inversion problem as the problem of sorting by reversals. Moreover, they provided a 2-approximation algorithm and a branch-and-bound exact algorithm for the problem.

Numerous works have been spawned from the marriage of permutations and genome rearrangements, some of which will be reviewed in the next chapters (for an extensive and detailed survey, the reader is referred to the book of Fertin *et al.* [54]). Among these works, we can find a number of doctoral theses, such as the ones of Hannenhalli [70], Vergara [121], Christie [30], Walter [122], Bourque [18], Dias [40], Eriksen [49], Hausen [75], Alekseyev [2], Mira [102], Labarre [89], Braga [19], Swenson [117], Ozery-Flato [105], Baudet [11], Bernt [16], Dias [35], Feijão [52], Kováč [88], and Bulteau [20].

We close this section by noting that the relationship between permutations and genome rearrangements is not exclusive. On the one hand, we have that permutation sorting problems find application in other fields, such as interconnection network design [1, 91], evaluation of OCR zoning [83, 93], and search landscape analysis [109].

On the other hand, permutations do not constitute the only model for genomes: other, somewhat more general, models have been proposed, such as strings [54, Chapter 7], graphs [54, Chapter 10], and collections of sets of genes [54, Chapter 12]. In fact, even the rearrangement operations have been generalized with concepts like double-cut-and-join [128], single-cut-or-join [53], and  $k$ -break rearrangements [3].

## 1.4 Contributions and Organization

This thesis consists of a collection of five articles that were published in peer-reviewed journals and conference proceedings in the course of the PhD. Specifically, each one of the next five chapters corresponds to one of these articles, while the last chapter concludes the thesis. The following paragraphs summarize the contents of each chapter (except the last), highlighting the main contributions.

Chapter 2 corresponds to an article [60] published in *ACM Journal of Experimental Algorithmics*. In this chapter, we present a tool, called GRAAu, to audit algorithms for permutation sorting problems. The audit consists in comparing, for all permutations of up to a given size, the distance outputted by a given algorithm with the related rearrangement distance, and then producing statistics that can be used to analyze the performance of this algorithm. We also present tightness results for some approximation algorithms regarding two variants of the permutation sorting problem: the problem of sorting by prefix reversals and the problem of sorting by prefix transpositions. The vast majority of the results presented in this chapter were also presented in the Master's thesis [56] of the author. For this reason, the importance of Chapter 2 is more related to the fact that it serves as a prelude to the other chapters (for instance, Section 2.2 introduces the basic concepts used throughout this thesis and provides a literature review of several variants of the permutation sorting problem) than to the originality of its content.

Chapter 3 corresponds to an article [39] published in *Journal of Bioinformatics and Computational Biology*. In this chapter, we present a general heuristic for permutation sorting problems. This heuristic is an improvement heuristic, that is, instead of producing solutions, it tries to improve the ones provided by other (non-optimal) algorithms. To evaluate the heuristic, we applied it to the solutions provided by 23 approximation algorithms regarding 9 variants of the permutation sorting problem that consider reversals or transpositions.

Chapter 4 corresponds to an article [62] published in *Journal of Universal Computer Science*. The best known algorithms for the problem of sorting by transpositions are based on a standard tool for tackling permutation sorting problems, the cycle graph. In an attempt to bypass it, a few researches proposed algorithms based on alternative tools. In Chapter 4, we address three of these algorithms: a 2.25-approximation algorithm proposed by Walter, Dias, and Meidanis [124], a 3-approximation algorithm proposed by Benoît-Gagné and Hamel [14], and a heuristic proposed by

Guyer, Heath, and Vergara [69]. On the theoretical side, we close a missing gap on the proof of the approximation ratio of Benoît-Gagné and Hamel’s algorithm [14] and we demonstrate a way to run their algorithm in  $O(n \log n)$  time. Moreover, we propose a minor adaptation to Guyer, Heath, and Vergara’s heuristic [69] that allow us to prove an approximation bound of 3. On the evaluation side, we present experimental data indicating that Walter, Dias, and Meidanis’ algorithm [124] is the best of the algorithms based on alternative approaches and that it is the only one comparable to the algorithms based on the cycle graph.

Chapter 5 corresponds to an article [61] published in *Algorithms for Molecular Biology*. In this chapter, we introduce four new variants of the permutations sorting problem: (i) the problem of sorting a signed permutation by reversals of length at most 2; (ii) the problem of sorting a signed permutation by reversals of length at most 3; (iii) the problem of sorting a signed permutation by reversals and transpositions of length at most 2; and (iv) the problem of sorting a signed permutation by reversals and transpositions of length at most 3. We present polynomial-time algorithms for problems (i) and (iii), a 5-approximation for problem (ii), and a 3-approximation for problem (iv). Moreover, we show that the expected approximation ratio of the 5-approximation algorithm is not greater than 3 for random signed permutations with more than 12 elements. Finally, we present experimental results that show that the approximation ratios of the approximation algorithms cannot be smaller than 3. In particular, this means that the approximation ratio of the 3-approximation algorithm is tight.

Chapter 6 corresponds to an article [57] published in the proceedings of the *11th International Symposium on Bioinformatics Research and Applications*. In this chapter, we consider the problem of sorting a signed circular permutation by reversals of length at most 2 and present a polynomial-time algorithm for solving it. We also perform an experiment for inferring distances and phylogenies for published *Yersinia* genomes and compare the results with the phylogenies presented in previous works.



## Chapter 2

# An Audit Tool for Genome Rearrangement Algorithms \*

**Abstract:** We consider the combinatorial problem of sorting a permutation using a minimum number of rearrangement events, which finds application in the estimation of evolutionary distance between species. Many variants of this problem, which we generically refer to as the rearrangement sorting problem, have been tackled in the literature, and for most of them, the best known algorithms are approximations or heuristics. In this article, we present a tool, called GRAAu, to aid in the evaluation of the results produced by these algorithms. To illustrate its application, we use GRAAu to evaluate the results of four approximation algorithms regarding two variants of the rearrangement sorting problem: the problem of sorting by prefix reversals and the problem of sorting by prefix transpositions. As a result, we show that the approximation ratios of three algorithms are tight and conjecture that the approximation ratio of the remaining one is also tight.

---

\* *Gustavo Rodrigues Galvão and Zanoni Dias. An audit tool for genome rearrangement algorithms. ACM Journal of Experimental Algorithmics, Volume 19, Article 1.7, 34 pages, 2014. Copyright 2014 Association for Computing Machinery, Inc. DOI: <http://dx.doi.org/10.1145/2661633>*

## 2.1 Introduction

One of the problems faced by biologists is how to estimate the evolutionary distance between species. A well-accepted approach for treating this problem is the genome rearrangement approach. It proposes to estimate the evolutionary distance between two species using the rearrangement distance between their genomes, which is the length of the shortest sequence of genome-wide mutations, called rearrangement events, that transforms one genome into the other. The problem of finding this sequence is called the pairwise genome rearrangement problem.

Representing genomes as permutations, in which genes appear as elements, the pairwise genome rearrangement problem can be equivalently stated as a combinatorial problem of sorting a permutation using a minimum number of rearrangement events, which is called the rearrangement sorting problem (see Section 2.2.2 for details). This problem varies according to the types of rearrangement events under consideration. Here, we focus on variants considering two types of events: reversals and transpositions. It is known that finding optimal solutions for these variants is difficult; therefore, most of the proposed algorithms – which we denominate generically as genome rearrangement algorithms – are approximations or heuristics (see Section 2.2.3 for details). To analyze these algorithms in practice, researchers often perform what we refer to as audit. Basically, it consists in comparing, for a great number of small problem instances, the distance output by a genome rearrangement algorithm to the related rearrangement distance. The process of auditing an algorithm consumes a considerable amount of time and effort, so we decided to build a tool to aid it. This tool was named GRAAu, which is an acronym for Genome Rearrangement Algorithm Auditor.

In addition to GRAAu, we present an exact algorithm for computing rearrangement distances that is more efficient in terms of memory than any algorithm we have found in literature. Additionally, we present conjectures on how the rearrangement distance are distributed and validate them regarding their maximum value, which is the greatest value that the rearrangement distance of a permutation can reach considering all permutations with the same number of elements. Finally, we use GRAAu to evaluate four genome rearrangement algorithms – two approximation algorithms for the problem of sorting by prefix reversals and two approximation algorithms for the problem of sorting by prefix transpositions – and we provide tight examples for three of them, which were obtained through an analysis of the audit results produced by GRAAu.

The rest of this article is organized as follows. Section 2.2 gives basic definitions and notation of the article. In Section 2.3, we describe the approach used to compute the rearrangement distances of the greatest number of permutations we could. Using this approach, we have computed the rearrangement distances of small permutations regarding 10 rearrangement models considered in the literature, and in Section 2.3.2 we present their distribution. In Section 2.3.3, we give a general description on how



GRAAu works. Finally, in Section 2.4, we show the results we have obtained by auditing four approximation algorithms for solving two variants of the rearrangement sorting problem: the problem of sorting by prefix reversals and the problem of sorting by prefix transpositions.

## 2.2 Background

In this section, we introduce the basic concepts used in this article and provide a brief literature review of the rearrangement sorting problem. Most of this section is based on the book of Fertin *et al.* [54], which we consider the main reference in the field of genome rearrangements.

### 2.2.1 Modeling Genomes and Rearrangement Events

In genome rearrangement literature, one can find many approaches to model genomes. Basically, the difference between them are the assumptions made on the genomes. Assuming they consist of a single linear chromosome, share the same set of genes, and contain no duplicated genes, we can model them as permutations of integers, where each integer corresponds to a gene. If the orientation of the genes is known, then each integer has a sign, + or −, indicating its orientation.

An unsigned permutation  $\pi$  is a bijection over the set  $\{1, 2, \dots, n\}$ . A classical notation used in combinatorics for denoting an unsigned permutation  $\pi$  is the two-row notation

$$\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix},$$

where  $\pi_i \in \{1, 2, \dots, n\}$  for  $1 \leq i \leq n$ . It indicates that the image of element  $i \in \{1, 2, \dots, n\}$  is  $\pi_i$  or, in other words,  $\pi(1) = \pi_1$ ,  $\pi(2) = \pi_2$ ,  $\dots$ ,  $\pi(n) = \pi_n$ . The notation used in genome rearrangement literature, which is the one we will adopt, is the one-row notation  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . We say that  $\pi$  has size  $n$ .

A signed permutation  $\pi$  is a bijection over the set  $\{-n, \dots, -2, -1, 1, 2, \dots, n\}$ . Its two-row notation is

$$\pi = \begin{pmatrix} -n & \dots & -2 & -1 & 1 & 2 & \dots & n \\ -\pi_n & \dots & -\pi_2 & -\pi_1 & \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \dots, n\}$  for  $1 \leq i \leq n$ . In one-row notation, we drop the mapping of the negative elements since  $\pi(-i) = -\pi(i)$  for all  $i \in \{1, 2, \dots, n\}$ , and we also write  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . By abuse of notation, we say that  $\pi$  has size  $n$ . This way, we can handle permutations in a unified way, making distinctions between signed and unsigned permutations when necessary.

The composition of permutations  $\pi$  and  $\sigma$ , denoted by  $\pi \circ \sigma$ , results in the permutation  $\gamma$  such that  $\gamma(i) = \pi(\sigma(i))$ . In other words,  $\gamma = (\pi_{\sigma_1} \pi_{\sigma_2} \dots \pi_{\sigma_n})$ . Such an

operation allows us to model not only genomes as permutations, but also rearrangement events, in such a way that a rearrangement event  $\rho$  transforms a genome  $\pi$  into the genome  $\pi \circ \rho$ . In Section 2.2.3 we define the rearrangement events that will be regarded in this work.

The composition operation induces a group structure on the set of all permutations of a given size. The group formed by all unsigned permutations of size  $n$  with  $\circ$  is called symmetric group and is denoted by  $S_n$ . The group formed by all signed permutations of size  $n$  with  $\circ$  is called hyperoctahedral group and is denoted by  $S_n^\pm$ .

Let  $G$  be a subset of  $S_n$  ( $S_n^\pm$ ) such that any permutation in  $S_n$  ( $S_n^\pm$ ) can be obtained by the composition of the permutations in  $G$ . The permutations belonging to  $G$  are said to be generators of  $S_n$  ( $S_n^\pm$ ). Moreover, if  $\pi^{-1} \in G$  for all  $\pi \in G$ , then we say that  $G$  is symmetric. A set  $M$  formed by generators of  $S_n$  ( $S_n^\pm$ ) is called a rearrangement model if all generators belonging to  $M$  model rearrangement events and  $M$  is symmetric.

## 2.2.2 Pairwise Genome Rearrangement Problem and Sorting

The pairwise genome rearrangement problem on permutations can be formulated as follows. Given two permutations  $\pi$  and  $\sigma$ , find the shortest sequence of generators  $\rho_1, \rho_2, \dots, \rho_t$  belonging to a rearrangement model  $M$  such that  $\pi \circ \rho_1 \circ \rho_2 \circ \dots \circ \rho_t = \sigma$ . The length of this sequence is the rearrangement distance between  $\pi$  and  $\sigma$  with respect to  $M$ , denoted by  $d_M(\pi, \sigma)$ . Since  $M$  is symmetric, we have that  $d_M(\pi, \sigma) = d_M(\sigma, \pi)$  because

$$\pi \circ \rho_1 \circ \rho_2 \circ \dots \circ \rho_t = \sigma \iff \sigma \circ \rho_t^{-1} \circ \rho_{t-1}^{-1} \circ \dots \circ \rho_1^{-1} = \pi.$$

The problem of sorting a permutation by means of a minimum number of rearrangement events, which we generically refer to as the rearrangement sorting problem, is formally defined as follows. Given a permutation  $\gamma$ , find the minimum-length sequence of generators  $\rho_1, \rho_2, \dots, \rho_t$  belonging to a rearrangement model  $M$  such that  $\gamma \circ \rho_1 \circ \rho_2 \circ \dots \circ \rho_t = \iota$ .

Note that rearrangement distance between permutations  $\pi$  and  $\sigma$  with respect to  $M$  is equal to the rearrangement distance between permutations  $\sigma^{-1} \circ \pi$  and  $\iota$  with respect to  $M$  because

$$\pi \circ \rho_1 \circ \rho_2 \circ \dots \circ \rho_t = \sigma \iff (\sigma^{-1} \circ \pi) \circ \rho_1 \circ \rho_2 \circ \dots \circ \rho_t = \iota.$$

It implies that we can reduce the pairwise genome rearrangement problem to the rearrangement sorting problem. As well, for simplicity's sake, we define the rearrangement distance of a permutation  $\pi$  with respect to  $M$ , denoted by  $d_M(\pi)$ , as the rearrangement distance between  $\pi$  and  $\iota$  with respect to  $M$ , that is,  $d_M(\pi) = d_M(\pi, \iota)$ .

When we defined the rearrangement sorting problem, we have fixed the rearrangement model beforehand. This means that depending on the rearrangement model

being considered, we have a different variant of the rearrangement sorting problem. The next section gives an overview of some variants that are relevant to this work.

### 2.2.3 Variants of the Rearrangement Sorting Problem

In this section, we provide a brief literature review of variants of the rearrangement sorting problem that will be treated in more detail in this article. Although some of these variants were not originally motivated by genome rearrangements, and in fact have little biological relevance, they have been swallowed by the field and now are presented as genome rearrangement problems [54].

A reversal  $r(i, j)$ ,  $1 \leq i \leq j \leq n$ , is a rearrangement event that transforms an unsigned permutation  $\pi = (\pi_1 \ \pi_2 \ \dots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \dots \ \pi_{j-1} \ \pi_j \ \pi_{j+1} \ \dots \ \pi_n)$  into the unsigned permutation  $\pi \circ r(i, j) = (\pi_1 \ \pi_2 \ \dots \ \pi_{i-1} \ \pi_j \ \pi_{j-1} \ \dots \ \pi_{i+1} \ \pi_i \ \pi_{j+1} \ \dots \ \pi_n)$ . In other words,  $r(i, j)$  is the unsigned permutation  $(1 \ 2 \ \dots \ i-1 \ j \ j-1 \ \dots \ i+1 \ i \ j+1 \ \dots \ n)$ .

The variant of the rearrangement sorting problem that considers a rearrangement model composed only by reversals is called problem of sorting by reversals. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as reversal distance. Caprara [23] has shown that the problem of sorting by reversals is NP-hard. Watterson *et al.* [126] were the first to present an approximation algorithm for this problem, which is a  $\frac{n-1}{2}$ -approximation. Kececioglu and Sankoff [84] were the first to present an approximation algorithm with constant factor; they have presented a 2-approximation algorithm. The best known result was presented by Berman, Hannenhalli, and Karpinski [15], which is a 1.375-approximation algorithm.

A prefix reversal  $pr(i)$ ,  $2 \leq i \leq n$ , is a rearrangement event equivalent to the reversal  $r(1, i)$ . The variant of the rearrangement sorting problem that considers a rearrangement model composed only by prefix reversals is called the problem of sorting by prefix reversals. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as prefix reversal distance. The problem of sorting by prefix reversals is also known as the pancake sorting problem and was introduced by Dweighter [46]. Bulteau *et al.* [21] proved that this problem is NP-hard, and the best known approximation algorithm for solving it was developed by Fischer and Ginzinger [55]. Such algorithm is a 2-approximation. Williams [127] has considered the problem of unsorting permutations by prefix reversals – that is, the problem of exploring all permutations generated by prefix reversals. He has presented a new data structure, called *boustrophedon linked list*, which allows substrings of any length to be reversed in constant time. In particular, this data structure can be used to perform a prefix reversal  $pr(i)$  in  $O(1)$ -time instead of  $O(i)$ -time.

A signed reversal  $sr(i, j)$ ,  $1 \leq i \leq j \leq n$ , is a rearrangement event that transforms a signed permutation  $\pi = (\pi_1 \ \pi_2 \ \dots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \dots \ \pi_{j-1} \ \pi_j \ \pi_{j+1} \ \dots \ \pi_n)$  into the

signed permutation  $\pi \circ sr(i, j) = (\pi_1 \ \pi_2 \ \dots \ \pi_{i-1} \ \underline{-\pi_j \ -\pi_{j-1} \ \dots \ -\pi_{i+1} \ -\pi_i} \ \pi_{j+1} \ \dots \ \pi_n)$ . In other words,  $sr(i, j)$  is the signed permutation  $(1 \ 2 \ \dots \ i-1 \ \underline{-j \ -(j-1)} \ \dots \ -(i+1) \ \underline{-i \ j+1} \ \dots \ n)$ .

The variant of the rearrangement sorting problem that considers a rearrangement model composed only by signed reversals is called the problem of sorting by signed reversals. In addition, the rearrangement distance of a signed permutation with respect to that rearrangement model is referred to as signed reversal distance. Hannenhalli and Pevzner [71] were the first to solve the problem of sorting by signed reversals, presenting an optimal algorithm that runs in  $O(n^4)$  time. Some refinements had been made in this algorithm over the years until Tannier *et al.* [119] presented an algorithm that runs in  $O(n^{\frac{3}{2}}\sqrt{\log n})$  time. Bader *et al.* [8] have shown how to compute the signed reversal distance (without obtaining the signed reversal sequence) in linear time.

A signed prefix reversal  $spr(i)$ ,  $1 \leq i \leq n$ , is a rearrangement event equivalent to the signed reversal  $sr(1, i)$ . The variant of the rearrangement sorting problem that considers a rearrangement model composed only by signed prefix reversals is called the problem of sorting by signed prefix reversals. In addition, the rearrangement distance of a signed permutation with respect to that rearrangement model is referred to as signed prefix reversal distance. The problem of sorting by signed prefix reversals is also known as the burnt pancake sorting problem and was introduced by Cohen and Blum [31]. The best known approximation algorithm for solving this problem is a 2-approximation developed by them. The complexity of the problem is unknown.

A transposition  $t(i, j, k)$ ,  $1 \leq i < j < k \leq n+1$ , is a rearrangement event that transforms a permutation  $\pi = (\pi_1 \ \dots \ \pi_{i-1} \ \underline{\pi_i \ \dots \ \pi_{j-1}} \ \underline{\pi_j \ \dots \ \pi_{k-1}} \ \pi_k \ \dots \ \pi_n)$  into the permutation  $\pi \circ t(i, j, k) = (\pi_1 \ \dots \ \pi_{i-1} \ \underline{\pi_j \ \dots \ \pi_{k-1}} \ \underline{\pi_i \ \dots \ \pi_{j-1}} \ \pi_k \ \dots \ \pi_n)$ . In other words,  $t(i, j, k)$  is the permutation  $(1 \ 2 \ \dots \ i-1 \ \underline{j \ j-1} \ \dots \ k-1 \ \underline{i} \ \dots \ j-1 \ k \ \dots \ n)$ .

The variant of the rearrangement sorting problem that considers a rearrangement model composed only by transpositions is called the problem of sorting by transpositions. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as transposition distance. Bulteau *et al.* [22] have proved that the problem of sorting by transpositions is NP-hard. Bafna and Pevzner [10] were the first to present approximation algorithms for this problem, and the best one had a 1.5 approximation factor. The best known result was presented by Elias and Hartman [48], which is a 1.375-approximation algorithm.

A prefix transposition  $pt(i, j)$ ,  $2 \leq i < j \leq n$ , is a rearrangement event equivalent to the transposition  $t(1, i, j)$ . The variant of the rearrangement sorting problem that considers a rearrangement model composed only by prefix transpositions is called the problem of sorting by prefix transpositions. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as prefix transposition distance. This problem was introduced by Dias and Meidanis [42], and the best known approximation algorithm for solving it is a 2-approximation

developed by them. The complexity of the problem is unknown.

The variant of the rearrangement sorting problem that considers a rearrangement model composed by reversals and transpositions is called the problem of sorting by reversals and transpositions. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as reversal and transposition distance. Walter *et al.* [123] were the first to present an approximation algorithm for this problem, which is a 3-approximation algorithm. Rahman *et al.* [107] have presented a  $2k$ -approximation algorithm where  $k$  is the approximation ratio of the algorithm used for cycle decomposition. For the best known value of  $k$  at the time of publication, the approximation ratio was  $2.8386+\delta$  for any  $\delta > 0$ . The complexity of the problem is unknown.

The variant of the rearrangement sorting problem that considers a rearrangement model composed by signed reversals and transpositions is called the problem of sorting by signed reversals and transpositions. In addition, the rearrangement distance of a signed permutation with respect to that rearrangement model is referred to as signed reversal and transposition distance. Walter *et al.* [123] have presented a 2-approximation algorithm for this problem, and this has been the best known result. The complexity of the problem is unknown.

The variant of the rearrangement sorting problem that considers a rearrangement model composed by prefix reversals and prefix transpositions is called the problem of sorting by prefix reversals and prefix transpositions. In addition, the rearrangement distance of an unsigned permutation with respect to that rearrangement model is referred to as prefix reversal and prefix transposition distance. Sharmin *et al.* [111] have presented 3-approximation algorithm for this problem, and this has been the best known result. The complexity of the problem is unknown.

Some researchers also considered a further type of rearrangement event, called reversal+transposition or transreversal. A signed transreversal of type A  $tr_a(i, j, k)$ ,  $1 \leq i < j < k \leq n+1$ , is a rearrangement event that transforms a signed permutation  $\pi$  into the signed permutation  $\pi \circ tr_a(i, j, k) = \pi \circ sr(i, j-1) \circ t(i, j, k)$ . A signed transreversal of type B  $tr_b(i, j, k)$ ,  $1 \leq i < j < k \leq n+1$ , is a rearrangement event that transforms a signed permutation  $\pi$  into the signed permutation  $\pi \circ tr_b(i, j, k) = \pi \circ sr(j, k-1) \circ t(i, j, k)$ .

The variant of the rearrangement sorting problem that considers a rearrangement model composed by signed reversals, transpositions, and signed transreversals of type A is called the problem of sorting by signed reversals, transpositions, and signed transreversals (type A). In addition, the rearrangement distance of signed permutation with respect to that rearrangement model is referred to as signed reversal, transposition, and signed transreversal (type A) distance. Gu *et al.* [68] have presented 2-approximation algorithm for this problem. Its complexity is unknown.

The variant of the rearrangement sorting problem that considers a rearrangement model composed by transpositions and signed transreversals of types A and B is called

the problem of sorting by transpositions and signed transreversals (types A and B). In addition, the rearrangement distance of signed permutation with respect to that rearrangement model is referred to as transposition, and signed transreversal (types A and B) distance. Hartman and Sharan [74] have presented a 1.5-approximation algorithm for this problem. Its complexity is unknown.

## 2.3 Implementation

In this section, we describe all of the steps toward the implementation of GRAAu. Section 2.3.1 describes the algorithm that we have developed to compute the rearrangement distances and compares it with other algorithms presented in the literature. This algorithm has allowed us to compute the distribution of the rearrangement distances for permutation sizes that have never been considered before. Section 2.3.2 presents these distributions along with some conjectures designed as an attempt to better characterize how the rearrangement distances are distributed. This section also describes how these data were stored and maintained in a database called the Rearrangement Distance Database. Finally, Section 2.3.3 describes how GRAAu works.

### 2.3.1 Algorithm for Computing Rearrangement Distances

Since we were interested in variants of the rearrangement sorting problem that do not have polynomial time solutions, we could think of basically two approaches for computing the rearrangement distance of all permutation in  $S_n$  ( $S_n^\pm$ ) with respect to a rearrangement model  $M$ :

1. to perform a breadth-first search in  $S_n$  ( $S_n^\pm$ ): initialize a permutation queue  $Q$  with  $\iota$  and set its distance to 0. While  $Q$  is not empty, remove a permutation  $\pi$  from  $Q$ , report  $\pi$  and  $d_M(\pi)$ , and compute all permutations that can be generated from  $\pi$  applying on it every possible rearrangement event belonging to  $M$ . The ones that have not been generated yet are added to  $Q$ , and their distances are set to  $d_M(\pi) + 1$ .
2. to develop an exact algorithm, which would execute in exponential time, for each variant of rearrangement sorting problem, and then run each algorithm for all permutations in  $S_n$  ( $S_n^\pm$ ). Examples of exact algorithms proposed in the literature to solve variants of the rearrangement sorting problem can be divided between branch-and-bound and linear programming strategies.

We have adopted approach (1) for mainly two reasons:

- **Simplicity and Correctness.** Implementing a breadth-first search in  $S_n$  ( $S_n^\pm$ ) is simpler and, therefore, less susceptible to errors than implementing a branch-and-bound or a linear programming algorithm. As an example, we refer to the

branch-and-bound algorithm developed by Kececioglu and Sankoff [84] for one variant of the rearrangement sorting problem. Its implementation comprises thousands of lines of code and, as we show in Section 2.3.2, it did not compute the rearrangement distances correctly.

- **Flexibility.** Considering that we were interested in taking into account a number of variants of the rearrangement sorting problem, it would be highly desirable if the approach could be easily adaptable. This is not the case of approach (2), as branch-and-bound and linear programming strategies tend to be highly variant specific because they rely on bounds or restrictions that are particular to each variant.

Our main concern when implementing approach (1) was how to optimize memory usage since we knew from literature that this was the bottleneck. Given the simplicity of the approach, the only point in this direction capable of being optimized was the way in which permutations would be represented. When we look at the definition of a permutation, the first representation that comes in mind is a vector of integers, but would there a way to represent permutations more concisely? The answer is yes, by representing them as natural numbers<sup>1</sup>.

### Ranking and Unranking Functions

To represent permutations as natural numbers, it is not only necessary to map a permutation into a natural number but it is also necessary to map a natural number into a permutation because we need to compose permutations to simulate the rearrangement events. This means that we need bijective functions

$$\begin{aligned} f &: S_n \rightarrow \{0, 1, \dots, n! - 1\}, \\ f^{-1} &: \{0, 1, \dots, n! - 1\} \rightarrow S_n, \\ g &: S_n^\pm \rightarrow \{0, 1, \dots, 2^n n! - 1\}, \text{ and} \\ g^{-1} &: \{0, 1, \dots, 2^n n! - 1\} \rightarrow S_n^\pm. \end{aligned}$$

We say that functions  $f$  and  $g$  rank a permutation, therefore they are called ranking functions. On the other hand, we say that functions  $f^{-1}$  and  $g^{-1}$  unrank a permutation, therefore they are called unranking functions.

For computing  $f$  and  $f^{-1}$ , we have used the ranking and unranking algorithms presented by Myrvold and Ruskey [104], which run in linear time on the size of the permutations. Unfortunately, we could not find algorithms for computing  $g$  and  $g^{-1}$  in the literature, therefore we developed a method to define these functions using  $f$  and  $f^{-1}$  respectively. Before presenting this method, we must introduce some definitions.

We define the sign vector  $sv(\pi) = [sv(\pi_1), sv(\pi_2), \dots, sv(\pi_n)]$  of a permutation  $\pi \in S_n^\pm$  in such a way that  $sv(\pi_i) = 1$  if  $\pi_i < 0$  and  $sv(\pi_i) = 0$  if  $\pi_i > 0$  for all

---

<sup>1</sup>There has been some work on succinctly representing permutations (see [103]), but representing permutations as natural numbers is more suitable for our purposes.

$1 \leq i \leq n$ . We define the modular permutation  $m(\pi)$  of a permutation  $\pi \in S_n^\pm$  as  $m(\pi) = (|\pi_1| \mid |\pi_2| \mid \dots \mid |\pi_n|)$ . It is easy to note that a permutation  $\pi \in S_n^\pm$  can be uniquely represented by modular permutation  $m(\pi) \in S_n$  and by sign vector  $sv(\pi)$ . Considering that we can view the sign vector of a permutation  $\pi \in S_n^\pm$  as the binary number  $sv(\pi_1)sv(\pi_2)\dots sv(\pi_n)$ , we define the bijective function  $h : V_n \rightarrow \{0, 1 \dots, 2^n - 1\}$  such that  $h(sv(\pi)) = \sum_{i=1}^n 2^{i-1}sv(\pi_{n+1-i})$ , where  $V_n = \{sv(\pi) : \pi \in S_n^\pm\}$ . Since  $h$  is defined analogously to the conversion of binary numbers to decimal numbers, the function  $h^{-1} : \{0, 1 \dots, 2^n - 1\} \rightarrow V_n$  is defined analogously to the conversion of decimal numbers to binary numbers.

Now, we can define  $g$  from  $f$  and  $h$  in such a way that  $g(\pi) = 2^n f(m(\pi)) + h(sv(\pi))$  for any  $\pi \in S_n^\pm$ . Moreover,  $g^{-1}$  is well defined from  $f^{-1}$  and  $h^{-1}$  because, given  $g(\pi)$  of a permutation  $\pi \in S_n^\pm$ , we have that  $m(\pi) = f^{-1}(\frac{g(\pi)-r}{2^n})$  and that  $sv(\pi) = h^{-1}(r)$ , where  $r = h(sv(\pi)) = g(\pi) \bmod 2^n$ . Since functions  $f$ ,  $h$ ,  $f^{-1}$ , and  $h^{-1}$  can be computed in linear time on the size of permutations, functions  $g$  and  $g^{-1}$  also can.

## The Algorithm

With both ranking and unranking functions in hand, it is possible to present the algorithm for computing the rearrangement distances of all permutations in  $S_n$  with respect to a rearrangement model  $M$  (Algorithm *AllDistances*). We omit the description of the algorithm that computes the rearrangement distances of all permutations in  $S_n^\pm$  with respect to a rearrangement model  $M$  because it is trivially derivable from Algorithm *AllDistances*.

Algorithm *AllDistances* takes as input two parameters: an integer number  $n$ , which is the size of permutations, and a rearrangement model  $M$ . Initially, the algorithm creates two vectors, namely  $Q$  and  $D$ , of size  $|S_n|$ , and then sets all the elements of vector  $D$  to  $-1$ . Vector  $Q$  is used to hold a queue of the permutations that are being generated – that is, we have that  $Q[i] = f(\pi)$  such that  $\pi$  was the  $i$ -th generated permutation during algorithms execution. Vector  $D$  is used to store the rearrangement distances of the permutations already generated – that is, we have that  $D[i] = d_M(f^{-1}(i))$  if permutation  $f^{-1}(i)$  has already been generated or  $D[i] = -1$  otherwise. It means that, in addition to storing the rearrangement distances, vector  $D$  also indicates whether a permutation has already been generated. The variables *next* and *last* are used to manage the queue  $Q$ : variable *next* holds the position in the queue containing the next permutation to be processed, whereas variable *last* holds the position in the queue where a new generated permutation must be inserted. At the end of its execution, the algorithm returns vector  $D$ .

Initially, the identity permutation is inserted in  $Q$  (line 4), its rearrangement distance (zero) is stored in  $D$  (line 5), and the variable *last* is set to 2 (line 7). We claim that the while loop (lines 8–21) maintains two loops invariants: (i)  $last = |Q| + 1$ , where  $|Q|$  is the number of permutations in  $Q$ , and (ii) vector  $D$  stores the rearrangement distances of all permutations in  $Q$ . This is because each time a



**Algorithm 1:** AllDistances

---

**Data:** An integer  $n$ , which is the size of permutations, and a rearrangement model  $M$ .

**Result:** A vector containing the rearrangement distances of all permutations in  $S_n$  with respect to  $M$ .

```

1 Let  $Q$  and  $D$  be two vectors of size  $|S_n|$ ;
2 Initialize  $D$  such that  $D[i] = -1, i \in \{1, 2, \dots, |S_n|\}$ ;
3  $i \leftarrow \text{Rank}(\iota)$ ;
4  $Q[1] \leftarrow i$ ;           ▷ insert the rank of the identity permutation in the queue
5  $D[i] \leftarrow 0$ ;           ▷ and set its rearrangement distance
6  $next \leftarrow 1$ ;
7  $last \leftarrow 2$ ;
8 while  $last \leq |S_n|$  do
9    $i \leftarrow Q[next]$ ;   ▷ retrieve from the queue the rank of the next permutation
10   $d \leftarrow D[i]$ ;       ▷ and obtain its rearrangement distance
11   $\pi \leftarrow \text{Unrank}(i)$ ;
12   $next \leftarrow next + 1$ ;
13  for  $\rho \in M$  do
14     $i \leftarrow \text{Rank}(\pi \circ \rho)$ ;
15    if  $D[i] = -1$  then
16       $Q[last] \leftarrow i$ ; ▷ insert the rank of the new permutation in the queue
17       $D[i] \leftarrow d + 1$ ;   ▷ and set its rearrangement distance
18       $last \leftarrow last + 1$ ;
19    end
20  end
21 end
22 return  $D$ ;
```

---

permutation, say  $\pi$ , is inserted in  $Q$ , vector  $D$  is updated with the value of  $d_M(\pi)$ , and the variable  $last$  is incremented by one unit. By definition,  $M$  is composed by generators of  $S_n$ , therefore all permutations in  $S_n$  will eventually be generated and inserted in  $Q$ . Moreover, each permutation is inserted in  $Q$  only once. These facts imply that  $last = |S_n| + 1$  when  $|Q| = |S_n|$ , what causes the while loop (lines 8–21) to terminate. In this moment, all permutations in  $S_n$  have been inserted in  $Q$ , and consequently their rearrangement distances have been stored in  $D$ .

Regarding the complexity of Algorithm *AllDistances*, each of lines 4–7, 9, 10, 12, and 15–19 takes constant time; each of lines 3, 11, and 14 takes  $O(n)$  time; each of lines 1 and 2 takes  $O(|S_n|)$  time; the while loop of lines 8 through 21 is executed  $O(|S_n|)$  times; finally, the for loop of lines 13 through 20 takes  $O(n|M|)$  time. Therefore, Algorithm *AllDistances* runs in  $O(n|M||S_n|)$  time, which means that it runs in time exponential in the size of permutations, but in time polynomial in the number of permutations.

## Implementation and Discussion

To compensate for the fact that the algorithm has exponential time complexity, we have developed a way to parallelize it. Basically, the idea was to encapsulate the while loop in a thread and then to create multiple threads. As a result, it created a race condition on vectors  $Q$  and  $D$ , as well as on the variables  $next$  and  $last$ . Thus, it was necessary to synchronize all threads, avoiding that two or more of them write on the same common data at the same time. This could be accomplished by using a binary semaphore.

We have implemented two multithreaded versions of the algorithm: in one version, we represented each permutation as an unsigned integer of 32 bits; in the other, as an unsigned integer of 64 bits. Therefore, the 32 bits version can handle all unsigned permutations of up to 12 elements and all signed permutations of up to 10 elements, whereas the 64 bits version can handle all unsigned permutations of up to 20 elements and all signed permutations of up to 16 elements. It means that the 64 bits version can handle more permutations but needs twice as much memory. Both versions support a number of rearrangement models considered in the literature as described in Section 2.3.2. They were implemented in C, using *pthread* library for dealing with threads. The source code is available for download at

<http://mirza.ic.unicamp.br:8080>.

Vergara [121] has developed an algorithm similar to ours for computing the rearrangement distances of all permutations in  $S_n$  with respect to a rearrangement model  $M$ . Although he has also employed the idea of representing permutations as natural numbers, one cannot say that he did it for optimizing memory usage. This is because Vergara [121] said to have ranked permutations as follows. He ordered the permutations in  $S_n$  in lexicographical order, then he mapped the permutations to integers considering this order. The problem is that ordering all permutations in  $S_n$  in lexicographical order implies in representing each of them using some kind of data structure that allows element-to-element comparison, such as vectors. Therefore, there is no advantage in terms of memory usage when ranking permutations that way.

Dias and Meidanis [42] have computed the prefix transposition distance of all unsigned permutations of up to 11 elements and stated that their method would need 30GB of physical memory to compute prefix transposition distances of all unsigned permutations of 12 elements, what made the computation for  $n = 12$  impossible (they had a computer with 8GB of RAM). Similarly, Walter *et al.* [124] could not compute the transposition distance of all unsigned permutations of 12 elements because they would need a machine with 18GB of RAM. Our 32 bit implementation needs 2.4GB of physical memory to compute the rearrangement distance of all unsigned permutations of 12 elements regarding any rearrangement model. This means that our method uses approximately 12 times less memory than the method used by Dias and Meidanis [42] and approximately 7 times less memory than the method used by Walter *et al.* [124].

There are further publications where the rearrangement distance of all permutations of up to a given size were computed for a particular rearrangement model [14, 37, 89, 125], but they do not provide information about memory usage, so we cannot make fair comparisons. Nevertheless, since these publications do not present the rearrangement distances of permutations with more than 11 elements, we infer that the methods used by their authors for computing rearrangement distances had reached a limit very much like the methods used by the authors we cited earlier.

Regarding the running time performance, we have performed an experiment to measure the average speed gain on the execution of the 32 bit implementation with multiple threads. The source code was compiled with gcc version 4.5.0, and the resulting program was executed on a PC featuring 16 Intel Xeon CPU E5520 at 2.27GHz, and 64GB of RAM running GNU/Linux 2.6.34.

The experiment consisted in computing the reversal distance 10 times for each pair  $(n, t)$ , where  $n$  is the size of the permutations, and  $t$  is the number of threads. Then, to determine the average speed gain, we calculated the average of the ratios  $\frac{T(t)}{T(1)}$  of each execution, where  $T(t)$  was the running time with  $t$  threads. The results are illustrated in Figure 2.1.

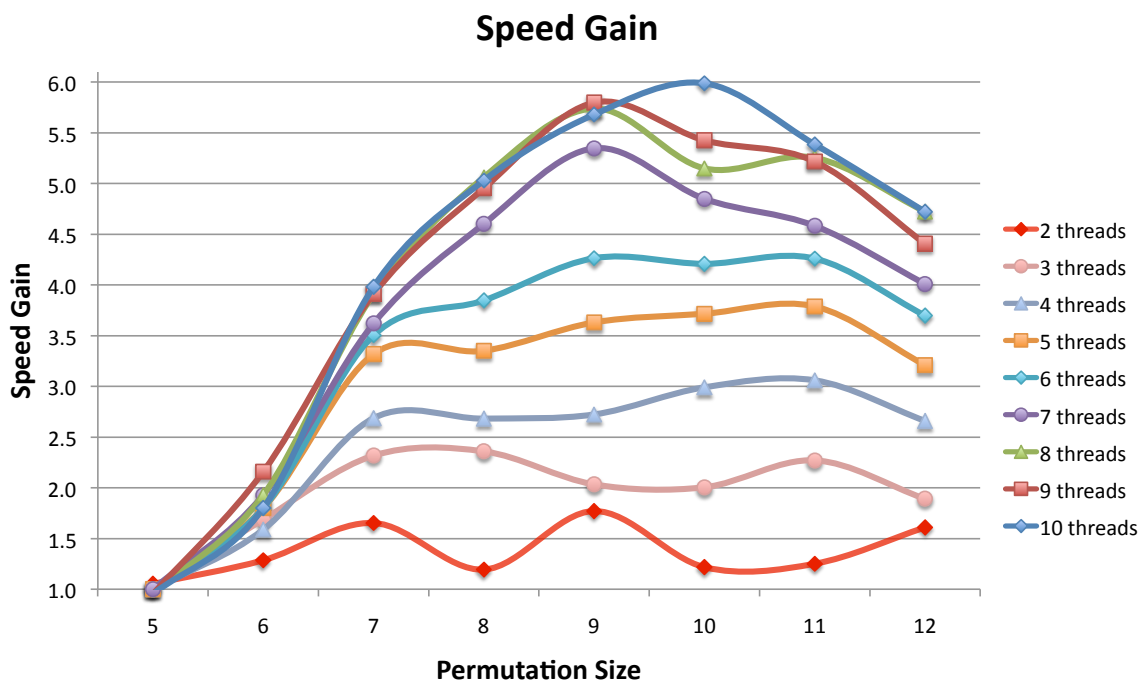


Figure 2.1: Average speed gain on the execution of the 32 bit implementation with multiple threads. The rearrangement model consisted of reversals only.

### 2.3.2 Computing Rearrangement Distances

Using the implementations described in the previous section, we have computed the rearrangement distances of all permutations in  $S_n$  and  $S_n^\pm$  with respect to 10 rear-

rearrangement models as summarized in Table 2.1. These rearrangement models give rise to variants of the rearrangement sorting problem for which the best known polynomial time solutions are approximations or heuristics, which is precisely the solutions GRAAu aims.

Table 2.1: Rearrangement models and values of  $n$  considered in the computation of the rearrangement distances.

Rearrangement Model	$n \leq$
Reversals	13
Prefix Reversals	13
Transpositions	13
Prefix Transpositions	13
Reversals and Transpositions	13
Prefix Reversals and Prefix Transpositions	13
Signed Prefix Reversals	10
Signed Reversals and Transpositions	10
Signed Reversals, Transpositions, and Transreversals (type A)	10
Transpositions and Transreversals (types A and B)	10

Note that we have computed the rearrangement distances of all unsigned permutations of sizes  $1 \leq n \leq 13$  and all signed permutations of sizes  $1 \leq n \leq 10$ . It was not possible to consider permutations with more elements due to memory constraints. Nevertheless, to the best of our knowledge, it is the first time that such computations were performed for these values of  $n$ .

### Distribution of Rearrangement Distances

Although our primary intention was to use the rearrangement distance data to build GRAAu, we have analyzed how these distances are distributed and have observed some possible patterns that might be of interest. Before presenting these patterns, we need to introduce some definitions and notation.

The greatest rearrangement distance of a permutation in  $S_n$  with respect to a rearrangement model  $M$  is said to be the diameter of  $S_n$ , and we denote it by  $D_M(n)$ . A slice of  $S_n$  with respect to a rearrangement model  $M$  is the subset  $S_n^i = \{\pi \mid d_M(\pi) = i \text{ and } \pi \in S_n\}$ ,  $0 \leq i \leq D_M(n)$ . The size of the largest slice of  $S_n$  with respect to rearrangement model  $M$  is said to be the traversal diameter of  $S_n$ , and it is denoted by  $T_M(n)$ . We establish a relation between  $D_M(n)$  and  $T_M(n)$  by defining a function  $L_M : \mathbb{N} \rightarrow \mathbb{N}$  such that  $D_M(n) = T_M(n) + L_M(n)$ . This function is said to be the longevity of  $S_n$  with respect to rearrangement model  $M$ . Note that the definitions of diameter, of slice, of traversal diameter, and of longevity also apply to  $S_n^\pm$ .

The problem of determining the diameter of  $S_n$  and  $S_n^\pm$  is also regarded in the genome rearrangement literature. Table 2.2 presents known results for the diameters

of  $S_n$  and  $S_n^\pm$  with respect to the rearrangement models listed in Table 2.1 (we have not considered rearrangement models for which no results were found). As we can see, the exact value of  $D_M(n)$  is not known for most of the rearrangement models we are considering. In light of this fact, some researchers have performed computations similar to the one we have and have conjectured some of these values. Therefore, analyzing the distribution of the rearrangement distance is a way of validating these conjectures.

Table 2.2: Exact values and bounds for the diameter.

Rearrangement Model (M)	$D_M(n) =$	$D_M(n) \geq$	$D_M(n) \leq$
Reversals	$n - 1$ [9]	–	–
Prefix Reversals	?	$\frac{15n}{14}$ [79]	$\frac{11n}{8} + O(1)$ [28]
Transpositions	?	$\frac{17n}{33} + \frac{1}{33}$ [99]	$\lfloor \frac{2n-2}{3} \rfloor$ [50]
Prefix Transpositions	?	$\lfloor \frac{3n}{4} \rfloor$ [90]	$n - \log_{\frac{9}{2}} n$ [29]
Signed Prefix Reversals	?	$\frac{3n}{2}$ [31]	$2n - 2$ [31]
Signed Reversals and Transpositions	?	$\lfloor \frac{n}{2} \rfloor + 2$ [101]	?

In addition validating conjectures, we have decided to propose measures  $T_M(n)$  and  $L_M(n)$  in an attempt to better characterize how the rearrangement distances are distributed in  $S_n$  and  $S_n^\pm$ . When we look at the distributions, we can note that the size of the slices of  $S_n$  and  $S_n^\pm$  monotonically increases until it reaches a peak, then it monotonically decreases. Thus,  $T_M(n)$  is the number of existing slices until that peak is reached, and  $L_M(n)$  is the number of existing slices afterward.

The distribution of rearrangement distances are given in tables 2.3 through 2.12, whereas the patterns (presented as conjectures) are given in Table 2.15. To facilitate the observation of the patterns, we have compiled the values of the diameter, of the traversal diameter, and of the longevity of  $S_n^\pm$  and  $S_n$  in tables 2.13 and 2.14, respectively.

Kececioglu and Sankoff [84] have presented the reversal distance distribution for  $n \leq 8$ . We note that the distribution obtained by us is different from theirs. For instance, we have found that  $|S_5^2| = 52$ ,  $|S_6^2| = 129$ ,  $|S_7^2| = 266$ , and  $|S_8^2| = 487$ , but they have found that  $|S_5^2| = 51$ ,  $|S_6^2| = 127$ ,  $|S_7^2| = 263$ , and  $|S_8^2| = 483$ . We have confirmed that our distribution is the right one by computing the reversal distances of the permutations in each one of these slices with GRIMM [120].

The conjecture of Dias and Meidanis [42] on the prefix transposition diameter (they have conjectured that  $D_M(n) = n - \lfloor \frac{n}{4} \rfloor$ ,  $n > 3$ ) holds for  $n \leq 13$ . This is an interesting result because Eriksson *et al.* [50] have shown that the first deviation of the transposition diameter from its original conjecture occurred when  $n = 13$ . Walter *et al.* [123] have demonstrated that  $\lfloor \frac{n}{2} \rfloor + 2$  is a lower bound for the signed reversal and transposition diameter, and have conjectured that it is an upper bound as well.









Table 2.8: Transposition distance distribution in  $S_n$ .

<b>d</b>	<b>n</b>											
	2	3	4	5	6	7	8	9	10	11	12	13
1	1	4	10	20	35	56	84	120	165	220	286	364
2		1	12	68	259	770	1932	4284	8646	16203	28600	48048
3			1	31	380	2700	13467	52512	170907	484440	1231230	2864719
4					45	1513	22000	191636	1183457	5706464	22822293	78829491
5							2836	114327	2010571	21171518	157499810	910047453
6									255053	12537954	265819779	3341572727
7											31599601	1893657570
8												427

Table 2.9: Reversals and transposition distance distribution in  $S_n$ .

<b>d</b>	<b>n</b>											
	2	3	4	5	6	7	8	9	10	11	12	13
1	1	5	13	26	45	71	105	148	201	265	341	430
2			10	89	408	1301	3331	7367	14672	27002	46716	76897
3				4	266	3467	24057	111767	396691	1167102	2993970	6919519
4						200	12826	233587	2321700	15036792	71584145	272688548
5								10010	895535	23229430	325121379	2887887456
6										456208	79255048	3046408308
7												13039641



Table 2.11: Prefix transposition distance distribution in  $S_n$ .

<b>d</b>	<b>n</b>											
	2	3	4	5	6	7	8	9	10	11	12	13
1	1	3	6	10	15	21	28	36	45	55	66	78
2		2	14	50	130	280	532	924	1500	2310	3410	4862
3			3	55	375	1575	4970	12978	29610	61050	116325	208065
4				4	194	2598	18096	85128	308988	933108	2456256	5812092
5					5	562	15532	188386	1364710	7030210	28488724	96641974
6						3	1161	74183	1679189	19713542	148968371	827628815
7								1244	244430	11759676	242448896	2832043750
8									327	416845	56288493	2323157040
9										3	231058	141492748
10												31375



Table 2.13: Diameter, traversal diameter, and longevity of  $S_n^\pm$ .

Table 2.3				Table 2.4			
n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)
2	4	3	1	2	2	1	1
3	6	4	2	3	3	2	1
4	8	5	3	4	4	3	1
5	10	6	4	5	4	3	1
6	12	8	4	6	5	4	1
7	14	9	5	7	6	4	2
8	15	10	5	8	6	5	1
9	17	11	6	9	7	5	2
10	18	13	5	10	8	6	2

Table 2.5				Table 2.6			
n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)
2	2	1	1	2	3	2	1
3	3	2	1	3	3	2	1
4	4	2	2	4	3	2	1
5	4	3	1	5	4	3	1
6	5	3	2	6	4	3	1
7	5	4	1	7	5	4	1
8	6	4	2	8	5	4	1
9	6	5	1	9	6	5	1
10	7	5	2	10	7	5	2

Table 2.14: Diameter, traversal diameter, and longevity of  $S_n$ .

Table 2.7				Table 2.8				Table 2.9			
n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)
2	1	1	0	2	1	1	0	2	1	1	0
3	2	1	1	3	2	1	1	3	1	1	0
4	3	2	1	4	3	2	1	4	2	1	1
5	4	3	1	5	3	2	1	5	3	2	1
6	5	3	2	6	4	3	1	6	3	2	1
7	6	4	2	7	4	3	1	7	4	3	1
8	7	5	2	8	5	4	1	8	4	3	1
9	8	5	3	9	5	4	1	9	5	4	1
10	9	6	3	10	6	5	1	10	5	4	1
11	10	7	3	11	6	5	1	11	6	5	1
12	11	7	4	12	7	6	1	12	6	5	1
13	12	8	4	13	8	6	2	13	7	6	1

Table 2.10				Table 2.11				Table 2.12			
n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)	n	D(n)	T(n)	L(n)
2	1	1	0	2	1	1	0	2	1	1	0
3	3	2	1	3	2	1	1	3	2	1	1
4	4	3	1	4	3	2	1	4	2	2	0
5	5	4	1	5	4	3	1	5	3	2	1
6	7	5	2	6	5	3	2	6	4	3	1
7	8	6	2	7	6	4	2	7	5	3	2
8	9	7	2	8	6	4	2	8	5	4	1
9	10	8	2	9	7	5	2	9	6	5	1
10	11	9	2	10	8	6	2	10	7	5	2
11	13	10	3	11	9	6	3	11	7	6	1
12	14	11	3	12	9	7	2	12	8	6	2
13	15	12	3	13	10	7	3	13	9	7	2

Table 2.15: Conjectures regarding  $D(n)$ ,  $T(n)$ , and  $L(n)$ .

Rearrangement Model (M)	$D_M(n)$	$T_M(n)$	$L_M(n)$	$n \geq$
Reversals	–	$\lceil \frac{2n}{3} \rceil - 1$	$\lfloor \frac{n}{3} \rfloor$	3
Prefix Reversals	–	$n - 1$	–	1
Transpositions	–	$\lfloor \frac{n}{2} \rfloor$	–	1
Prefix Transpositions	–	$n - \lceil \frac{2n}{5} \rceil$	$\lceil \frac{2n}{5} \rceil - \lfloor \frac{n}{4} \rfloor$	4
Signed Prefix Reversals	–	$\lfloor \frac{5n+2}{4} \rfloor$	–	1
Signed Reversals and Transpositions	$n - \lfloor \frac{n-2}{3} \rfloor$	$n - \lceil \frac{n-2}{2} \rceil$	$\lceil \frac{n-2}{2} \rceil - \lfloor \frac{n-2}{3} \rfloor$	3

### Rearrangement Distance Database

The rearrangement distances were stored in files indexed by permutations – that is, for each pair  $(n, M)$ , we have created a file containing the rearrangement distances of all permutations in  $S_n$  ( $S_n^\pm$ ) with respect to  $M$  such that the record in the position  $i$  of that file contains the rearrangement distance of the permutation  $f^{-1}(i)$  ( $g^{-1}(i)$ ). As the values of the rearrangement distances have not exceeded 255, each record has one-byte length. Overall, we have created 119 files totaling about 60GB of data.

We have designed a web interface to enable users access to the information contained in these files. The access address is

<http://mirza.ic.unicamp.br:8080>.

Using this interface, a user can do the following:

- Search the rearrangement distance of a permutation with respect to a rearrangement model;
- Search for permutations belonging to a slice of  $S_n$  ( $S_n^\pm$ ) with respect to a rearrangement model. The search result is limited to 200 permutations for efficiency reasons;
- Verify the distribution of the rearrangement distances in  $S_n$  ( $S_n^\pm$ ) with respect to a rearrangement model.

In addition to searching for the rearrangement distance of a permutation with respect to a rearrangement model, it is possible to view a sequence of permutations that illustrates the transformation of that permutation into the identity permutation. We refer to this sequence as solution. Note that Algorithm *AllDistances* presented in Section 2.3.1 does not compute the solution, so we had to modify that algorithm in order to compute it.

We created a vector  $P$  of size  $|S_n|$  such that  $P[i] = f(\pi)$ , where  $\pi$  is a parent permutation of permutation  $\sigma = f^{-1}(i)$  (i.e.  $\pi$  is the permutation such that  $\sigma = \pi \circ \rho$ ,

$\rho \in M$ , and  $d_M(\sigma) = d_M(\pi) + 1 = D[i]$ . Then, we have made vector  $P$  be updated every time a new permutation was inserted in vector  $Q$  and its rearrangement distance was stored in vector  $D$ , which corresponds to the for loop of lines 13 through 20. When Algorithm *AllDistances* terminates, in addition to returning vector  $D$ , it also returns vector  $P$ . With this vector in hand, it is not hard to see how to obtain the solution recursively.

As well as the rearrangement distances, the parent permutations were also stored in files indexed by permutations – that is, for each pair  $(n, M)$ , we created a file containing the parent permutations of all permutations in  $S_n$  ( $S_n^\pm$ ) with respect to  $M$  such that the record in the position  $i$  of that file contains the parent permutation of the permutation  $f^{-1}(i)$  ( $g^{-1}(i)$ ). Differently from the rearrangement distances, which only need 1 byte per record, parent permutations need 4 bytes in the case of the 32 bits implementation, and 8 bytes in the case of the 64 bits implementation. It makes the files too big and, for this reason, we did not compute the parent permutations of the permutations in  $S_{13}$ . Overall, we have created 112 files totaling about 72GB of data.

We created this web interface to give users the ability to extract information of interest regarding the rearrangement distances. For instance, Grusea and Labarre have used the information about the distribution of the rearrangement distances in a recent work [67]. As another example, for those who are interested in proving exact values or bounds for the diameter, it may be useful to know which permutations belong to the slice  $S_n^i$  such that  $i = D_M(n)$ .

### 2.3.3 Implementation of GRAAu

The audit performed by GRAAu is similar to the method adopted in previous works for analyzing approximation algorithms and heuristics for genome rearrangements [14, 38, 121, 124, 125]. It consists in comparing, for all permutations of up to a given size, the distance output by a given genome rearrangement algorithm with the related rearrangement distance and then producing statistics that can be used to analyze the performance of this algorithm. The statistics produced by GRAAu are as follows:

- **Diameter:** Greatest distance output by the algorithm.
- **Average Distance:** Average of the distances output by the algorithm.
- **Average Ratio:** Average of the ratios between the distance output by the algorithm and the related rearrangement distance.
- **Maximum Ratio:** Greatest ratio among all the ratios between the distance output by the algorithm and the related rearrangement distance.
- **Equals:** Percentage of distances output by the algorithm that is equal to the related rearrangement distance.



In addition to the statistics, GRAAu outputs up to 50 permutations exhibiting the maximum ratio.

GRAAu was implemented as a client-server application, and hence it is composed of two components: a server that stores the rearrangement distances (such as described in Section 2.3.2) and the audit results (i.e. the statistics and the permutations which exhibited the maximum ratio); and a client that executes the rearrangement algorithm (which must be implemented by the user), compares the output distances with the rearrangement distances (obtained from the server), and reports the results back to the server.

Both the client and the server were implemented in Java. The communication between them is achieved through standard web service messaging over HTTP (we have adopted Apache Axis2 1.5.2 engine to support the implementation and the deployment of the web services). The server is installed on a machine featuring a Intel Core i7-2600K, which has four cores at 3.4 GHz each, and 16GB of RAM, and running Apache Tomcat 6.0.26 on GNU/Linux 2.6.32. Compiled code, setup instructions, and tutorial are available for download at

<http://mirza.ic.unicamp.br:8080/bioinfo/graaui.jsf>.

GRAAu does not require registration by the user. In the beginning of the auditing process of a rearrangement algorithm, the web server generates a unique ID to this algorithm, a random 256-bit AES key, and a counter, and sends them to the client using RSA. Then, for each web service call that changes the audit state of this algorithm, the client sends its ID and the encrypted value of the counter along with the other parameters to allow the web server to authenticate the call. After such calls, the client and the web server increment the counter. The AES key and the algorithm ID are recorded in a file in the client computer so that the audit can be resumed after any interruption.

The audit results and audit progress information for each rearrangement algorithm being audited by GRAAu become available online. In addition to these, there is other information available, namely the name of the rearrangement algorithm, the rearrangement model that it considers, and its description. The name and the description of an algorithm can be edited; the audit results, and even all the information about an algorithm, can be deleted. To perform any of these operations, the user has to enter a 128-bit password exchanged between the client and the web server using RSA and recorded in the same file as the AES key and the algorithm ID.

The time needed to complete the audit of a rearrangement algorithm is not easily predictable because it depends on many factors, such as the complexity of the algorithm, number of threads chosen, Internet bandwidth, and CPU speed. For instance, we implemented Watterson *et. al.* [126] algorithm and audited it with GRAAu using a computer featuring a Intel Core 2 Duo CPU at 2.20 GHz and a broadband Internet connection (4Mbps). The audit took less than 4 hours to complete. On the other hand, it took about 72 hours to audit a naive implementation of the algorithm of

Kececioglu and Sankoff algorithm [84] using the same configuration.

## 2.4 Application of GRAAu

In this section, we present some results that we have obtained by using GRAAu to evaluate approximation algorithms for two variants of the rearrangement sorting problem, namely the problem of sorting by prefix reversals and the problem of sorting by prefix transpositions.

### 2.4.1 Sorting by Prefix Reversals

In this section, we present the results that we have obtained from the audit of two versions of the 2-approximation algorithm developed by Fischer and Ginzinger [55] for the problem of sorting by prefix reversals. As we have shown in Section 2.2.3, this is the best known approximation algorithm for this problem. Before we present the results, we provide a brief description of the theory underlying their algorithm.

Given a permutation  $\pi$  in  $S_n$ , we extend it with two elements  $\pi_0 = 0$  and  $\pi_{n+1} = n + 1$ . The extended permutation is still denoted by  $\pi$ . The prefix reversal distance of  $\pi$  is denoted by  $d_{pr}(\pi)$ . A breakpoint in  $\pi$  is a pair of adjacent elements  $(\pi_i, \pi_{i+1})$  such that  $|\pi_i - \pi_{i+1}| \neq 1$ ,  $1 \leq i \leq n$ . Note that the pair  $(\pi_0, \pi_1)$  is not considered a breakpoint. The number of breakpoints in  $\pi$  is denoted by  $b_{pr}(\pi)$ .

**Example 1.** Let  $\pi = (0 \ 1 \ 3 \ 2 \ 4 \ 5 \ 6)$  be an extended permutation. Then, we have that the pairs of adjacent elements  $(1, 3)$  and  $(2, 4)$  are breakpoints, therefore  $b_{pr}(\pi) = 2$ .

Note that  $b_{pr}(\pi) = 0$  if and only if  $\pi = \iota$ . Since a prefix reversal can remove at most one breakpoint of an unsigned permutation, the following lemma holds.

**Lemma 1** (Fischer and Ginzinger [55]). *For any unsigned permutation  $\pi$ , we have that  $d_{pr}(\pi) \geq b_{pr}(\pi)$ .*

A strip of  $\pi$  is a subsequence of contiguous elements  $\pi_i \pi_{i+1} \dots \pi_j$ ,  $1 \leq i \leq j \leq n$ , such that  $(\pi_{i-1}, \pi_i)$  and  $(\pi_j, \pi_{j+1})$  are breakpoints, and none of the pairs  $(\pi_k, \pi_{k+1})$ ,  $i \leq k \leq j - 1$ , is a breakpoint. A strip with more than one element is called decreasing if  $\pi_i > \pi_{i+1} > \dots > \pi_j$ , otherwise it is called increasing.

**Example 2.** Let  $\pi$  be the permutation of Example 1. Then, we have that  $\underline{1}$ ,  $\underline{3 \ 2}$ , and  $\underline{4 \ 5}$  are strips of  $\pi$ . In addition, the second one is an decreasing strip and the last one is an increasing strip.

Unless stated otherwise, in the rest of this section we will only consider permutations where the pair of elements  $(\pi_n, \pi_{n+1})$  is a breakpoint. This is simply because if there are  $m$  ordered elements at the end of  $\pi$ , say  $\pi = (\pi_1, \dots, \pi_{n-m}, n - m + 1, \dots, n - 1, n)$ , we can reduce the problem of sorting  $\pi$  to sorting the permutation  $\sigma = (\pi_1, \dots, \pi_{n-m})$ .

The breakpoint graph  $G(\pi) = (V, E)$  of a permutation  $\pi \in S_n$  is a directed graph whose vertex set is composed by the elements of  $\pi$  – that is,  $V = \{\pi_0, \pi_1, \dots, \pi_{n+1}\}$  – and whose edge set  $E$  is composed by so-called red and blue edges, defined as follows: an edge  $e$  is red if  $e = (\pi_i, \pi_{i+1})$  and position  $i$  is a breakpoint, or  $e = (\pi_0, \pi_1)$  and  $|\pi_0 - \pi_1| \neq 1$ ; an edge  $e = (\pi_i, \pi_j)$ ,  $1 \leq i < j \leq n + 1$ , is blue if  $\pi_j = \pi_i \pm 1$  and  $i < j - 1$ . Figure 2.2 illustrates the breakpoint graph of permutation  $(4\ 2\ 1\ 3)$ .

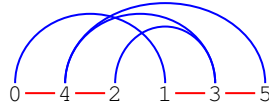


Figure 2.2: Breakpoint graph  $G(\pi)$  of permutation  $\pi = (4\ 2\ 1\ 3)$ .

Let  $\pi$  be an unsigned permutation and  $e = (\pi_i, \pi_j)$  be a blue edge of  $G(\pi)$ . Since there is at least one adjacent red edge on each side of  $e$ , we can classify it into at least one of the following types:

- type 1, if  $(\pi_{i-1}, \pi_i)$  and  $(\pi_{j-1}, \pi_j)$  are red edges;
- type 2, if  $(\pi_i, \pi_{i+1})$  and  $(\pi_j, \pi_{j+1})$  are red edges;
- type 3, if  $(\pi_i, \pi_{i+1})$  and  $(\pi_{j-1}, \pi_j)$  are red edges;
- or type 4, if  $(\pi_{i-1}, \pi_i)$  and  $(\pi_j, \pi_{j+1})$  are red edges.

Moreover, a blue edge  $e = (\pi_i, \pi_j)$  is said to be a good edge if at least one of the three conditions holds:

1.  $e$  is of type 1 with  $i = 1$ ;
2.  $e$  is of type 2 with  $i \neq 0$ ;
3.  $e$  is of type 3.

Fischer and Ginzinger [55] have shown that if the breakpoint graph of a permutation  $\pi \in S_n$  has a good blue edge, then it is possible to remove a breakpoint with at most two prefix reversals, such as described next:

- if the blue edge  $e = (\pi_i, \pi_j)$  satisfies condition (1), then  $(\pi_{j-1}, \pi_j)$  is a breakpoint because  $e$  is of type 1, and the prefix reversal  $pr(j - 1)$  removes it since  $\sigma_{j-1} = \pi_i$  and  $\sigma_j = \pi_j$  in the permutation  $\sigma = \pi \circ pr(j - 1)$ ;
- if the blue edge  $e = (\pi_i, \pi_j)$  satisfies condition (2), we can apply the prefix reversal  $pr(j)$  on  $\pi$ , yielding the permutation  $\sigma = \pi \circ pr(j)$ . Note that  $j < n + 1$  since  $e$  is of type 2; therefore the prefix reversal  $pr(j)$  is valid. The breakpoint graph  $G(\sigma)$  contains the blue edge  $e' = (\sigma_1, \sigma_{j-i+1})$  because  $\sigma_1 = \pi_j$  and  $\sigma_{j-i+1}$

$= \pi_i$ ; moreover,  $(\sigma_{j-i}, \sigma_{j-i+1})$  is a breakpoint because  $\sigma_{j-i} = \pi_i$  and  $\sigma_{j-i+1} = \pi_{i+1}$ , and the prefix reversal  $pr(j-i)$  removes it since  $\gamma_{j-i} = \sigma_1$  and  $\gamma_{j-i+1} = \sigma_{j-i+1}$  in the permutation  $\gamma = \sigma \circ pr(j-i)$ ;

- if the blue edge  $e = (\pi_i, \pi_j)$  satisfies condition (3), we can apply the prefix reversal  $pr(i)$  on  $\pi$ , yielding the permutation  $\sigma = \pi \circ pr(i)$ . The breakpoint graph  $G(\sigma)$  contains the blue edge  $e' = (\sigma_1, \sigma_j)$  because  $\sigma_1 = \pi_i$  and  $\sigma_j = \pi_j$ ; moreover,  $(\sigma_{j-1}, \sigma_j)$  is a breakpoint because  $\sigma_{j-1} = \pi_{j-1}$  and  $\sigma_j = \pi_j$ , and the prefix reversal  $pr(j-1)$  removes it since  $\gamma_{j-1} = \sigma_1$  and  $\gamma_j = \sigma_j$  in the permutation  $\gamma = \sigma \circ pr(j-1)$ . Note that if  $i = 1$ , then we do not need to apply the prefix reversal  $pr(i)$ .

If the breakpoint graph of a permutation  $\pi \in S_n$ ,  $\pi \neq \iota$ , does not have a good blue edge, then Fischer and Ginzinger [55] have proved that  $\pi$  is of the form

$$\pi = \underbrace{(p_1 \dots 1)}_{l_1} \underbrace{(p_2 \dots p_1 + 1)}_{l_2} \dots \underbrace{(l \dots p_{b_{pr}(\pi)-1} + 1)}_{l_{b_{pr}(\pi)}}.$$

In other words,  $\pi$  consists of  $b_{pr}(\pi) \geq 2$  decreasing strips of length  $l_i$  for all  $1 \leq i \leq b_{pr}(\pi)$ . In this case, they have demonstrated that the sequence of  $2b_{pr}(\pi)$  prefix reversals

$$pr(n), pr(n - l_1), pr(n), pr(n - l_2), \dots, pr(n), pr(n - l_{b_{pr}(\pi)})$$

sorts  $\pi$ .

Therefore, it is possible to sort any unsigned permutation  $\pi$  applying no more than  $2b_{pr}(\pi)$  prefix reversals using the following strategy. While  $G(\pi)$  contains good blue edges, we choose one of them and apply at most two prefix reversals to remove a breakpoint; if it is not the case, the permutation will have the form described previously, so we can sort it applying  $2b_{pr}(\pi)$  prefix reversals. Since  $d_{pr}(\pi) \geq b_{pr}(\pi)$  (Lemma 1), we have that any algorithm using such strategy is a 2-approximation.

Fischer and Ginzinger [55] have not specified an algorithm that implements this strategy. They have only said that good blue edges that satisfy condition (1) were preferred over good blue edges that satisfy conditions (2) and (3) because the former just need one prefix reversal to remove a breakpoint instead of two. For this reason, we have considered two algorithms: one that favors good blue edges satisfying condition (2) over good blue edges satisfying condition (3) (Algorithm 2), and another one that favors the opposite (Algorithm 3). We have audited algorithms 2 and 3 with GRAAu, and the results are presented in tables 2.16 and 2.17 respectively. As we can see later in Figure 2.3, both algorithms exhibited a very similar performance, with a small advantage to Algorithm 3.

Fischer and Ginzinger [55] have conducted a different experimental investigation on the performance of their algorithm. They have computed the prefix reversal distance of 10,000 random permutations of length up to 71 using a branch-and-bound method

**Algorithm 2:** 2-approximation algorithm for sorting by prefix reversals**Data:** A permutation  $\pi \in S_n$ .**Result:** Number of prefix reversals applied to sort  $\pi$ .

---

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq \iota$  do
3   if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (1) then
4      $\pi \leftarrow \pi \circ pr(j - 1)$ ;
5      $d \leftarrow d + 1$ ;
6   else if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (2) then
7      $\pi \leftarrow \pi \circ pr(j)$ ;
8      $\pi \leftarrow \pi \circ pr(j - i)$ ;
9      $d \leftarrow d + 2$ ;
10  else if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (3) then
11     $\pi \leftarrow \pi \circ pr(i)$ ;
12     $\pi \leftarrow \pi \circ pr(j - 1)$ ;
13     $d \leftarrow d + 2$ ;
14  else
15    Let  $l$  be the number of elements of  $\pi$  that are not ordered elements at
    the end of  $\pi$ , and let  $l_i$  be the length of the  $i$ -th strip of  $\pi$ ;
16     $\pi \leftarrow \pi \circ pr(l) \circ pr(l - l_1) \circ pr(l) \circ pr(l - l_2) \circ \dots \circ pr(l) \circ pr(l -$ 
     $l_{b_{pr}(\pi)})$ ;
17     $d \leftarrow d + 2b_{pr}(\pi)$ ;
18  end
19 end
20 return  $d$ ;

```

---

Table 2.16: Results from the audit of Algorithm 2

n	Diameter	Avg. Distance	Avg. Ratio	Max. Ratio	Equals
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	3	1.50	1.00	1.00	100.00%
4	5	2.79	1.09	1.33	70.83%
5	8	4.01	1.12	1.75	62.50%
6	9	5.26	1.14	1.80	51.25%
7	12	6.50	1.15	1.83	42.54%
8	14	7.74	1.16	1.83	34.51%
9	15	8.98	1.17	2.00	27.75%
10	17	10.21	1.17	2.00	22.17%
11	19	11.45	1.18	2.00	17.63%
12	21	12.68	1.18	2.00	13.98%
13	23	13.91	1.19	2.00	11.07%

**Algorithm 3:** 2-approximation algorithm for sorting by prefix reversals**Data:** A permutation  $\pi \in S_n$ .**Result:** Number of prefix reversals applied to sort  $\pi$ .

---

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq \iota$  do
3   if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (1) then
4      $\pi \leftarrow \pi \circ pr(j - 1)$ ;
5      $d \leftarrow d + 1$ ;
6   else if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (3) then
7      $\pi \leftarrow \pi \circ pr(i)$ ;
8      $\pi \leftarrow \pi \circ pr(j - 1)$ ;
9      $d \leftarrow d + 2$ ;
10  else if  $G(\pi)$  contains a good blue edge  $(\pi_i, \pi_j)$  satisfying condition (2) then
11     $\pi \leftarrow \pi \circ pr(j)$ ;
12     $\pi \leftarrow \pi \circ pr(j - i)$ ;
13     $d \leftarrow d + 2$ ;
14  else
15    Let  $l$  be the number of elements of  $\pi$  that are not ordered elements at
    the end of  $\pi$ , and let  $l_i$  be the length of the  $i$ -th strip of  $\pi$ ;
16     $\pi \leftarrow \pi \circ pr(l) \circ pr(l - l_1) \circ pr(l) \circ pr(l - l_2) \circ \dots \circ pr(l) \circ pr(l -$ 
     $l_{b_{pr}(\pi)})$ ;
17     $d \leftarrow d + 2b_{pr}(\pi)$ ;
18  end
19 end
20 return  $d$ ;

```

---

Table 2.17: Results from the audit of Algorithm 3

n	Diameter	Avg. Distance	Avg. Ratio	Max. Ratio	Equals
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	3	1.50	1.00	1.00	100.00%
4	5	2.71	1.06	1.33	79.17%
5	7	3.93	1.10	1.75	69.17%
6	9	5.18	1.12	1.75	57.36%
7	11	6.44	1.14	1.75	47.66%
8	13	7.68	1.15	1.83	38.94%
9	15	8.93	1.16	1.86	31.61%
10	17	10.17	1.17	1.89	25.52%
11	19	11.41	1.18	1.90	20.54%
12	21	12.65	1.18	1.91	16.48%
13	23	13.89	1.19	1.91	13.20%

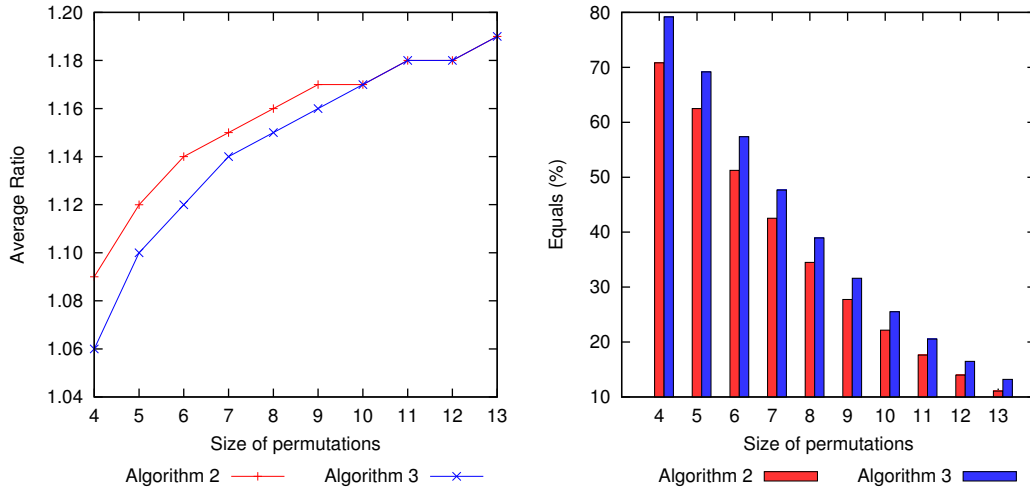


Figure 2.3: Performance comparison between algorithms 2 and 3 based on the results provided by GRAAu.

and have compared them with the distance computed by the algorithm. The results they have obtained led them to believe that “with a deeper analysis of the algorithm the theoretical approximation ratio could even be lowered”. Our results point to an opposite direction.

In the case of Algorithm 2, the maximum ratio has matched the theoretical approximation ratio, which is equal to 2, for  $9 \leq n \leq 13$ . This is sufficient for proving that the approximation ratio of Algorithm 2 is tight (Theorem 1). In the case of Algorithm 3, the maximum ratio has not matched the theoretical approximation ratio, but it seems to be converging to it. Therefore, we conjecture that the approximation ratio of Algorithm 3 is tight (Conjecture 1).

**Theorem 1.** *The approximation ratio of Algorithm 2 is tight.*

*Proof.* Let  $\pi = (1\ 7\ 8\ 2\ 4\ 3\ 9\ 5\ 6)$  be an unsigned permutation. Since  $d_{pr}(\pi) \geq b_{pr}(\pi) = 6$  and the sequence of prefix reversals  $pr(3)$ ,  $pr(6)$ ,  $pr(2)$ ,  $pr(7)$ ,  $pr(9)$ ,  $pr(6)$  sorts  $\pi$ , we have that  $d_{pr}(\pi) = 6$ . On the other hand, Algorithm 2 sorts  $\pi$  as follows:

1. As we can see in Figure 2.4(a),  $G(\pi)$  does not contain a blue edge satisfying condition (1), but it contains a blue edge satisfying condition (2), namely  $(\pi_1, \pi_4)$ . Therefore, Algorithm 2 applies the prefix reversal  $pr(4)$  followed by the prefix reversal  $pr(3)$ , yielding the permutation  $\pi = (7\ 8\ 2\ 1\ 4\ 3\ 9\ 5\ 6)$ ;
2. As we can see in Figure 2.4(b),  $G(\pi)$  does not contain a blue edge satisfying condition (1), but it contains a blue edge satisfying condition (2), namely  $(\pi_2, \pi_7)$ . Therefore, Algorithm 2 applies the prefix reversal  $pr(7)$  followed by the prefix reversal  $pr(5)$ , yielding the permutation  $\pi = (2\ 1\ 4\ 3\ 9\ 8\ 7\ 5\ 6)$ ;
3. As we can see in Figure 2.4(c),  $G(\pi)$  does not contain a blue edge satisfying condition (1), but it contains a blue edge satisfying condition (2), namely  $(\pi_7,$

$\pi_9$ ). Therefore, Algorithm 2 applies the prefix reversal  $pr(9)$  followed by the prefix reversal  $pr(2)$ , yielding the permutation  $\pi = (5\ 6\ 7\ 8\ 9\ 3\ 4\ 1\ 2)$ ;

4. As we can see in Figure 2.4(d),  $G(\pi)$  does not contain a blue edge satisfying condition (1) neither a blue edge satisfying condition (2), but it contains a blue edge satisfying condition (3), namely  $(\pi_5, \pi_{10})$ . Therefore, Algorithm 2 applies the prefix reversal  $pr(5)$  followed by the prefix reversal  $pr(9)$ , yielding the permutation  $\pi = (2\ 1\ 4\ 3\ 5\ 6\ 7\ 8\ 9)$ ;
5. As we can see in Figure 2.4(e),  $G(\pi)$  does not contain a good blue edge, so let  $\sigma = (\pi_1\ \pi_2\ \pi_3\ \pi_4) = (2\ 1\ 4\ 3)$  be the permutation formed by the elements of  $\pi$  that are out of position (note that the elements  $\pi_5, \pi_6, \pi_7, \pi_8,$  and  $\pi_9$  are ordered elements at the end of  $\pi$ ). We have that Algorithm 2 sorts  $\pi$  applying the sequence of  $2b_{pr}(\sigma) = 4$  prefix reversals  $pr(4), pr(2), pr(4),$  and  $pr(2)$ .

Thus, denoting by  $A_2(\pi)$  the number of prefix reversals applied by Algorithm 2 for sorting  $\pi$ , we have that  $A_2(\pi) = 12$ .

Let  $\gamma = (1\ 7\ 8\ 2\ 4\ 3\ 9\ 5\ 6\ 10\ 11\ \dots\ n)$ ,  $n \geq 10$ , be an unsigned permutation. Since the elements  $\gamma_i$ ,  $10 \leq i \leq n$ , are in the right position, Algorithm 2 will sort  $\gamma$  the same way it sorts  $\pi$ ; moreover, we have that  $d_{pr}(\gamma) = d_{pr}(\pi) = 6$ . Therefore,  $\frac{A_2(\gamma)}{d_{pr}(\gamma)} = \frac{12}{6} = 2$ .  $\square$

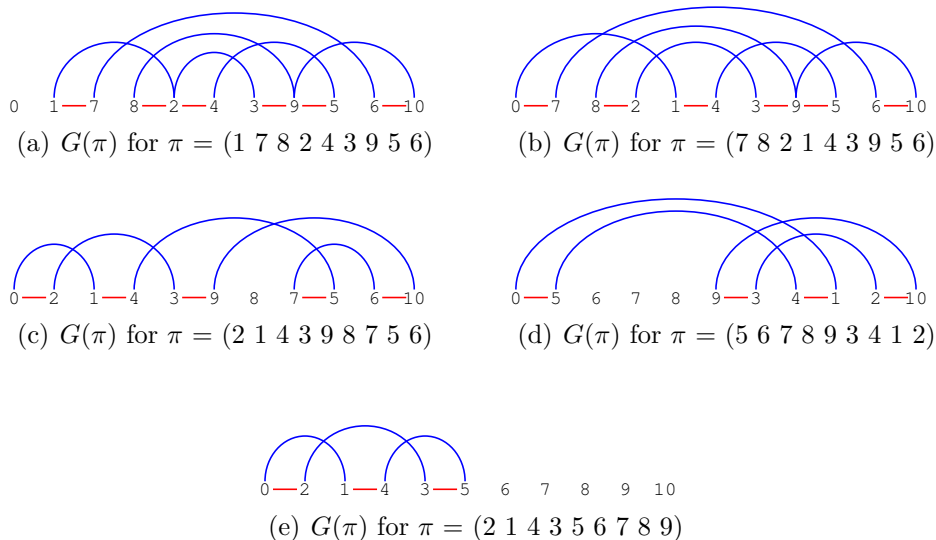


Figure 2.4: Breakpoint graphs of the permutations produced by Algorithm 2 when sorting permutation  $\pi = (1\ 7\ 8\ 2\ 4\ 3\ 9\ 5\ 6)$ .

**Conjecture 1.** *The approximation ratio of Algorithm 3 is tight.*



### 2.4.2 Sorting by Prefix Transpositions

In this section, we present the results that we have obtained from the audit of two approximation algorithms for the problem of sorting by prefix transpositions. The 2-approximation algorithm proposed by Dias and Meidanis [42]; it is the best known approximation algorithm for this problem and an improved version of this algorithm proposed by us. As we did in the previous section, before we present the results, we provide a brief description of the theory underlying these algorithms.

Given a permutation  $\pi$  in  $S_n$ , we extend it with two elements  $\pi_0 = 0$  and  $\pi_{n+1} = n + 1$ . The extended permutation is still denoted by  $\pi$ . The prefix transposition distance of  $\pi$  is denoted by  $d_{pt}(\pi)$ . A breakpoint in  $\pi$  is a pair of adjacent elements  $(\pi_i, \pi_{i+1})$  such that  $\pi_{i+1} - \pi_i \neq 1$ ,  $1 \leq i \leq n$ . Note that the pair  $(\pi_0, \pi_1)$  is not considered a breakpoint. The number of breakpoints in  $\pi$  is denoted by  $b_{pt}(\pi)$ .

**Example 3.** Let  $\pi = (0 \ 1 \ 3 \ 4 \ 2 \ 5 \ 6)$  be an extended permutation. Then, we have that the pairs  $(1, 3)$ ,  $(4, 2)$ , and  $(2, 5)$  are breakpoints; therefore  $b_{pt}(\pi) = 3$ .

Note that  $b_{pt}(\pi) = 0$  if and only if  $\pi = \iota$ . Since a prefix transposition can remove at most two breakpoints of an unsigned permutation, the following lemma holds.

**Lemma 2** (Dias and Meidanis [42]). *For any unsigned permutation  $\pi$ , we have that  $d_{pt}(\pi) \geq \frac{b_{pt}(\pi)}{2}$ .*

A strip of  $\pi$  is a subsequence of contiguous elements  $\pi_i \pi_{i+1} \dots \pi_j$ ,  $1 \leq i \leq j \leq n$ , such that  $(\pi_{i-1}, \pi_i)$  and  $(\pi_j, \pi_{j+1})$  are breakpoints, and none of the pairs  $(\pi_k, \pi_{k+1})$ ,  $i \leq k \leq j - 1$ , is a breakpoint.

**Example 4.** Let  $\pi$  be the permutation of Example 3. Then, we have that  $\underline{1}$ ,  $\underline{3 \ 4}$ ,  $\underline{2}$ , and  $\underline{5}$  are strips of  $\pi$ .

Dias and Meidanis [42] have shown that it is always possible to remove at least one breakpoint of a unsigned permutation  $\pi$  by applying a prefix transposition. Let  $\pi \in S_n$  and let  $\pi_1 \dots \pi_i$  be the first strip of  $\pi$ . If  $\pi_i < n$ , then there exists a strip of  $\pi$  that begins with the element  $\pi_j = \pi_i + 1$  such that  $i < j - 1$  and  $(\pi_{j-1}, \pi_j)$  is a breakpoint. In this case, the prefix transposition  $pt(i + 1, j)$  removes that breakpoint. Otherwise, if  $\pi_i = n$ , then the prefix transposition  $pt(i + 1, n + 1)$  removes the breakpoint  $(\pi_n, \pi_{n+1})$ . Algorithm 4 is the algorithm derived from this analysis. Since it removes at least one breakpoint for each prefix transposition it applies and  $d_{pt}(\pi) \geq \frac{b_{pt}(\pi)}{2}$  (Lemma 2), we have that Algorithm 4 is a 2-approximation.

Dias and Meidanis [42] have noted that a prefix transposition can eliminate two breakpoints and proved that there exists at most one such prefix transposition. Let  $\pi \in S_n$  and let  $pt(i, j)$  be the prefix transposition that removes two breakpoints of  $\pi$ . We have that  $\pi \circ pt(i, j) = (\pi_i \dots \pi_{j-1} \pi_1 \dots \pi_{i-1} \pi_j \dots \pi_n)$ , where  $\pi_{i-1} \neq \pi_i - 1$ ,  $\pi_{j-1} \neq \pi_j - 1$ ,  $\pi_{j-1} = \pi_1 - 1$ , and  $\pi_{i-1} = \pi_j - 1$ . Therefore,  $\pi_1$  determines uniquely the index  $j$ , and  $j$  determines uniquely the index  $i$ . It means that we can

make Algorithm 4 “greedier” – that is, before applying the prefix transposition which removes one breakpoint, we can verify if it is possible to apply the prefix transposition which removes two. Such an algorithm is presented in Algorithm 5. It is easy to see that this algorithm is a 2-approximation as well.

---

**Algorithm 4:** 2-approximation algorithm for sorting by prefix transpositions

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of prefix transpositions applied to sort  $\pi$

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq \iota$  do
3   Let  $\pi_i$  be the last element of the first strip of  $\pi$ ;
4   if  $\pi_i = n$  then
5      $\pi \leftarrow \pi \circ pt(i + 1, n + 1)$ ;
6      $d \leftarrow d + 1$ ;
7   else
8     Let  $\pi_j$  be the element of  $\pi$  such that  $\pi_j = \pi_i + 1$ ;
9      $\pi \leftarrow \pi \circ pt(i + 1, j)$ ;
10     $d \leftarrow d + 1$ ;
11  end
12 end
13 return  $d$ ;
```

---



---

**Algorithm 5:** Improved 2-approximation algorithm for sorting by prefix transpositions

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of prefix transpositions applied to sort  $\pi$

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq \iota$  do
3   Let  $i$  be the position of  $\pi_1 - 1$  in  $\pi$ ;
4    $y \leftarrow i + 1$ ;
5   Let  $j$  be the position of  $\pi_y - 1$  in  $\pi$ ;
6    $x \leftarrow j + 1$ ;
7   if  $x > 1$  and  $x < y$  then
8      $\pi \leftarrow \pi \circ pt(x, y)$ ;
9      $d \leftarrow d + 1$ ;
10  else
11    Apply the same steps as shown in lines 3–11 of Algorithm 4;
12  end
13 end
14 return  $d$ ;
```

---

We have audited algorithms 4 and 5 with GRAAu, and the results are presented in tables 2.18 and 2.19, respectively. As we can see in Figure 2.5, Algorithm 5 presented

much better results than Algorithm 4; however, when it comes to the maximum ratio obtained for both algorithms, the results suggest that it is converging to the theoretical approximation ratio, which is equal to 2. Before proving that the maximum ratio of these algorithms indeed converges to 2, we must introduce some concepts and notation.

Table 2.18: Results obtained from the audit of Algorithm 4.

<b>n</b>	<b>Diameter</b>	<b>Avg. Distance</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Equals</b>
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	2	1.17	1.00	1.00	100.00%
4	3	1.92	1.06	1.50	87.50%
5	4	2.72	1.12	1.50	70.83%
6	5	3.55	1.16	1.67	54.72%
7	6	4.41	1.20	1.67	39.60%
8	7	5.28	1.24	1.75	26.92%
9	8	6.17	1.27	1.75	17.33%
10	9	7.07	1.29	1.80	10.55%
11	10	7.98	1.32	1.80	6.07%
12	11	8.90	1.34	1.83	3.32%
13	12	9.82	1.36	1.83	1.73%

Table 2.19: Results obtained from the audit of Algorithm 5

<b>n</b>	<b>Diameter</b>	<b>Avg. Distance</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Equals</b>
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	2	1.17	1.00	1.00	100.00%
4	3	1.79	1.00	1.00	100.00%
5	4	2.45	1.01	1.33	97.50%
6	5	3.10	1.01	1.33	95.28%
7	6	3.77	1.02	1.50	91.11%
8	7	4.43	1.03	1.50	86.61%
9	8	5.10	1.04	1.60	81.31%
10	9	5.77	1.05	1.60	75.55%
11	10	6.44	1.06	1.67	69.64%
12	11	7.12	1.07	1.67	63.56%
13	12	7.79	1.07	1.71	57.58%

We say that a permutation is reduced if it only contains strips of length 1. We can reduce any permutation that contains one or more strips of length  $> 1$  as follows.

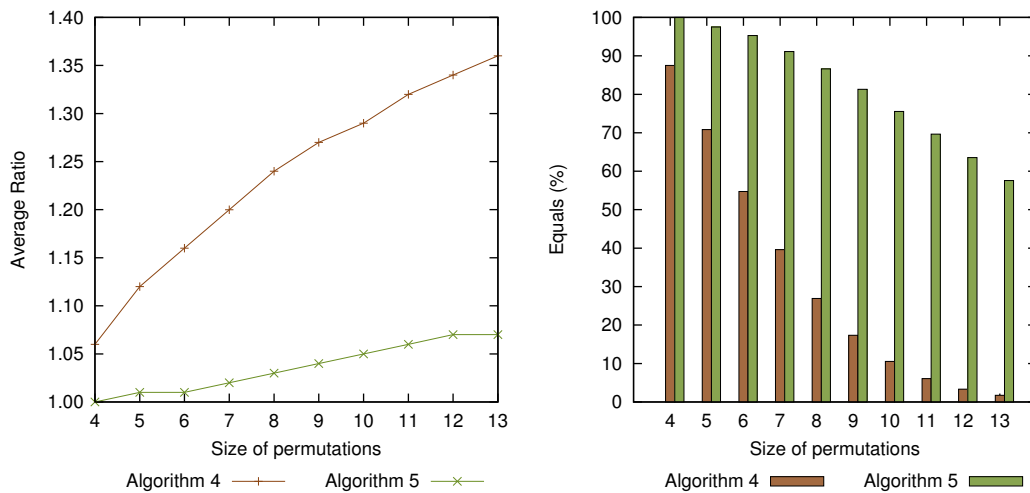


Figure 2.5: Performance comparison between algorithms 4 and 5 based on the results provided by GRAAu.

Discard the right-most strip if it ends with  $n$ , then keep the minimal element of each strip of  $\pi$ , and finally renumber the remaining elements appropriately. For example, the reduced permutation corresponding to  $(\underline{1} \ 2 \ \underline{4} \ 5 \ \underline{3} \ \underline{6})$ , in which strips are underlined, is  $(\underline{1} \ \underline{3} \ \underline{2})$ , through discarding the strip  $\underline{6}$  and replacing strips  $\underline{1} \ 2$ ,  $\underline{4} \ 5$ , and  $\underline{3}$  by  $\underline{1}$ ,  $\underline{3}$ , and  $\underline{2}$  respectively. Lemmas 3 and 4 allow us to restrict our attention to sorting reduced permutations. With regard to Lemma 3, we remark that Christie [30] has proved an analogous result for the problem of sorting by transpositions, and our proof is a direct adaptation of that provided by him.

**Lemma 3.** *If a permutation  $\pi \in S_n$  is reduced to a permutation  $\sigma \in S_m$ ,  $m \leq n$ , then  $d_{pt}(\pi) = d_{pt}(\sigma)$ .*

*Proof.* Clearly  $d_{pt}(\pi) \leq d_{pt}(\sigma)$ , since any prefix transposition on  $\sigma$  may be mimicked by a prefix transposition on  $\pi$ . It remains to be shown that  $d_{pt}(\sigma) \leq d_{pt}(\pi)$ .

Form a vector  $V_\pi$  of length  $n$  by inserting  $n - m$  asterisks (\*) into  $\sigma$  such that  $V_\pi[i]$  is an asterisk if  $i > 1$  and  $(\pi_{i-1}, \pi_i)$  is not a breakpoint in  $\pi$  or  $\pi_j = j$  for all  $j \geq i$ . For example, if  $\pi = (1 \ 3 \ 4 \ 5 \ 7 \ 8 \ 2 \ 6 \ 9)$ , then  $\sigma = (1 \ 3 \ 5 \ 2 \ 4)$ , and  $V_\pi = [1 \ 3 \ * \ * \ 5 \ * \ 2 \ 4 \ *]$ . Then any prefix transposition on  $\pi$  can be applied to  $V_\pi$ , and ignoring asterisks this prefix transposition can be applied to  $\sigma$ . Note that if any prefix transposition on  $V_\pi$  moves a block that consists only of asterisks, then the corresponding prefix transposition on  $\sigma$  does nothing, so can be ignored. A sequence of prefix transpositions that sorts  $\pi$  will sort  $V_\pi$  (ignoring asterisks), and so a sequence of prefix transpositions of at most the same length exists that sorts  $\sigma$ . Hence  $d_{pt}(\sigma) \leq d_{pt}(\pi)$ , and the lemma has been proved.  $\square$

**Lemma 4.** *Let  $A_4(\pi)$  and  $A_5(\pi)$  be the number of prefix transpositions applied by algorithms 4 and 5, respectively, to sort an unsigned permutation  $\pi$ . If  $\pi$  is reduced to an unsigned permutation  $\sigma$ , then  $A_4(\pi) = A_4(\sigma)$  and  $A_5(\pi) = A_5(\sigma)$ .*

*Proof.* The claim follows directly from the fact that algorithms 4 and 5 never apply a prefix transposition that breaks a strip apart.  $\square$

To prove that the approximation ratio of Algorithm 4 is tight (see Theorem 2), we show that there exists a class of permutations, denoted by  $H^n$ , for which  $A_4(H^n)$  is (asymptotically) twice the prefix transposition distance.

**Lemma 5.** *Let  $H^n$  be an unsigned permutation of size  $n$ , where  $n$  is an even integer, such that*

$$H_i^n = \begin{cases} n - \frac{i-1}{2} & \text{if } i \in \{1, 3, 5, \dots, n-1\} \\ \frac{i}{2} & \text{if } i \in \{2, 4, 6, \dots, n\}. \end{cases}$$

*Then, we have  $d_{pt}(H^n) = \frac{n}{2}$ .*

*Proof.* The proof follows from induction on  $n$ . As for the base case, we have  $d_{pt}(H^2) = 1$ . As for the induction step, assume that  $d_{pt}(H^k) = \frac{k}{2}$  for  $k \in \{2, 4, \dots, n\}$ . If we apply the prefix transposition  $pt(3, 4)$  on  $H^{n+2}$ , we obtain the unsigned permutation  $\pi = H^{n+2} \circ pt(3, 4)$  such that

$$\pi_i = \begin{cases} n+i & \text{if } i \in \{1, 2\} \\ i-2 & \text{if } i \in \{3, 4\} \\ n - \frac{i-5}{2} & \text{if } i \in \{5, 7, 9, \dots, n+1\} \\ \frac{i}{2} & \text{if } i \in \{6, 8, 10, \dots, n+2\}. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $H^n$ , therefore  $d_{pt}(H^{n+2}) \leq 1 + d_{pt}(H^n) = \frac{n+2}{2}$ . Since  $d_{pt}(H^{n+2}) \geq \frac{b_{pt}(H^{n+2})}{2} = \frac{n+2}{2}$ , we conclude that  $d_{pt}(H^{n+2}) = \frac{n+2}{2}$ , and the lemma follows.  $\square$

**Lemma 6.**  $A_4(H^n) = n - 1$ .

*Proof.* The proof follows from induction on  $n$ . As for the base case, we have  $A_4(H^2) = 1$ . As for the induction step, assume that  $A_4(H^k) = k - 1$  for  $k \in \{2, 4, \dots, n\}$ . If permutation  $H^{n+2}$  is given as input to Algorithm 4, it will apply the prefix transposition  $pt(2, n+3)$  followed by prefix transposition  $pt(2, 3)$ , yielding the permutation  $\pi$  such that

$$\pi_i = \begin{cases} n+1 & \text{if } i = 1 \\ i-1 & \text{if } i \in \{2, 3\} \\ n - \frac{i-4}{2} & \text{if } i \in \{4, 6, 8, \dots, n\} \\ \frac{i+1}{2} & \text{if } i \in \{5, 7, 9, \dots, n+1\} \\ n+2 & \text{if } i = n+2. \end{cases}$$

But the permutation  $\pi$  can be reduced to the permutation  $H^n$ , therefore  $A_4(H^{n+2}) = A_4(H^n) + 2 = n + 1$  and the lemma follows.  $\square$

**Theorem 2.** *The approximation ratio of Algorithm 4 is tight.*

*Proof.* By lemmas 5 and 6, we have  $\frac{A_4(H^n)}{d_{pt}(H^n)} = \frac{2n-2}{n}$ . Since this ratio converges to 2 in the limit, the theorem follows.  $\square$

The proof that the approximation ratio of Algorithm 5 is tight (see Theorem 3) is very similar to that of Theorem 2. We show that there exists a class of permutations, denoted by  $P^n$ , for which  $A_5(P^n)$  is (asymptotically) twice the prefix transposition distance. The only difference is that we need more intermediate results (see Lemmas 7, 8, and 9).

**Lemma 7.** *Let  $J^n$  be an unsigned permutation of size  $n$ , where  $n$  is an even integer, such that*

$$J_i^n = \begin{cases} 2i & \text{if } i \in \{1, 2, 3, \dots, \frac{n}{2}\} \\ 2(i - \frac{n}{2}) - 1 & \text{if } i \in \{\frac{n}{2} + 1, \frac{n}{2} + 2, \frac{n}{2} + 3, \dots, n\}. \end{cases}$$

*Then, we have  $d_{pt}(J^n) = \frac{n}{2}$ .*

*Proof.* The proof follows from induction on  $n$ . As for the base case, we have  $d_{pt}(J^2) = 1$ . As for the induction step, assume that  $d_{pt}(J^k) = \frac{k}{2}$  for  $k \in \{2, 4, \dots, n\}$ . If we apply the prefix transposition  $pt(2, \frac{n+2}{2} + 2)$  on  $J^{n+2}$ , we obtain the unsigned permutation  $\pi = J^{n+2} \circ pt(2, \frac{n+2}{2} + 2)$  such that

$$\pi_i = \begin{cases} 2(i+1) & \text{if } i \in \{1, 2, 3, \dots, \frac{n}{2}\} \\ 1 & \text{if } i = \frac{n+2}{2} \\ 2 & \text{if } i = \frac{n+2}{2} + 1 \\ 2(i - \frac{n+2}{2}) - 1 & \text{if } i \in \{\frac{n+2}{2} + 2, \frac{n}{2} + 3, \frac{n}{2} + 4, \dots, n+2\}. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $J^n$ , therefore  $d_{pt}(J^{n+2}) \leq 1 + d_{pt}(J^n) = \frac{n+2}{2}$ . Since  $d_{pt}(J^{n+2}) \geq \frac{b_{pt}(J^{n+2})}{2} = \frac{n+2}{2}$ , we conclude that  $d_{pt}(J^{n+2}) = \frac{n+2}{2}$ , and the lemma follows.  $\square$

**Lemma 8.** *Let  $K^n$  be an unsigned permutation of size  $n$ , where  $n$  is an even integer, such that*

$$K_i^n = \begin{cases} \frac{n}{2} + \frac{i+1}{2} & \text{if } i \in \{1, 3, 5, \dots, n-1\} \\ \frac{i}{2} & \text{if } i \in \{2, 4, 6, \dots, n\}. \end{cases}$$

*Then, we have  $d_{pt}(K^n) = \frac{n}{2}$ .*

*Proof.* The proof follows from induction on  $n$ . As for the base case, we have  $d_{pt}(K^2) = 1$ . As for the induction step, assume that  $d_{pt}(K^k) = \frac{k}{2}$  for  $k \in \{2, 4, \dots, n\}$ . If we apply the prefix transposition  $pt(n+2, n+3)$  on  $K^{n+2}$ , we obtain the unsigned permutation  $\pi = K^{n+2} \circ pt(n+2, n+3)$  such that

$$\pi_i = \begin{cases} \frac{n+2}{2} & \text{if } i = 1 \\ \frac{n+2}{2} + \frac{i}{2} & \text{if } i \in \{2, 4, 6, \dots, n+2\} \\ \frac{i-1}{2} & \text{if } i \in \{3, 5, 7, \dots, n+1\}. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $K^n$ , therefore  $d_{pt}(K^{n+2}) \leq 1 + d_{pt}(K^n) = \frac{n+2}{2}$ . Since  $d_{pt}(K^{n+2}) \geq \frac{b_{pt}(K^{n+2})}{2} = \frac{n+2}{2}$ , we conclude that  $d_{pt}(K^{n+2}) = \frac{n+2}{2}$ , and the lemma follows.  $\square$

**Lemma 9.** *Let  $L^n$  be an unsigned permutation of size  $n$ , where  $n$  is an even integer and  $n \geq 4$ , such that*

$$L_i^n = \begin{cases} n & \text{if } i = 1 \\ 1 & \text{if } i = 2 \\ i & \text{if } i \in \{3, 5, 7, \dots, n-1\} \\ i-2 & \text{if } i \in \{4, 6, 8, \dots, n\}. \end{cases}$$

Then, we have  $d_{pt}(L^n) = \frac{n}{2}$ .

*Proof.* First of all, note that  $d_{pt}(L^n) \geq \frac{b_{pt}(L^n)}{2} = \frac{n}{2}$ . To show that this lower bound is tight, we divide our analysis into two cases:

1.  $n \equiv 0 \pmod{4}$ .

In this case, we divide the sorting process of permutation  $L^n$  into two phases. In the first phase, we transform  $L^n$  into permutation  $M^n$  such that

$$M_i^n = \begin{cases} 2i+1 & \text{if } i \in \{1, 3, 5, \dots, \frac{n}{2}-1\} \\ 2i & \text{if } i \in \{2, 4, 6, \dots, \frac{n}{2}\} \\ 2(i-\frac{n}{2})-1 & \text{if } i \in \{\frac{n}{2}+1, \frac{n}{2}+3, \frac{n}{2}+5, \dots, n-1\} \\ 2(i-\frac{n}{2})-2 & \text{if } i \in \{\frac{n}{2}+2, \frac{n}{2}+4, \frac{n}{2}+6, \dots, n\}. \end{cases}$$

In the second phase, we transform  $M^n$  into the identity permutation. Since transforming  $L^n$  into  $M^n$  is equivalent to transforming  $M^{n-1} \circ L^n$  into the identity permutation, we can conclude that  $d_{pt}(L^n) \leq d_{pt}(M^{n-1} \circ L^n) + d_{pt}(M^n)$ .

We have that

$$M_i^{n-1} = \begin{cases} \frac{n}{2} + \frac{i+1}{2} & \text{if } i \in \{1, 5, 9, \dots, n-3\} \\ \frac{n}{2} + \frac{i+2}{2} & \text{if } i \in \{2, 6, 10, \dots, n-2\} \\ \frac{i-1}{2} & \text{if } i \in \{3, 7, 11, \dots, n-1\} \\ \frac{i}{2} & \text{if } i \in \{4, 8, 12, \dots, n\}, \end{cases}$$

therefore the permutation  $\pi = M^{n-1} \circ L^n$  is such that

$$\pi_i = \begin{cases} \frac{n}{2} & \text{if } i = 1 \\ \frac{n}{2} + 1 & \text{if } i = 2 \\ \frac{i-1}{2} & \text{if } i \in \{3, 7, 11, \dots, n-1\} \\ \frac{n}{2} + \frac{i}{2} & \text{if } i \in \{4, 8, 12, \dots, n\} \\ \frac{n}{2} + \frac{i+1}{2} & \text{if } i \in \{5, 9, 13, \dots, n-3\} \\ \frac{i-2}{2} & \text{if } i \in \{6, 10, 14, \dots, n-2\}. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $K^{\frac{n}{2}}$ , therefore  $d_{pt}(\pi) = d_{pt}(K^{\frac{n}{2}})$ . Besides, permutation  $M^n$  can be reduced to permutation  $J^{\frac{n}{2}}$ . Thus,  $d_{pt}(L^n) \leq d_{pt}(M^{n-1} \circ L^n) + d_{pt}(M^n) = d_{pt}(K^{\frac{n}{2}}) + d_{pt}(J^{\frac{n}{2}}) = \frac{n}{2}$ .

2.  $n \equiv 2 \pmod{4}$ .

In this case, we also divide the sorting process of permutation  $L^n$  into two phases. In the first phase, we transform  $L^n$  into permutation  $O^n$  such that

$$O_i^n = \begin{cases} 2 & \text{if } i = 1 \\ 2i + 1 & \text{if } i \in \{2, 4, 6, \dots, \frac{n}{2} - 1\} \\ 2i & \text{if } i \in \{3, 5, 7, \dots, \frac{n}{2}\} \\ 1 & \text{if } i = \frac{n}{2} + 1 \\ 2(i - \frac{n}{2}) - 1 & \text{if } i \in \{\frac{n}{2} + 2, \frac{n}{2} + 4, \frac{n}{2} + 6, \dots, n - 1\} \\ 2(i - \frac{n}{2}) - 2 & \text{if } i \in \{\frac{n}{2} + 3, \frac{n}{2} + 5, \frac{n}{2} + 7, \dots, n\}. \end{cases}$$

In the second phase, we transform  $O^n$  into the identity permutation. Since transforming  $L^n$  into  $O^n$  is equivalent to transforming  $O^{n-1} \circ L^n$  into the identity permutation, we can conclude that  $d_{pt}(L^n) \leq d_{pt}(O^{n-1} \circ L^n) + d_{pt}(O^n)$ .

We have that

$$O_i^{n-1} = \begin{cases} \frac{n}{2} + 1 & \text{if } i = 1 \\ 1 & \text{if } i = 2 \\ \frac{n}{2} + \frac{i+1}{2} & \text{if } i \in \{3, 7, 11, \dots, n - 3\} \\ \frac{n}{2} + \frac{i+2}{2} & \text{if } i \in \{4, 8, 12, \dots, n - 2\} \\ \frac{i-1}{2} & \text{if } i \in \{5, 9, 13, \dots, n - 1\} \\ \frac{i}{2} & \text{if } i \in \{6, 10, 14, \dots, n\}, \end{cases}$$

therefore the permutation  $\pi = O^{n-1} \circ L^n$  is such that

$$\pi_i = \begin{cases} \frac{n}{2} & \text{if } i = 1 \\ \frac{n}{2} + 1 & \text{if } i = 2 \\ \frac{n}{2} + \frac{i+1}{2} & \text{if } i \in \{3, 7, 11, \dots, n - 3\} \\ \frac{i-2}{2} & \text{if } i \in \{4, 8, 12, \dots, n - 2\} \\ \frac{i-1}{2} & \text{if } i \in \{5, 9, 13, \dots, n - 1\} \\ \frac{n}{2} + \frac{i}{2} & \text{if } i \in \{6, 10, 14, \dots, n\}. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $K^{\frac{n-2}{2}}$ , therefore  $d_{pt}(\pi) = d_{pt}(K^{\frac{n-2}{2}})$ . Besides, the permutation  $O^n$  can be reduced to permutation  $J^{\frac{n+2}{2}}$ . Thus,  $d_{pt}(L^n) \leq d_{pt}(O^{n-1} \circ L^n) + d_{pt}(O^n) = d_{pt}(K^{\frac{n-2}{2}}) + d_{pt}(J^{\frac{n+2}{2}}) = \frac{n}{2}$ .

Since  $d_{pt}(L^n) \leq \frac{n}{2}$  in both cases, we can conclude that  $d_{pt}(L^n) = \frac{n}{2}$ .  $\square$

**Lemma 10.** *Let  $P^n$  be an unsigned permutation of size  $n$ , where  $n$  is an even integer and  $n \geq 6$ , such that*



$$P_i^n = \begin{cases} 1 & \text{if } i = 1 \\ i + 1 & \text{if } i \in \{2, 4, 6, \dots, n - 2\} \\ i - 1 & \text{if } i \in \{3, 5, 7, \dots, n - 1\} \\ n & \text{if } i = n. \end{cases}$$

Then, we have  $d_{pt}(P^n) = \frac{n}{2}$ .

*Proof.* Applying the prefix transposition  $pt(n-1, n)$  on  $P^n$ , we obtain the permutation  $\pi = P^n \circ pt(n-1, n)$  such that

$$\pi_i = \begin{cases} n - 2 & \text{if } i = 1 \\ 1 & \text{if } i = 2 \\ i & \text{if } i \in \{3, 5, 7, \dots, n - 3\} \\ i - 2 & \text{if } i \in \{4, 6, 8, \dots, n - 2\} \\ n - 1 & \text{if } i = n - 1 \\ n & \text{if } i = n. \end{cases}$$

But permutation  $P^n$  can be reduced to permutation  $L^{n-2}$ , therefore  $d_{pt}(P^n) \leq 1 + d_{pt}(L^{n-2}) = \frac{n}{2}$ . Since  $d_{pt}(P^n) \geq \frac{b_{pt}(P^n)}{2} = \frac{n-1}{2}$ , we can conclude that  $d_{pt}(P^n) = \frac{n}{2}$ .  $\square$

**Lemma 11.**  $A_5(P^n) = n - 2$ .

*Proof.* The proof follows from induction on  $n$ . As for the base case, we have  $A_5(P^6) = 4$ . As for the induction step, assume that  $A_5(P^k) = k - 2$  for  $k \in \{6, 8, \dots, n\}$ . If permutation  $P^{n+2}$  is given as input to Algorithm 5, it will apply prefix transposition  $pt(2, 3)$  followed by prefix transposition  $pt(2, 5)$ , yielding the permutation  $\pi = P^{n+2} \circ pt(2, 3) \circ pt(2, 5)$  such that

$$\pi_i = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = 2 \\ 5 & \text{if } i = 3 \\ i + 1 & \text{if } i \in \{4, 6, 8, \dots, n\} \\ i - 1 & \text{if } i \in \{5, 7, 9, \dots, n + 1\} \\ n + 2 & \text{if } i = n + 2. \end{cases}$$

But the permutation  $\pi$  can be reduced to permutation  $P^n$ , therefore  $A_5(P^{n+2}) = 2 + A_5(P^n) = n$  and the lemma follows.  $\square$

**Theorem 3.** *The approximation ratio of Algorithm 5 is tight.*

*Proof.* By lemmas 10 and 11, we have  $\frac{A_5(P^n)}{d_{pt}(P^n)} = \frac{2n-4}{n}$ . Since this ratio converges to 2 in the limit, the theorem follows.  $\square$

## 2.5 Conclusion

It has been a long time since researchers have addressed the problem of finding a shortest sequence of rearrangement events that transform the genome of one species

into another. Representing the order of the genes in a genome as permutations, the previous problem can be equivalently stated as the combinatorial problem of sorting a permutation using a minimum number of rearrangement events. In general, sorting permutations using rearrangement events is a difficult problem; therefore, the best known solution for many of its variants are approximation algorithms and heuristics. That is why we have decided to build GRAAu, a tool for evaluating approximation algorithms and heuristics for genome rearrangements.

To build this tool, we computed the rearrangement distances of all permutations in  $S_n$ ,  $1 \leq n \leq 13$ , and in  $S_n^\pm$ ,  $1 \leq n \leq 10$ , with respect to a number of rearrangement models regarded in the literature that take into account reversals or transpositions. To best of our knowledge, this was the first time that the rearrangement distances have been computed for these values of  $n$ . This achievement is attributable to our development of a simple and flexible breadth-first search algorithm that is more efficient in terms of memory usage than any other algorithm that we have found in the literature. In addition, to improve its execution time, which is exponential on the size of permutations, we have developed a way to parallelize it.

By analyzing the distribution of the rearrangement distances, we were able to notice some interesting facts. First, we have looked for other works that also presented the distribution of the rearrangement distances so that we could compare them with the distribution we have computed, and we have discovered that the reversal distance distribution presented by Kececioglu and Sankoff [84] is not correct. Then, we have looked for conjectures on the diameter of  $S_n$  and  $S_n^\pm$  that could be validated. As a result, we have verified that the conjecture of Dias and Meidanis [42] on the prefix transposition diameter is valid for  $n = 12$  and  $n = 13$ , whereas the conjecture of Walter *et al.* [123] on the signed reversal and transposition diameter is not valid for  $n = 7$  and  $n = 9$ . For this reason, we have presented a new conjecture on the signed reversal and transposition diameter. Last, as an attempt to better characterize how the rearrangement distances are distributed, we have proposed two new measures – the traversal diameter and the longevity – and we have presented conjectures on them as well.

We have illustrated the application of GRAAu by using it to evaluate two approximation algorithms for the problem of sorting by prefix reversals and two approximation algorithms for the problem of sorting by prefix transpositions. We have focused on the use of GRAAu's output for proving the tightness of the approximation ratio of these approximation algorithms, and based on the values of maximum ratio and on the permutations that exhibited it, we have proved that the approximation ratios of three out of the four algorithms analyzed are tight. Although we have not been able to prove the tightness of the approximation ratio of one approximation algorithm, we have conjectured that its approximation ratio is tight because the maximum ratio obtained for this algorithm was converging to the theoretical approximation ratio. The tightness results regarding the two approximation algorithms for the problem of sort-

ing by prefix reversals contradict the hypothesis raised by Fischer and Ginzinger [55] that the approximation ratio of the approximation algorithm yielded by their greedy strategy may be lowered.

Finally, we remark that GRAAu can be used for purposes other than evaluating genome rearrangement algorithms. For instance, Labarre [90] has proved a lower bound on the prefix transposition distance, and for comparing it to the lower bounds proved by Dias and Meidanis [42] and Chitturi and Sudborough [29], he had to compute the prefix transposition distance of all permutations in  $S_n$  for  $1 \leq n \leq 12$ . However, he could have used GRAAu to perform such comparison. By doing so, he would benefit in two ways: (1) he would not spend time and effort computing the distances, and (2) the comparison could have been performed for  $n = 13$ .



## Chapter 3

# A General Heuristic for Genome Rearrangement Problems \*

**Abstract:** In this paper we present a general heuristic for several problems in the genome rearrangement field. Our heuristic does not solve any problem directly, it is rather used to improve the solutions provided by any non-optimal algorithm that solve them. Therefore, we have implemented several algorithms described in the literature and several algorithms developed by ourselves. As a whole, we implemented 23 algorithms for 9 well known problems in the genome rearrangement field. A total of 13 algorithms were implemented for problems that use the notions of prefix and suffix operations. In addition, we worked on 5 algorithms for the classic problem of sorting by transposition and we conclude the experiments by presenting results for 3 approximation algorithms for the sorting by reversals and transpositions problem and 2 approximation algorithms for the sorting by reversals problem. Another algorithm with better approximation ratio can be found for the last genome rearrangement problem, but it is purely theoretical with no practical implementation. The algorithms we implemented in addition to our heuristic lead to the best practical results in each case. In particular, we were able to improve results on the sorting by transpositions problem, which is a very special case because many efforts have been made to generate algorithms with good results in practice and some of these algorithms provide results that equal the optimum solutions in many cases. Our source codes and benchmarks are freely available upon request from the authors so that it will be easier to compare new approaches against our results.

---

\* *Ulisses Dias, Gustavo Rodrigues Galvão, Carla Négre Lintzmayer, and Zanoni Dias. A general heuristic for genome rearrangement problems. Journal of Bioinformatics and Computational Biology, Volume 12, Issue 03, 26 pages, 2014. Copyright 2014 Imperial College Press. DOI: <http://dx.doi.org/10.1142/S0219720014500127>*

### 3.1 Introduction

Genome rearrangements are mutational events that affect large stretches of the DNA sequence. They occur when a chromosome breaks at two or more locations and its pieces are reassembled in a different order. It was proposed in 1936 by Dobzhansky and Sturtevant that the degree of disorder between two genomes can be an indicator of the evolution distance between them [54]. Due to the principle of parsimony, it is common to consider the minimum number of events that transform one genome into the other as an approximation for the evolutive distance. The study of the combinatorial problems in genome rearrangements area exists for over twenty years now [64, 126] and despite they have become, in some sense, independent of the application, their biological background still inspires some variants.

Reversals and transpositions are the best studied rearrangement events. The former occurs when a block of DNA sequence is reverted and the latter occurs when a block of DNA moves from one place to another in the same chromosome. The reversal (transposition) distance is the minimum number of such operations that transforms a given genome into another. Caprara [23] proved that finding this minimum number of reversals is a NP-hard problem while Bulteau, Fertin and Rusu [22] proved the same for transpositions. The best algorithms for both have an approximation factor of 1.375 [15, 48].

Several efforts have been made to consider algorithms that take more than one rearrangement operation into account. Here, we consider the case when reversals and transpositions are allowed. The first algorithm for this problem was developed by Walter, Dias and Meidanis [123] with an approximation factor of 3. The author's only concern was to prove the theoretical approximation bound, so they overlooked some details that could have made the algorithm more suitable for a practical analysis. Thus, we decided to add these details in order to create a significantly improved version of this algorithm (see Section 3.5).

The approximation factor for the sorting by transpositions and reversals problem was later improved to  $2k$  by Rahman, Shatabda and Hasan [107] where  $k$  is the approximation ratio of the algorithm used for cycle decomposition. Although the best approximation ratio for the cycle decomposition problem was recently published by Chen [27] with  $k = \frac{17}{12} + \epsilon \approx 1.4167 + \epsilon$ , for any positive  $\epsilon$ , we implemented the cycle decomposition algorithm devised by Christie [30] with  $k = \frac{3}{2} = 1.5$  because it is simpler. Therefore, all the algorithms for the sorting by reversals and transpositions problem have the same ratio in our implementation.

When genome rearrangement operations affect segments from the beginning of the genome, we call them prefix rearrangements. The well-known Pancake Flipping Problem [46] considers prefix reversals and was proved recently to be NP-hard by Bulteau, Fertin and Rusu [21]. The best algorithm has an approximation factor of 2 and it was given by Fischer and Ginzinger [55]. Lintzmayer and Dias [98] recently gave an improved version of the same algorithm, which prefers bigger prefix reversals

and, although still being a 2-approximation, in practice it showed better results.

When we consider prefix transpositions, the best algorithm is also a 2-approximation given by Dias and Meidanis [42], but the problem remains open. A greedy and better version of their algorithm was given by Galvão and Dias [58].

Another open problem considers both prefix reversals and prefix transpositions. It was introduced by Sharmin *et al.* [111], which also provided a 3-approximation algorithm. Lintzmayer and Dias [98] gave an improved version of this algorithm with greedy features that prefers prefix transpositions that remove two breakpoints at once and bigger prefix reversals. Their algorithm still has an approximation factor of 3, but presents better results in practice. Recently, Dias and Dias [41] presented a 2-approximation algorithm for the same problem.

It is also possible to consider operations restricted to the end of genomes. We call them suffix rearrangements. The three problems involving prefix rearrangements mentioned above were considered along with their suffix versions by Lintzmayer and Dias [98]. They presented two 2-approximation algorithms for each of the three new problems: considering prefix reversals and suffix reversals, considering prefix transpositions and suffix transpositions and considering prefix reversals, prefix transpositions, suffix reversals and suffix transpositions. The two algorithms for each problem follow the same general idea. One of them extends the existing algorithm for the problem that allows only the prefix operations; generally, the actions of the prefix algorithm are mimicked and merged with the corresponding actions of the suffix. The second is always an improved version of the first one that uses greedy choices such as bigger operations or operations that remove more breakpoints.

In sum, approximation algorithms have been proposed for several rearrangement problems. The algorithms provide non-optimal solutions in many cases even for small permutations. Here, we present a method to improve these solutions. We refer to our heuristic as general because it can be used to improve solutions provided by any non-optimal algorithm in the genome rearrangement field. Therefore, the previously mentioned approximation algorithms can benefit from this work. We highlight that our heuristic is not restricted to the problems we tested. We describe in the paper some properties that must be satisfied in order to apply our approach.

After applying our heuristic, we were able to generate the best results in practice for each genome rearrangement problem.

This paper is organized as follows. Section 3.2 defines the notation used throughout the paper and provides a formal presentation for each problem. Section 3.3 describes our general heuristic. Section 3.4 presents the database we created in order to fulfill a requirement in our heuristic. Section 3.5 shows the improvements obtained by applying our heuristic on the solutions provided by a set of non-optimal algorithms. Finally, Section 3.6 condenses the main aspects of this work.

## 3.2 Background

We are interested in the model where the order of genes is known and where the genomes share a subset of genes without duplications, which allows us to represent genomes using permutations. A *permutation*  $\pi$  is a bijection of  $\{1, 2, \dots, n\}$  onto itself. The group of all permutations of  $\{1, 2, \dots, n\}$  is denoted by  $S_n$ , and we write a permutation  $\pi \in S_n$  as  $\pi = (\pi_1 \pi_2 \dots \pi_n)$  such that  $\pi(i) = \pi_i$ .

The composition of two permutations  $\pi$  and  $\sigma$  is the permutation  $\pi \cdot \sigma = (\pi_{\sigma_1} \pi_{\sigma_2} \dots \pi_{\sigma_n})$ . We can see the composition as the relabeling of elements in  $\pi$  according to elements in  $\sigma$ . Let  $\iota = (1 \ 2 \ \dots \ n)$  be the identity permutation. We can easily verify that  $\iota$  is a neutral element such that  $\pi \cdot \iota = \iota \cdot \pi = \pi$ . We define the inverse of a permutation  $\pi$  as the permutation  $\pi^{-1}$  such that  $\pi \cdot \pi^{-1} = \pi^{-1} \cdot \pi = \iota$  and it satisfies  $\pi_{\pi_i}^{-1} = i$ . In other words, it is the function that returns the position in  $\pi$  of each element  $\pi_i$ .

In this paper, we mention several distance genome rearrangement problems that are defined as follows. Let  $\xi$  be a set of rearrangement events that can be applied to  $\pi$ , the distance  $d_\xi(\pi)$  is the minimum number  $t$  of operations  $\xi_1, \xi_2, \dots, \xi_t$  such that  $\pi \cdot \xi_1 \cdot \xi_2 \dots \xi_t = \iota$ .

A *transposition* is an operation  $\rho_t(i, j, k)$ ,  $1 \leq i < j < k \leq n+1$ , that moves blocks of contiguous elements of a permutation  $\pi$  in such a way that  $(\pi_1 \dots \pi_{i-1} \pi_i \dots \pi_{j-1} \pi_j \dots \pi_{k-1} \pi_k \dots \pi_n) \cdot \rho_t(i, j, k) = (\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_{k-1} \pi_i \dots \pi_{j-1} \pi_k \dots \pi_n)$ . The *transposition distance* of a permutation  $\pi$  is denoted by  $d_t(\pi)$ .

A *prefix transposition*  $\rho_{pt}(j, k)$ ,  $2 \leq j < k \leq n+1$ , is an operation equivalent to the transposition  $\rho_t(1, j, k)$ . A *suffix transposition*  $\rho_{st}(i, j)$ ,  $1 \leq i < j \leq n$ , is an operation equivalent to the transposition  $\rho_t(i, j, n+1)$ . The *prefix transposition distance* of a permutation  $\pi$  is denoted by  $d_{pt}(\pi)$ . The *prefix and suffix transposition distance* of a permutation  $\pi$  is denoted by  $d_{ptst}(\pi)$ .

A reversal is an operation  $\rho_r(i, j)$ ,  $1 \leq i < j \leq n$ , that reverses the order of  $\pi[i..j]$ . Therefore,  $(\pi_1 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_j \pi_{j+1} \dots \pi_n) \cdot \rho_r(i, j) = (\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_{i+1} \pi_i \pi_{j+1} \dots \pi_n)$ . The *reversal distance* of permutation  $\pi$  is denoted by  $d_r(\pi)$ . When reversals and transpositions are allowed, we have the distance problem denoted by  $d_{rt}(\pi)$ .

A *prefix reversal*  $\rho_{pr}(j)$ ,  $2 \leq j \leq n$ , is equivalent to the reversal  $\rho_r(1, j)$ . The *suffix reversal*  $\rho_{sr}(i)$ ,  $1 \leq i < n$ , is an operation equivalent to the reversal  $\rho_r(i, n)$ . The *prefix reversal distance* of a permutation  $\pi$  is denoted by  $d_{pr}(\pi)$ . The *prefix and suffix reversal distance* of a permutation  $\pi$  is denoted by  $d_{prsr}(\pi)$ .

The *prefix reversal and prefix transposition distance* is denoted by  $d_{prpt}(\pi)$ . When prefix and suffix transpositions as well as prefix and suffix reversals are allowed, we have the distance problem denoted by  $d_{prptsrst}(\pi)$ .

Given a permutation  $\pi \in S_n$ , we extend it with two elements  $\pi_0 = 0$  and  $\pi_{n+1} = n+1$ . The extended permutation is still denoted as  $\pi$ . A *transposition breakpoint* of  $\pi \in S_n$  is a pair of adjacent elements that are not consecutive, that is, a pair  $(\pi_i,$



$\pi_{i+1}$ ) such that  $\pi_{i+1} - \pi_i \neq 1$ ,  $0 \leq i \leq n$ .

Transposition breakpoints divide a permutation into *transposition strips*, which are maximal intervals with no transposition breakpoints. Christie [30] showed that every permutation  $\pi$  can be uniquely transformed into a reduced permutation  $\pi_r$  such that  $d_t(\pi) = d_t(\pi_r)$ . Moreover, Dias and Meidanis [42] showed that this property is also valid for the prefix transposition distance, that is,  $d_{pt}(\pi_r) = d_{pt}(\pi)$ .

We say that a permutation is *reduced* if it only contains transposition strips of length 1. We can reduce any permutation that contains one or more transposition strips of length greater than 1 as follows. Assuming the permutation is in the extended form, keep the minimal element of each strip of  $\pi$  and discard the others. After that, renumber the sequence appropriately. For example, to reduce the permutation (0 2 3 1 4 5 6), in which transposition strips are underlined, we keep only the minimal element of each strip and generated (0 2 1 4). This permutation is not valid because the element “3” is missing, so we renumber it in order to obtain the reduced permutation (0 2 1 3).

The reduced permutation as defined above cannot be used for distance problems that involve any kind of reversals. That happens because usually  $d_\xi(\pi_r) \neq d_\xi(\pi)$  if  $r, pr, sr \in \xi$ . We have tried to circumvent this issue by adopting *reversal breakpoints* and partially reducing the input permutations. A pair of elements  $\pi_i$  and  $\pi_{i+1}$ , with  $0 \leq i \leq n$ , is a reversal breakpoint if  $|\pi_{i+1} - \pi_i| \neq 1$ .

When we are dealing with prefix rearrangement problems (either prefix transposition or prefix reversals), we always consider the pair  $(\pi_0, \pi_1)$  as a breakpoint. Similarly, when we are dealing with suffix rearrangement problems, we always consider the pair  $(\pi_n, \pi_{n+1})$  as a breakpoint.

Reversal breakpoints divide a permutation into *reversal strips*, which are maximal intervals with no reversal breakpoints. We use the notion of reversal strips to partially reduce permutations. We say that a permutation is *partially reduced* if it only contains reversal strips of length 1 or 2.

We can transform any permutation  $\pi$  that contains one or more reversal strips of length greater than 2 into a partially reduced permutation  $\pi_{pr}$  as follows. Assuming  $\pi$  is a permutation in the extended form, for each strip longer than 2 elements we keep the first and the second elements and discard the others. After that, we renumber the sequence appropriately.

For example, to partially reduce the permutation (0 2 3 1 4 5 6 9 8 7 10), in which reversal strips are underlined, we keep only the first and the second element of each strip longer than 2 elements and generate (0 2 3 1 4 5 9 8 10). This permutation is not valid because the elements “6” and “7” are missing, so we renumber it in order to obtain the partially reduced permutation (0 2 3 1 4 5 7 6 8).

When we partially reduce a permutation, we save the number of discarded elements from each reversal strip in order to later recreate the original permutation. In the previous partial reduction example, the array  $[0, 0, 0, 1, 1, 0]$  stores the number of

discarded elements and it is enough to recover the original permutation. It informs that one element was removed from the fourth reversal strip and one was removed from the fifth reversal strip, so we can add those elements taking into account that reversal strips are either increasing or decreasing sequences of consecutive elements. After that, we renumber the sequence appropriately to obtain the original permutation.

The procedure for recreating a reduced permutation is similar, but in this case each element in the reduced permutation is a transposition strip (note that we do not consider a sequence of decreasing elements as a transposition strip). Therefore, the discarded elements were increasing sequences of consecutive elements. So, we simply add as many elements as indicated by the array in increasing order and later renumber the sequence appropriately.

For conciseness, we may henceforth talk of breakpoints and strips where we mean reversal or transposition breakpoints and reversal or transposition strips. To distinguish between each case, one should keep in mind that we use the notion of transposition breakpoints or transposition strips if reversals are not allowed by the rearrangement problem. Other than that, breakpoints and strips may be safely understood as reversal breakpoints and reversal strips. Also, note that the elements  $(\pi_0, \pi_1)$  and  $(\pi_n, \pi_{n+1})$  are always breakpoints if at least one of the allowed operations are prefix operations or suffix operations, respectively.

### 3.3 A General Heuristic

Let  $\xi$  be a set of allowed rearrangement events that characterize a distance problem. We represent a solution as a sequence  $S = \langle S_0, S_1, \dots, S_m \rangle$ , where each  $S_i$  is a permutation,  $S_i = S_{i-1} \cdot \rho_\xi$ ,  $1 \leq i \leq m$ ,  $S_0$  is the input permutation and  $S_m = \iota$ . A sliding window  $W = \langle S_{start}, \dots, S_{end} \rangle$  is a subsequence of  $S$ ,  $0 \leq start < end \leq m$ .

Our heuristic works by iteratively improving an initial solution. Each step makes a local change within a sliding window, which moves across the solution. We repeat this step until the sliding window reaches the last element in the solution. The main idea here is to transform the sliding window into a small instance of the sorting problem. Then, we can retrieve the optimal solution for that instance in the database built as described in Section 3.4.

Our database stores the exact solution for all possible permutations whose sizes go from 1 to 12. That said, we constrain the size of the sub-problem instances accordingly, and hence limit the number of elements in the sliding window.

Let  $W = \langle S_{start}, S_{start+1}, \dots, S_{end} \rangle$  be our window. The sequence of operations that transform  $S_{start}$  into  $S_{end}$  is the same that would be used to transform  $S_{end}^{-1} \cdot S_{start}$  into  $\iota$ . That is because we can relabel the entire window using composition in order to create the sequence  $\langle S_{end}^{-1} \cdot S_{start}, S_{end}^{-1} \cdot S_{start+1}, \dots, S_{end}^{-1} \cdot S_{end} \rangle$ , where  $S_{end}^{-1} \cdot S_{end} = \iota$ .

This property offers a clue about how the sub-problem can be created from a given window. In cases where reversals are not being used, we look for a window

where  $\pi = S_{end}^{-1} \cdot S_{start}$  is equivalent by reduction to a permutation  $\pi_r$ , such that  $|\pi_r| \leq 12$ , then we can retrieve the solution for  $\pi$  in our database and use it to mimic a solution for  $S_{end}^{-1} \cdot S_{start}$ . That solution for  $S_{end}^{-1} \cdot S_{start}$  can be transformed back into a window  $W_1 = \langle S_{start}, \dots, S_{end} \rangle$ . If  $|W_1| < |W|$  we replace the window  $W$  for the optimized window  $W_1$  in  $S$ .

The procedure for the cases where reversals are allowed is similar, except that we partially reduce  $\pi$  to the permutation  $\pi_{pr}$ . In this case, we look for windows such that  $\pi = S_{end}^{-1} \cdot S_{start}$  is equivalent by partial reduction to a permutation  $\pi_{pr}$ , such that  $|\pi_{pr}| \leq 12$ .

Our method begins by defining  $S_0$  as the first element in the sliding window ( $start = 0$ ). The last element in the window depends on the database we have available. Algorithm 6 shows how to dynamically identify the last element in the sliding window from the solution  $S$  and the index of the first element.

In short, Algorithm 6 searches for the biggest index  $end$  such that the reduction of  $S_{end}^{-1} \cdot S_{start}$  results in a permutation whose size is less than or equals to 12. Therefore, the while loop in Algorithm 6 stops when we find the  $end$  or when there is no more permutations in  $S$ . The function *reduce* in line 4 performs a reduction or partial reduction depending on the problem we are dealing with. The *discarded* variable is an array with the number of discarded elements from each strip. This array helps to recreate the original permutation as explained in Section 3.2.

---

**Algorithm 6:** Sliding Window

---

**Data:**  $S = \langle S_0, S_1, \dots, S_m \rangle, start, Database\_Size$

```

1  $end \leftarrow start + 1$ 
2 while True do
3    $\pi \leftarrow S_{end}^{-1} \cdot S_{start}$ 
4    $\pi_{red}, discarded \leftarrow reduce(\pi)$ 
5   if  $|\pi_{red}| > Database\_Size$  then
6     return  $end - 1$ 
7   if  $end = m$  then
8     return  $end$ 
9    $end \leftarrow end + 1$ 

```

---

We start Algorithm 6 by setting  $end = start + 1$  in line 1. This line could be optimized depending on the operation we allow. For example, for transpositions we know that each operation can change the number of breakpoints by at most 3. Thus, we could have made  $end = start + \lfloor Database\_Size/3 \rfloor$ . However, we want to let this algorithm as generic as possible.

Algorithm 7 shows our iterative procedure. The stop condition occurs when  $S_{end}$  is the last element in  $S$  (lines 14–15). Each step we select a new sliding window by incrementing  $start$  (line 16) and using Algorithm 6 to find a proper  $end$  (line 3).

After that, we search the database for a reduced version  $\pi_{red}$  of the permutation

**Algorithm 7:** General Heuristic

---

**Data:**  $S = \langle S_0, S_1, \dots, S_m \rangle, Database$

```

1 start  $\leftarrow$  0
2 while True do
3   end  $\leftarrow$  Sliding_Window(S, start, Database.size)
4    $W \leftarrow \langle S_{start}, \dots, S_{end} \rangle$ 
5    $\pi \leftarrow S_{end}^{-1} \cdot S_{start}$ 
6    $\pi_{red}, discarded \leftarrow reduce(\pi)$ 
7   database_seq  $\leftarrow Database.get(\pi_{red})$ 
8   if  $|database\_seq| < end - start + 1$  then
9      $W_1 \leftarrow []$ 
10    for each permutation in database_seq do
11       $expanded \leftarrow expand(permutation, discarded)$ 
12       $W_1.append(S_{end} \cdot expanded)$ 
13     $S.replace(W, W_1)$ 
14  if end = m then
15    return S
16  start  $\leftarrow start + 1$ 

```

---

$\pi = S_{end}^{-1} \cdot S_{start}$  and retrieve a sorting sequence for it (lines 5 – 7). This sequence can be adjusted to fit in  $S$  by using a 2-step process. The first step is to expand each permutation in the sorting sequence. Recall that we generate the *discard* array every time we reduce (or partially reduce) a permutation. This array is helpful not only to transform  $\pi_{red}$  back into  $\pi$ , but also to transform the sorting sequence for  $\pi_{red}$  into a sorting sequence for  $\pi$ . The procedure that performs this task is done by the function *expand* in line 11. We have already explained this procedure with an example in Section 3.2.

After the first step, we have a sorting sequence  $X = \langle S_{end}^{-1} \cdot S_{start}, \dots, \iota \rangle$ . Therefore, we build a new window by applying the composition  $S_{end} \cdot X_i$  to each element  $X_i$  in  $X$  (line 12). Since  $S_{end} \cdot S_{end}^{-1} \cdot S_{start} = S_{start}$  and  $S_{end} \cdot \iota = S_{end}$ , we have just created a new window that can easily replace the original window in  $S$  (line 13).

Algorithm 6 runs in  $O(n)$ , where  $n$  is the number of elements in each permutation in the sorting sequence  $S = \langle S_0, S_1, \dots, S_m \rangle$ . Lines 3 and 4 are the most time consuming steps inside the while loop that goes from line 2 to line 9. The number of iterations is limited by a constant factor that depends on *Database\_Size* (which in our case is 12) and on the number of breakpoints that can be removed by each genome rearrangement operation. In the worst case, one breakpoint can be removed by each operation in the genome rearrangement problems we have studied in this paper. Therefore, the stated complexity follows.

Algorithm 7 runs in  $O(mn)$  time, since the number of iterations performed by the while loop (lines 2 – 16) is limited by the number of elements in  $S$  and the most time

consuming steps in the loop are  $O(n)$ . Note that the for-loop in lines 10–12 iterates over the elements in the window, which is limited by a constant, as we explained above for Algorithm 6.

### 3.4 Solution Database

We have built a database containing optimal solutions for small instances of the genome rearrangement problems described in Section 3.2. As mentioned in Section 3.3, this database stores solutions for all possible permutations up to 12 elements. We have not considered longer permutations due to space constraints.

We generated the solutions by performing breadth-first search in the symmetric group  $S_n$ ,  $1 \leq n \leq 12$ . We start by initializing a permutation queue  $Q$  with  $\iota$  and set its solution to  $\langle \iota \rangle$ . While  $Q$  is not empty, we remove an arbitrary permutation  $\pi$  from  $Q$  and compute all permutations that can be generated from  $\pi$  by applying on it every possible rearrangement event belonging to  $\xi$  that does not break any strip of  $\pi$ . Each permutation, say  $\gamma$ , that has not been generated yet is added to  $Q$  and its solution is set to  $\langle \gamma, T_k, T_{k-1}, \dots, \iota \rangle$ , where  $\langle T_k, T_{k-1}, \dots, \iota \rangle$  is the solution of permutation  $\pi$ .

Note that we have only considered rearrangement events that do not break strips. This is because the heuristic uses the solution to an instance stored in the database to derive a solution to an arbitrary instance of a genome rearrangement problem. Regarding the sorting by transposition problem, Christie [30] shows that, for any permutation  $\pi$ , there is a minimum sequence of transpositions that sorts  $\pi$  without creating new breakpoints. That said, our database stores the optimal solution for any instance of the sorting by transpositions problem. The same happens for the sorting by prefix transpositions problem [42]. However, we cannot guarantee it for any instance of problems that use reversals. In this case, our database stores short solutions that can be used to replace those proposed by approximation algorithms, but they can possibly be non-optimal.

### 3.5 Experimental Results

In order to verify the performance of the proposed heuristic, we implemented 23 approximation algorithms and applied our heuristic to the solutions provided by them. Table 3.1 refers to each algorithm implemented by us for the sake of this analysis. We also referred to the original authors of each algorithm and to the papers where they were initially published. We ran all these algorithms on the same set of arbitrarily large permutations. This set is composed of 58,000 random permutations of sizes ranging from 15 to 300 in intervals of 5, with 1,000 permutations of each size.

Table 3.1 also shows the approximation ratio for each algorithm we implemented. On a side note, we highlight that the algorithm coded as RSH\* was originally proposed

Table 3.1: List of algorithms used in this paper.

Problem	Code	Authors	Ratio
Prefix Reversal	FG	Fischer and Ginzinger [55]	2
	LD	Lintzmayer and Dias [98]	2
Prefix Reversal and Prefix Transposition	DD	Dias and Dias [41]	2
	LD	Lintzmayer and Dias [98]	3
	SEA	Sharmin <i>et al.</i> [111]	3
Prefix Reversal, Prefix Transposition, Suffix Reversal and Suffix Transposition	LD1	Lintzmayer and Dias [98]	2
	LD2	Lintzmayer and Dias [98]	2
Prefix Reversal and Suffix Reversal	LD1	Lintzmayer and Dias [98]	2
	LD2	Lintzmayer and Dias [98]	2
Prefix Transposition and Suffix Transposition	LD1	Lintzmayer and Dias [98]	2
	LD2	Lintzmayer and Dias [98]	2
Prefix Transposition	DM	Dias and Meidanis [42]	2
	GD	Galvão and Dias [58]	2
Reversal	C	Christie [30]	1.5
	KS	Kececioglu and Sankoff [84]	2
Reversal and Transposition	DEA	Dias <i>et al.</i> <sup>This</sup>	3
	RSH*	Rahman, Shatabda and Hasan [107]	3
	WDM	Walter, Dias and Meidanis [123]	3
Transposition	BP	Bafna and Pevzner [10]	1.5
	DD1	Dias and Dias [36]	1.5
	DD2	Dias and Dias [38]	1.375
	EH	Elias and Hartman [48]	1.375
	WDM	Walter, Dias and Meidanis [124]	2.25

as a  $2k$ -approximation algorithm by Rahman, Shatabda and Hasan [107], where  $k$  is the approximation ratio of the algorithm used for cycle decomposition. Since the best algorithm for the cycle decomposition problem has an approximation factor  $k = \frac{17}{12} + \epsilon \approx 1.4167 + \epsilon$  for any positive  $\epsilon$  [27], it would be reasonable to expect an approximation ratio approximately equal to 2.83 for Rahman, Shatabda and Hasan's algorithm. However, we implemented the cycle decomposition algorithm devised by Christie [30] with  $k = \frac{3}{2} = 1.5$  because it is simpler. Therefore, our implementation guarantees the approximation factor 3.

A second remark about Table 3.1 regards the algorithm coded as DEA. We implemented this algorithm based on a greedy removal of reversal breakpoints similarly to WDM. However, the WDM algorithm only tries to remove one transposition breakpoint each time by increasing the first strip with its next element. This approach suffices to prove the theoretical approximation bound.

Our algorithm finds a better outcome by always trying to remove the most amount of breakpoints. One must keep in mind that we will refer only to transpositions  $\rho_t(i, j, k)$  such that  $(\pi_{i-1}, \pi_i)$ ,  $(\pi_{j-1}, \pi_j)$  and  $(\pi_{k-1}, \pi_k)$  are breakpoints, and reversals  $\rho_r(l, m)$  such that  $(\pi_{l-1}, \pi_l)$  and  $(\pi_m, \pi_{m+1})$  are breakpoints. First, the algorithm tries

to remove three breakpoints with one transposition. This is possible if  $\pi_i = \pi_{k-1} \pm 1$ ,  $\pi_j = \pi_{i-1} \pm 1$  and  $\pi_k = \pi_{j-1} \pm 1$ . If that is not possible, it tries to remove two breakpoints with either a transposition or a reversal. A transposition removes two breakpoints if either (i)  $\pi_i = \pi_{k-1} \pm 1$  and  $\pi_j = \pi_{i-1} \pm 1$ , (ii)  $\pi_i = \pi_{k-1} \pm 1$  and  $\pi_k = \pi_{j-1} \pm 1$ , or (iii)  $\pi_j = \pi_{i-1} \pm 1$  and  $\pi_k = \pi_{j-1} \pm 1$ . A reversal removes two breakpoints if  $\pi_l = \pi_{m+1} \pm 1$  and  $\pi_m = \pi_{l-1} \pm 1$ . Finally, if removing two breakpoints is not possible, the algorithm removes only one, with either a transposition or a reversal. A transposition removes one breakpoint if one of the following three circumstances occurs: (i)  $\pi_i = \pi_{k-1} \pm 1$ , (ii)  $\pi_j = \pi_{i-1} \pm 1$ , or (iii)  $\pi_k = \pi_{j-1} \pm 1$ . A reversal removes one breakpoint if either  $\pi_l = \pi_{m+1} \pm 1$  or  $\pi_m = \pi_{l-1} \pm 1$ .

Note that the search for any of these operations (that remove three, two or one breakpoint) is  $O(n)$ , it is enough to vary one of the indices to find the others.

The results for all algorithms in Table 3.1 are shown in figures 3.1 to 3.9. Each figure relates to a specific genome rearrangement problem and presents four types of graphics.

- The first type reports in percentages how many times the heuristic improved the initial solutions. Keep in mind that this graph does not make any assumption about individual improvements. It simply informs the percentage of the 1,000 instances of each size that had the initial solution improved.
- The second type complements the data already available in the first graph. It presents the average improvement of the solutions when the initial solution has improved. In other words, every time our heuristic improves the initial solution provided by the algorithm, we calculate the difference in size between the sequence produced by our implementation and the initial solution. The graph plots the average value.
- The third graph plots the average distance over all permutations of a given size after we run our heuristic on them. We also add a lower bound (**LB**) for comparison purposes.
- The fourth graph plots the average approximation ratio after we run our heuristic on them. To plot this graph, we used the same lower bound used in the third graph.

The performance of our method varies depending on the algorithm that produces the initial solution. The main explanation for the variation is how close to the optimal solution the result produced by the algorithm is. Usually, the closer the initial solutions are to the optimal solution, the more difficult it is to improve them.

As an example for the sorting by transposition problem, many efforts have been made in order to generate algorithms with good practical results. Therefore, the transposition problem is the one such that our heuristics leads to lowest improvements.

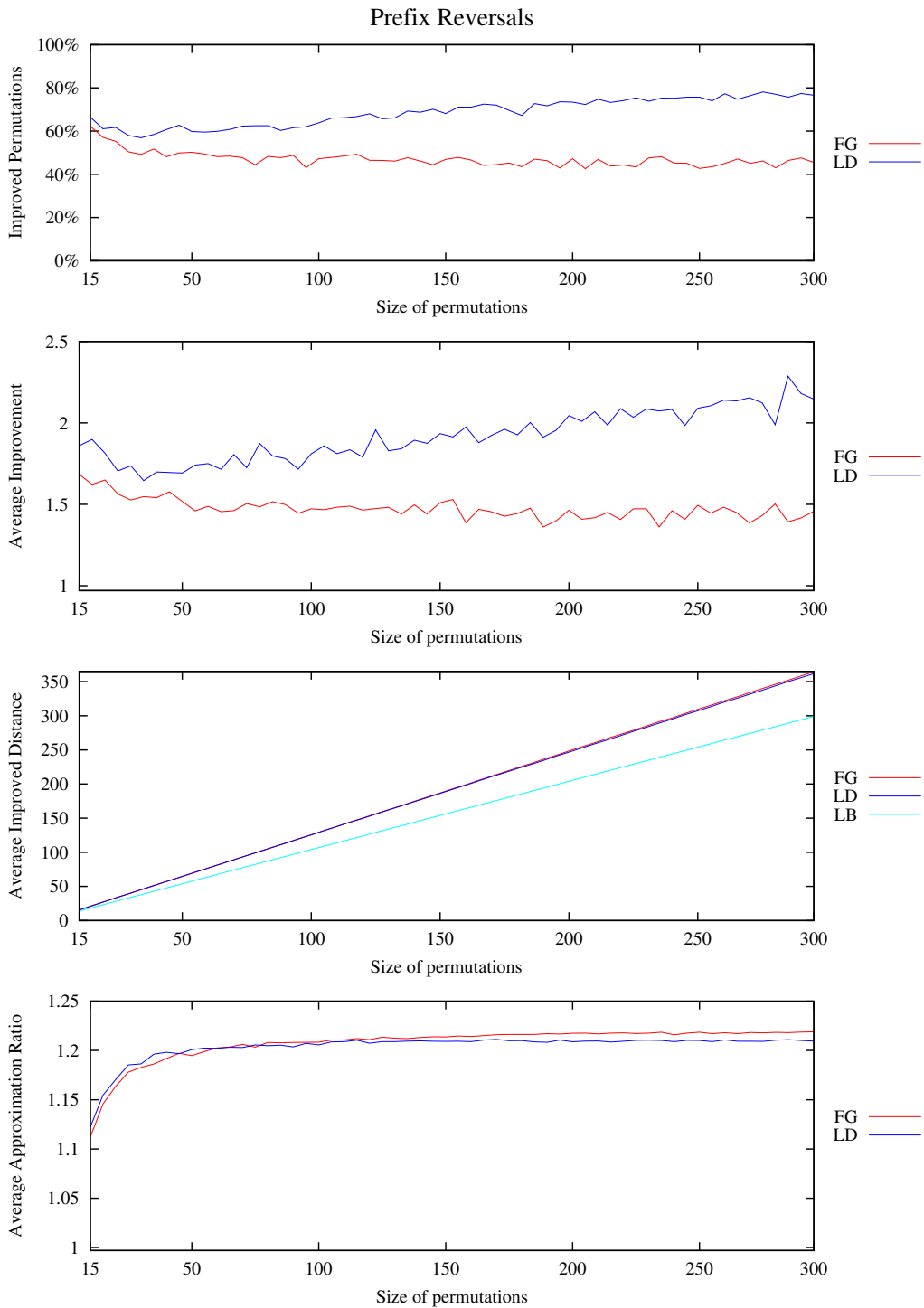


Figure 3.1: Results regarding the algorithms for the problem of sorting by prefix reversals. We were able to improve 68.7% of the solutions provided by LD, whereas 47.1% of FG solutions were improved. In addition, the average improvement for LD was around 1.9, while it was around 1.5 for FG. The lower bound used in our analysis is  $d_{pr}(\pi) \geq b_{pr}(\pi) - 1$ .



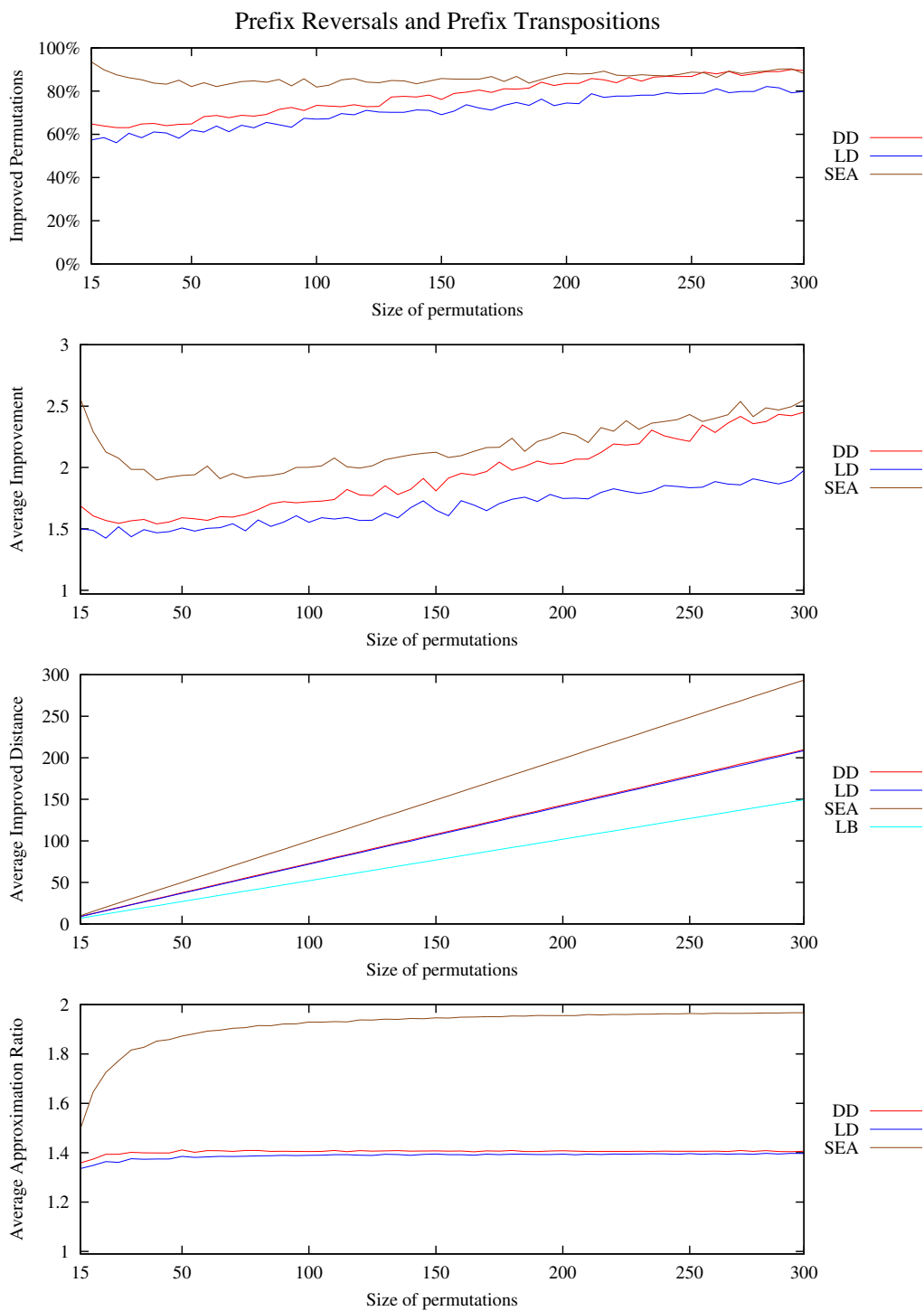


Figure 3.2: Results regarding the algorithms for the problem of sorting by prefix reversals and prefix transpositions. We observe that the algorithm SEA benefits more from our approach, followed by DD. The LD algorithm benefits less than the others. Overall, we were able to improve 86.1% of the solutions provided by SEA, 77.5% of the solutions provided by DD and 70.8% of the solutions provided by LD. The average improvement is around 2.2, 1.9 and 1.7 for SEA, DD and LD, respectively. The lower bound used in our analysis is  $d_{prpt}(\pi) \geq \left\lceil \frac{b_{prpt}(\pi)-1}{2} \right\rceil$ .

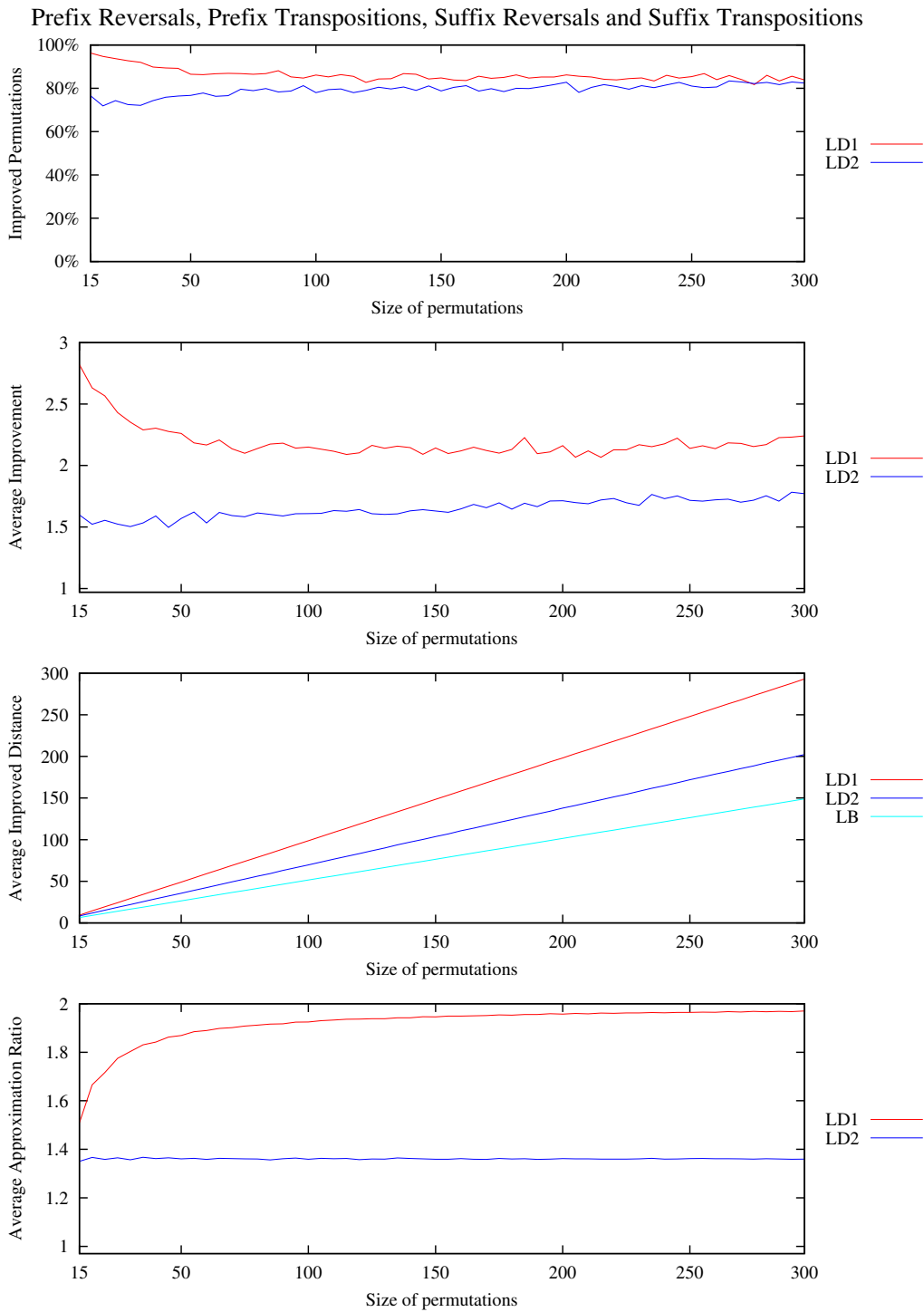


Figure 3.3: Results regarding the algorithms for the problem of sorting by prefix reversals, prefix transpositions, suffix reversals and suffix transpositions. LD2 is an improved version of LD1 and provides shorter solutions in almost every case. We were able to improve 86.2% of the solutions provided by LD1 and 79.4% of the solutions proved by LD2. In addition, the average improvement for LD1 and LD2 is 2.2 and 1.6, respectively. The lower bound used in our analysis is  $d_{prptsrst}(\pi) \geq \left\lceil \frac{b_{prptsrst}(\pi) - 2}{2} \right\rceil$ .

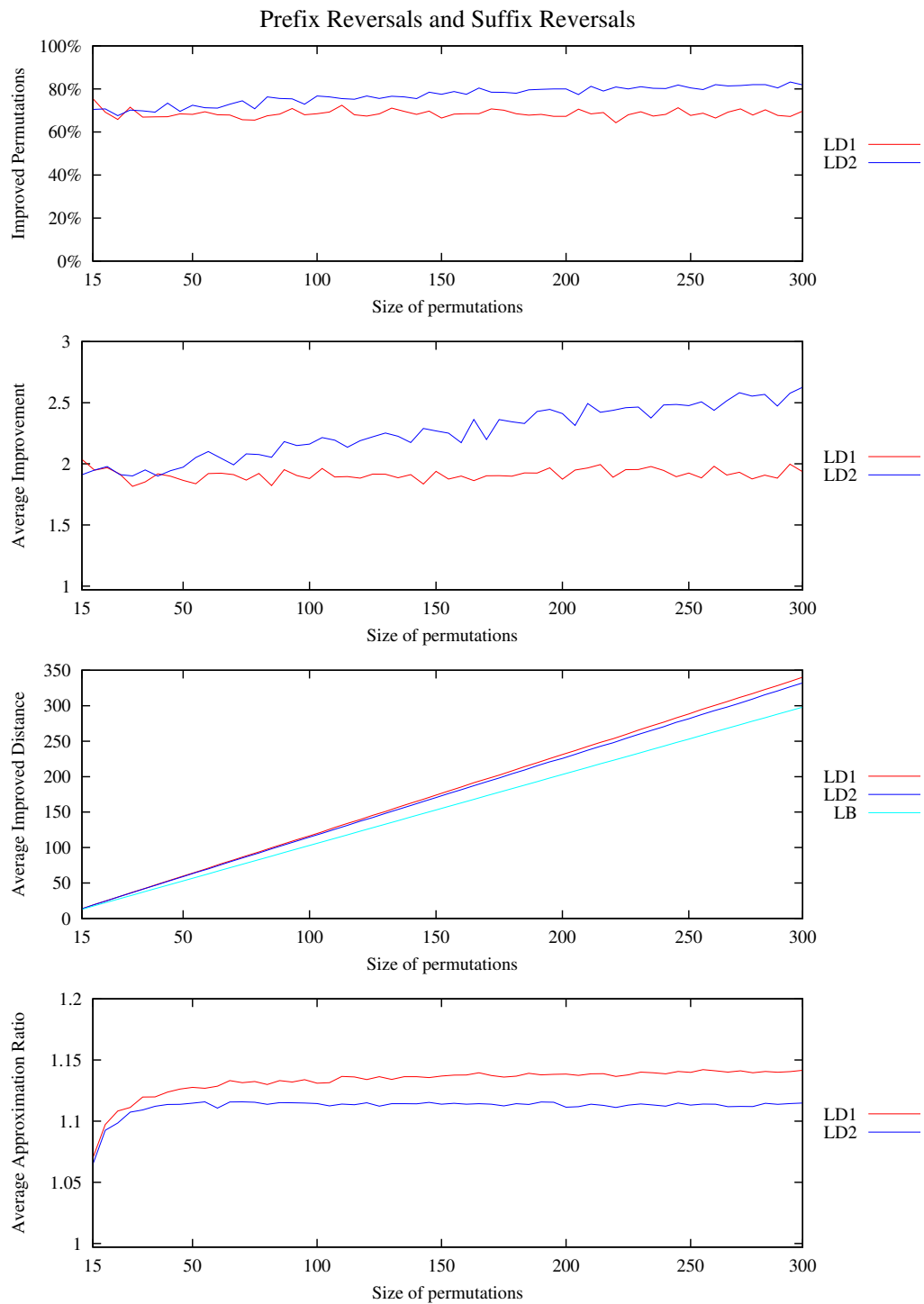


Figure 3.4: Results regarding the algorithms for the problem of sorting by prefix reversals and suffix reversals. LD2 is an improved version of LD1 and provides shorter solution in almost every case. So, it would be reasonable to expect that LD1 should benefit more from our approach than LD2. However, it is not the case since we were able to improve 68.6% of the solutions provided by LD1 and 76.9% of the solutions provided by LD2. In addition, the average improvement for LD1 and LD2 is 1.9 and 2.3, respectively. The lower bound used in our analysis is  $d_{prsr}(\pi) \geq b_{prsr}(\pi) - 2$ .

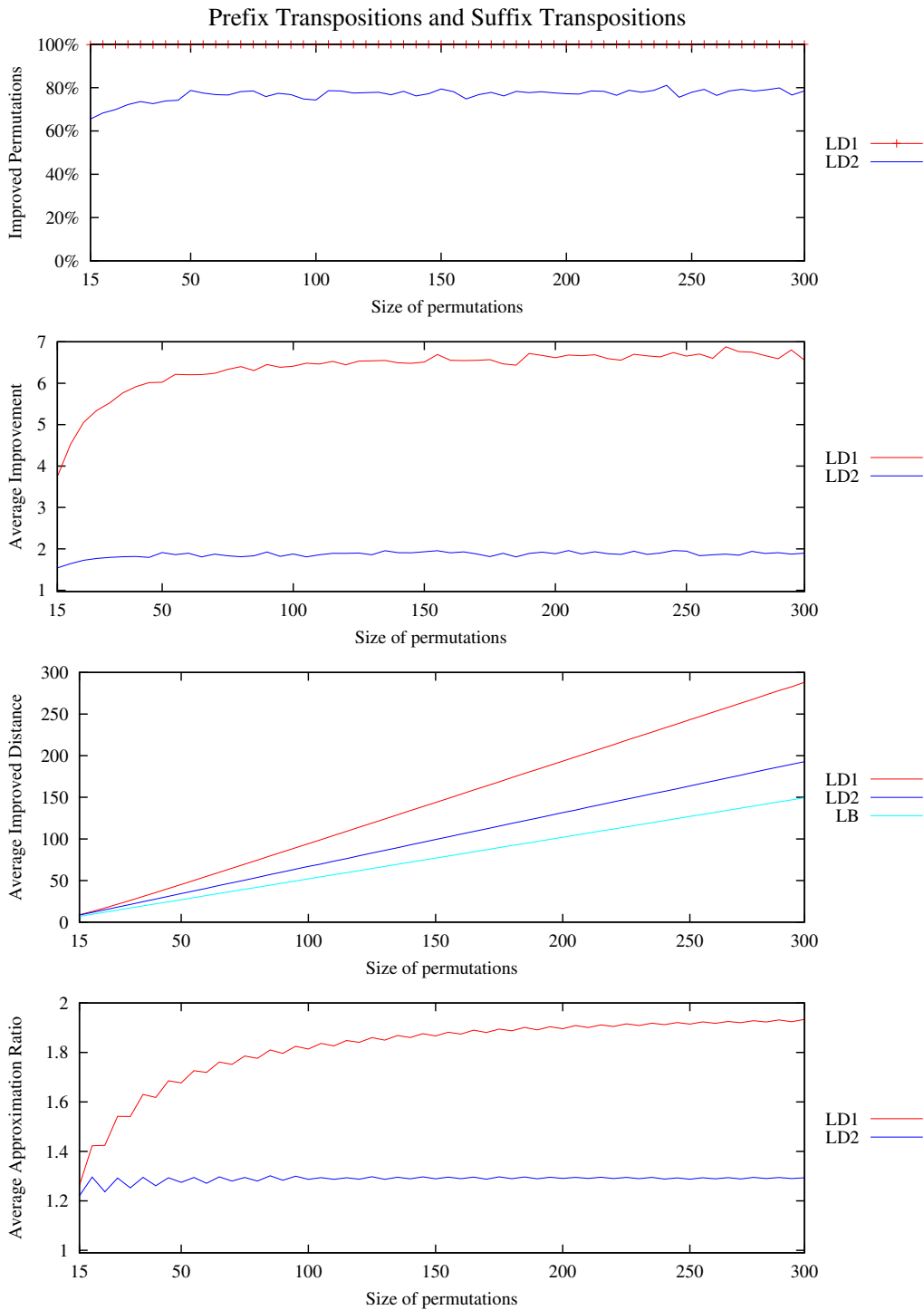


Figure 3.5: Results regarding the algorithms for the problem of sorting by prefix transpositions and suffix transpositions. We were able to improve almost 100.0% of the solutions provided by LD1 and 76.8% of the solutions provided by LD2. The average improvement is around 6.4 and 1.9 for LD1 and LD2, respectively. This discrepancy can be explained by the fact that LD2 is way better than LD1. As an example, for  $n = 300$ , LD2 can return solutions that have 80 less operations than LD1 in 86.2% of the cases. The lower bound used in our analysis is  $d_{ptst}(\pi) \geq \left\lceil \frac{b_{ptst}(\pi) - 2}{2} \right\rceil$ .

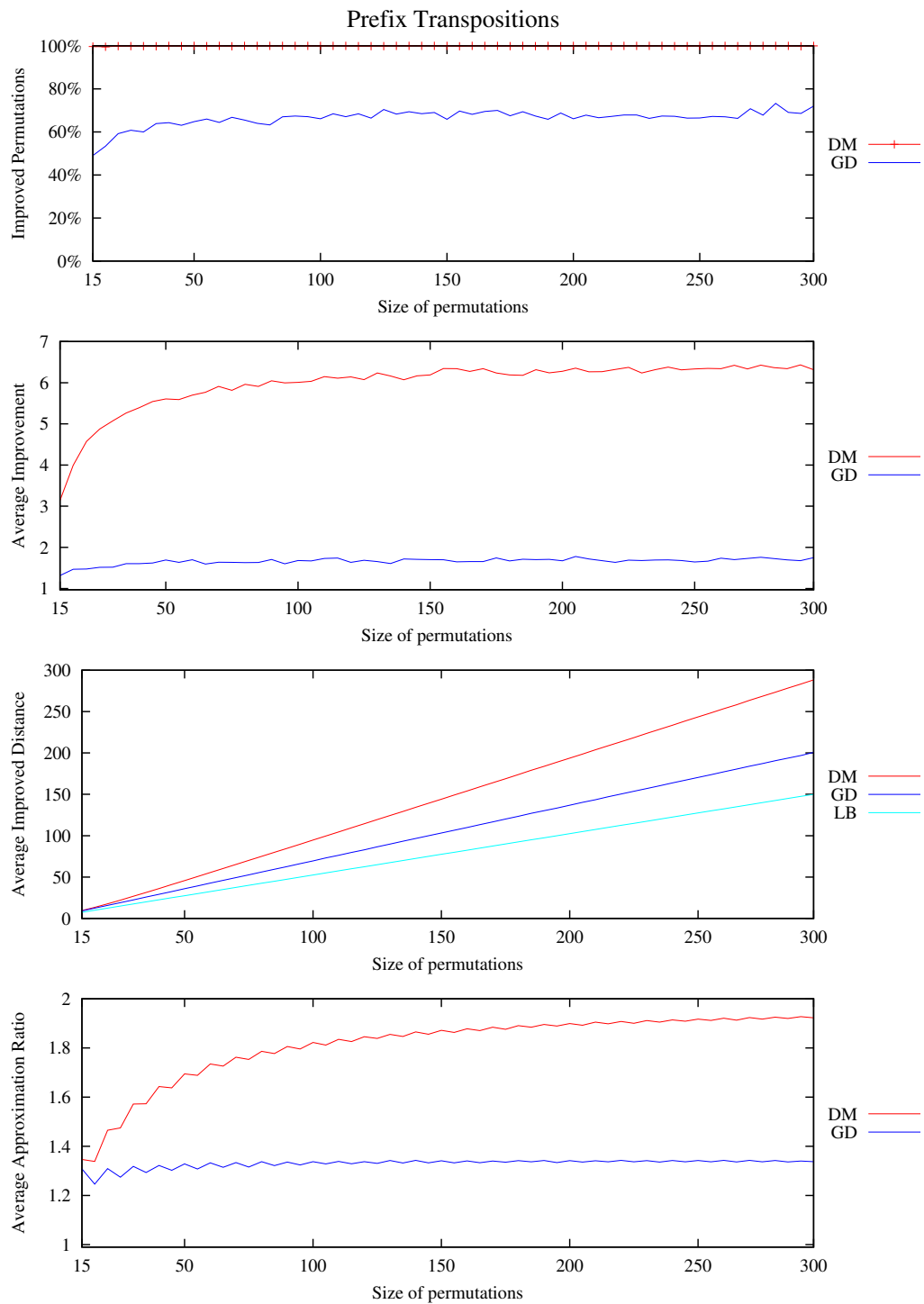


Figure 3.6: Results regarding the algorithms for the problem of sorting by prefix transpositions. We observe that the algorithm DM was improved in almost every case and the average improvement is around 6.0. The improvements for the GD algorithm were more modest, so that 66.4% of the solutions were improved and the average improvement is around 1.7. The lower bound used in our analysis is  $d_{pt}(\pi) \geq \left\lceil \frac{b_{pt}(\pi)-1}{2} \right\rceil$ .

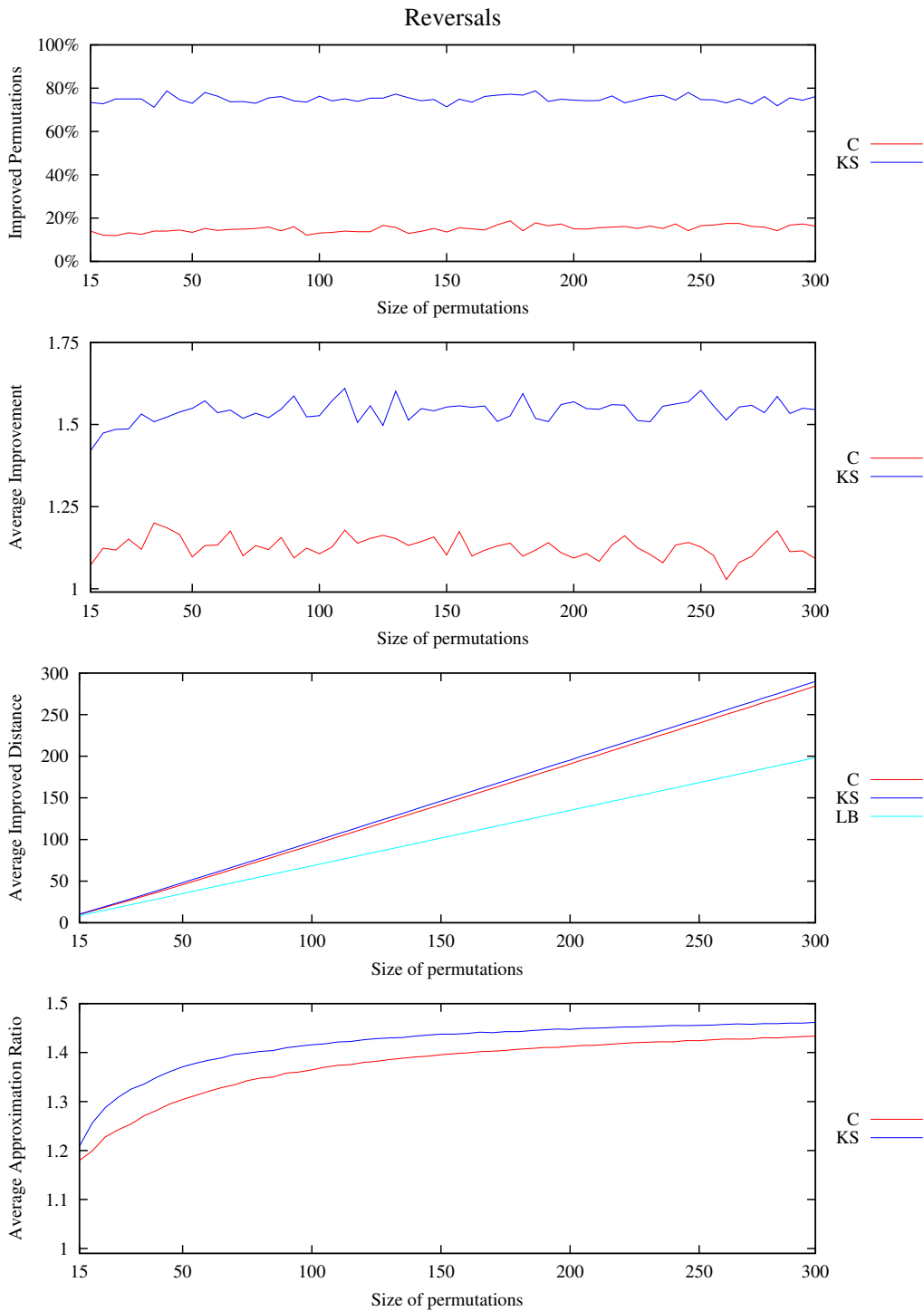


Figure 3.7: Results regarding the algorithms for the problem of sorting by reversals. We were able to improve 74.9% of the solutions provided by KS and only 15.1% of the solutions provided by C. The average improvement was about 1.5 and 1.1 for KS and C, respectively. The lower bound used in our analysis was developed by Kececioğlu and Sankoff<sup>21</sup>:  $d_r(\pi) \geq \left\lceil \frac{1}{2}m + \frac{2}{3}(b_r(\pi) - m) \right\rceil$ , where at least  $m$  reversals are guaranteed to eliminate  $2m$  breakpoints.

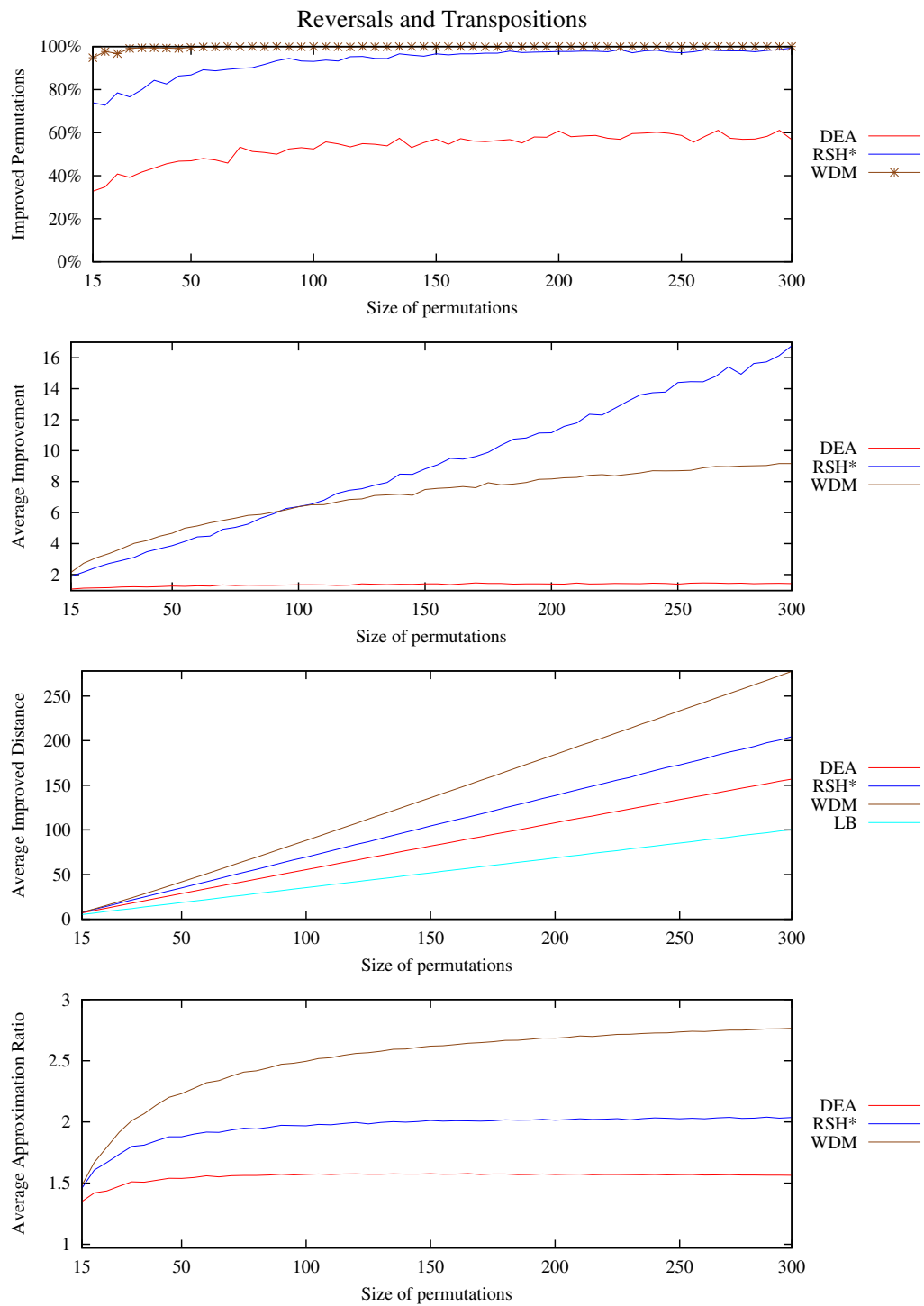


Figure 3.8: Results regarding the algorithm for the problem of sorting by reversals and transpositions. We observe that the algorithm WDM was improved in almost every case and the average improvement was about 6.9. The algorithm RSH\* was improved in 93.4% of the cases and the average improvement was about 9.1. The algorithm DEA was improved in 53.5% of the cases and the average improvement was about 1.3. The lower bound used in our analysis is  $d_{rt}(\pi) \geq \left\lceil \frac{b_{rt}(\pi)}{3} \right\rceil$ .

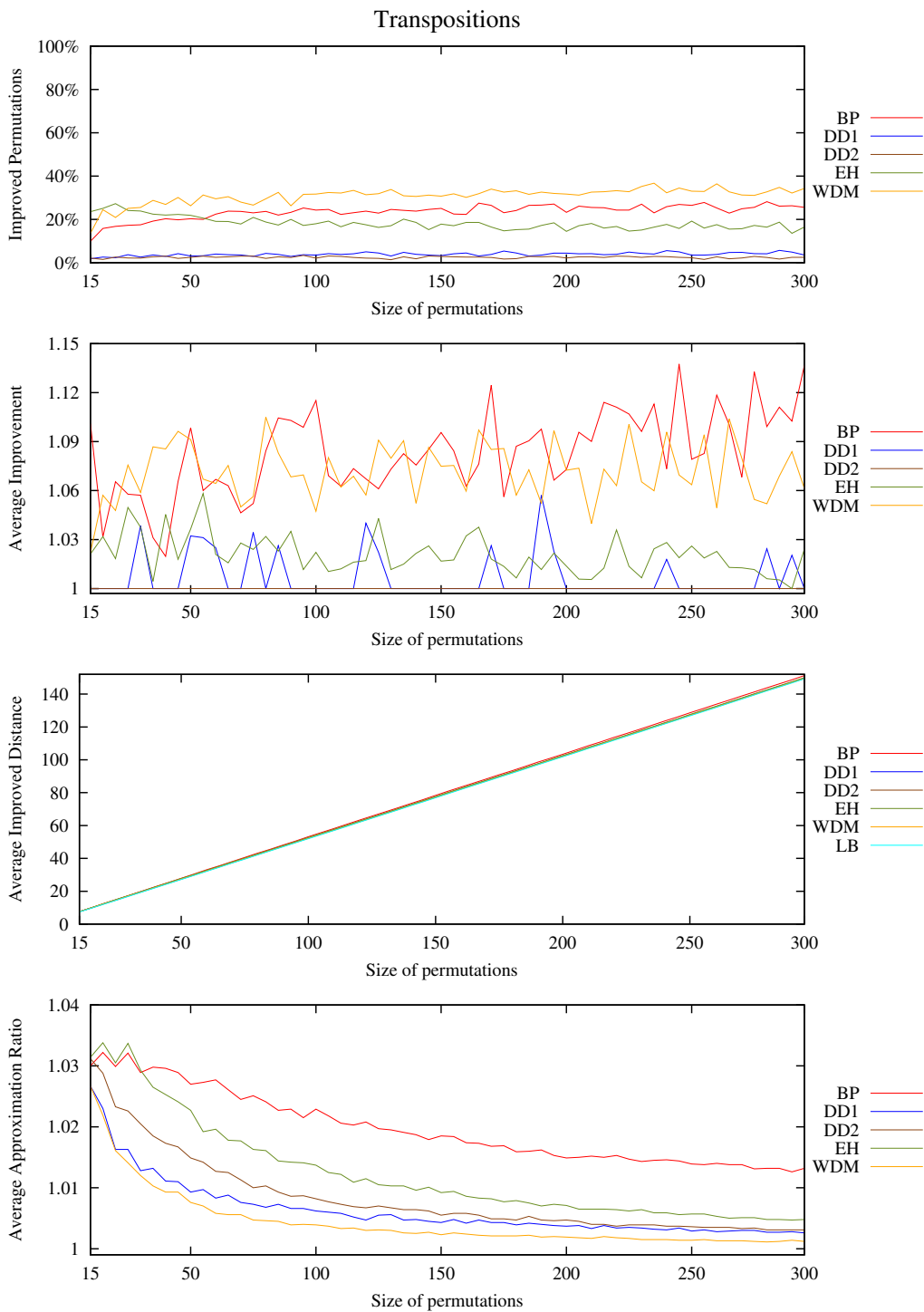


Figure 3.9: Results regarding the 5 algorithms for the problem of sorting by transpositions. These algorithms did not benefit too much from our heuristic and we believe that such modest results are due to the fact that they provide near-optimal solutions. The second graph shows that the algorithms are very close to the lower bound on average. The lower bound used in our analysis is  $d_t(\pi) \geq \left\lceil \frac{n+1-c_{\text{odd}}(\pi)}{2} \right\rceil$ , where  $c_{\text{odd}}$  is the amount of cycles with an odd number of black edges in the directed edge-colored cycle graph developed by Bafna and Pevzner [10].



As we observe in the second graph in Figure 3.9, on average the improved results are almost equal to the lower bound, which shows that not much could be done to improve them.

Table 3.2 summarizes some key points for the sorting by transpositions problem. It reports the average distance for each approximation algorithm for the sorting by transpositions problem. Observe that the algorithm WDM plus our heuristic is the approach that offers the best results on average. When we do not use the heuristic, the best algorithm is DD1. We highlight the best results on average for each size.

The results presented so far are averages. In Table 3.3, we present results concerning the fraction of the instances in which each program was the winner (provider of the shorter sequence that sorts the input permutation). This table reinforces the idea that our heuristic leads to the best practical result. Clearly, DD1 is the best algorithm among those that do not use our heuristic. When we consider the results with our heuristic, we observe that WDM was improved more often than the others, and in the end it generates the shortest sequence in more cases.

It is worth mentioning that after using our heuristic we were able to generate the best practical results to date for the sorting by transposition problem as well as the others.

The other rearrangement problems found larger improvements. Table 3.4 summarizes the results shown in the graphs by considering all the 58,000 test cases at once. The column “(%) of Improvement” shows the percentage of improved solutions and the column “Average Improvement” complements that information by reporting the average difference in size between the sequence produced by our implementation and the initial solution on cases when our heuristic improves the initial solution.

Table 3.2: Average distance for each approximation algorithm for the sorting by transpositions problem. We highlight the best results produced both with and without our heuristic.

	Code	Size of Permutation					
		50	100	150	200	250	300
Original	<b>BP</b>	25.6	50.9	76.0	101.2	126.4	151.5
	<b>DD1</b>	25.0	49.9	74.8	99.8	124.7	149.7
	<b>DD2</b>	25.1	50.0	74.9	99.9	124.7	149.8
	<b>EH</b>	25.5	50.4	75.3	100.3	125.1	150.2
	<b>WDM</b>	25.2	50.0	74.9	99.9	124.8	149.8
Improved	<b>BP</b>	25.4	50.6	75.7	100.9	126.1	151.3
	<b>DD1</b>	24.9	49.8	74.7	99.7	124.7	149.7
	<b>DD2</b>	25.1	49.9	74.8	99.8	124.7	149.7
	<b>EH</b>	25.2	50.2	75.1	100.1	125.0	150.0
	<b>WDM</b>	24.9	49.7	74.6	99.6	124.4	149.5
<b>Lower Bound</b>		24.6	49.5	74.4	99.4	124.3	149.3

Table 3.3: Percentage of instances in which each program yielded the best results for the sorting by transpositions problem. Columns do not add up to 100% because of ties. Best results highlighted.

	Code	Size of Permutation					
		50	100	150	200	250	300
Original	<b>BP</b>	43.2	26.0	18.7	15.7	12.1	9.2
	<b>DD1</b>	85.3	78.1	75.1	71.5	69.2	70.9
	<b>DD2</b>	73.2	67.1	64.4	64.2	64.3	62.9
	<b>EH</b>	45.5	38.4	36.2	34.8	38.3	36.9
	<b>WDM</b>	65.8	62.2	61.2	60.2	60.9	60.1
Improved	<b>BP</b>	53.7	33.9	24.1	20.7	14.7	12.0
	<b>DD1</b>	86.6	78.0	75.0	70.5	68.6	69.2
	<b>DD2</b>	73.3	68.3	64.6	64.4	63.8	61.1
	<b>EH</b>	62.7	50.7	46.6	48.1	49.8	47.9
	<b>WDM</b>	91.1	88.6	86.7	86.1	89.4	87.0

The columns “Average Original Distance” and “Average Improved Distance” show the Average distance for each approximation algorithm both with and without our heuristic, respectively. The last two columns “Average Improved Ratio” and “Maximum Improved Ratio” show how the approximation ratio behaves. The approximation ratio is calculated with the lower bounds shown in figures 3.1 to 3.9.

Table 3.4 allows us to check which algorithm is producing the best results. Usually, the algorithm that produces the best initial solution leads to the best result after applying our heuristic, except for the sorting by transpositions problem that we have mentioned before.

For the sorting by prefix reversals problem, LD has a slight advantage and benefits more from our heuristic than FG, but it is not enough to make the difference very remarkable. In the end, the approximation ratios are very similar.

For the sorting by prefix reversals and prefix transpositions problem, LD is the best before and after applying our heuristic, which is unexpected since this is a 3-approximation algorithm and DD guarantees the approximation ratio of 2. Observe, however, that the maximum approximation ratio presented by LD in our experiments after applying our heuristic is 1.889, which suggests that one could work on this algorithm in order to decrease the theoretical approximation bound.

For the sorting by prefix reversals and suffix reversals problem, the best algorithm (coded as LD2) benefits more from our heuristic. For the sorting by prefix reversals, prefix transpositions, suffix reversals and suffix transpositions problem, the sorting by prefix transpositions and suffix transpositions problem and the sorting by prefix transpositions problem, the worst algorithms (coded as LD1) are the ones which benefit more from our heuristic. Indeed, the solutions provided by them were improved in many cases (see column “(%) of Improvement”).

Table 3.4: Summary of the results produced for each algorithm. The values are averages over all the 58,000 test instances of sizes ranging from 15 to 300 in intervals of 5, with 1,000 permutations of each size.

<b>Problem</b>	<b>Code</b>	<b>(%) of Improvement</b>	<b>Average Improvement</b>	<b>Average Original Distance</b>	<b>Average Improved Distance</b>	<b>Average Improved Ratio</b>	<b>Maximum Improved Ratio</b>
Prefix Reversal	FG	47.09	1.47	190.73	190.03	1.207	1.409
	LD	68.68	1.92	190.43	189.10	1.204	1.419
Prefix Reversal and Prefix Transposition	DD	77.52	1.93	111.17	109.65	1.404	1.857
	LD	70.80	1.67	109.80	108.60	1.388	1.857
	SEA	86.13	2.18	153.76	151.88	1.916	2.111
Prefix Reversal, Prefix Transposition, Suffix Reversal and Suffix Transposition	LD1	86.18	2.19	152.92	151.02	1.917	2.000
	LD2	79.34	1.65	106.74	105.43	1.361	1.875
Prefix Reversal and Suffix Reversal	LD1	68.57	1.91	178.17	176.85	1.133	1.391
	LD2	76.93	2.26	174.89	173.14	1.112	1.321
Prefix Transposition and Suffix Transposition	LD1	99.99	6.36	153.05	146.69	1.818	1.973
	LD2	76.77	1.86	102.16	100.73	1.288	1.667
Prefix Transposition	DM	99.95	5.98	153.04	147.07	1.811	1.978
	GD	66.44	1.66	106.00	104.89	1.330	1.588
Reversal	C	15.08	1.13	145.22	145.05	1.373	1.489
	KS	74.93	1.54	150.08	148.93	1.417	1.556
Reversal and Transposition	DEA	53.54	1.35	83.49	82.76	1.555	1.875
	RSH*	93.42	9.06	114.57	105.89	1.960	2.315
	WDM	99.71	6.95	146.52	139.58	2.513	2.876
Transposition	BP	23.53	1.08	79.77	79.52	1.020	1.250
	DD1	3.89	1.01	78.58	78.54	1.006	1.167
	DD2	2.47	1.00	78.66	78.63	1.008	1.222
	EH	18.26	1.02	79.05	78.86	1.012	1.222
	WDM	30.90	1.07	78.73	78.40	1.004	1.250

However, even after the improvement, these algorithms performed poorly when compared to their counterparts as we assess them by the average distance columns.

For the sorting by reversals problem, the solutions provided by the best algorithm (coded as C) were improved in 15.08% of the solutions and the average improvement was around 1.13. The KS algorithm benefited more from our heuristic, but it was not enough to outdo C.

For the sorting by reversals and transpositions problem, the best algorithm (coded as DEA) was implemented by ourselves based on a greedy removal of breakpoints. Our algorithm did not benefit too much from our heuristic like the others, but in the end it keeps leading to the best results. The maximum improved ratio for DEA was 1.875 in our experiments, which is very far from the theoretical approximation ratio 3. This suggests that some work could be done in order to improve the theoretical approximation ratio.

## 3.6 Conclusions

We presented a general heuristic that improves an initial solution for several rearrangement problems. Our method selects sub-sequences from the initial solution and tries to improve the solution by replacing those sub-sequences with shorter solutions retrieved from a database.

We applied our heuristic to the solutions provided by 23 approximation algorithms. We observed great improvements when we considered only prefix transpositions. Taking the 2 algorithms implemented for this problem into account, we observed that around 83% of the cases were improved.

For most of the sorting problems that allow reversals, we observed that our heuristic performance is close to that observed for the sorting by prefix transpositions problem. For example, the initial solution was improved in more than 70% of the cases in the following problems: sorting by prefix reversals and prefix transpositions, and sorting by prefix reversals and suffix reversals, sorting by prefix reversals, prefix transpositions, suffix reversals and suffix transpositions. When only reversals were considered, we were able to improve around 45% of the initial solutions and when only prefix reversals were considered, around 58% of the initial solutions were improved.

For the sorting by reversals and transpositions problem, we were able to improve 82% of the test instances. Finally, for the sorting by transpositions problem, we were able to improve more than 15% of the test cases. Around 94% of these improvements occurred by just one unit. The sorting by transpositions problem is a very special case because many efforts have been made to generate algorithms with good results in practice and some of these algorithms provide results that equal the optimum solutions in many cases. Although the improvements were comparatively lower than those for other problems, we believe they are relevant because improvements on the problem of sorting by transpositions are quite rare and hard to obtain.

## Chapter 4

# On Alternative Approaches for Approximating the Transposition Distance \*

**Abstract:** We study the problem of sorting by transpositions, which consists in computing the minimum number of transpositions required to sort a permutation. This problem is NP-hard and the best approximation algorithms for solving it are based on a standard tool for attacking problems of this kind, the cycle graph. In an attempt to bypass it, some researches posed alternative approaches. In this paper, we address three algorithms yielded by such approaches: a 2.25-approximation algorithm based on breakpoint diagrams, a 3-approximation algorithm based on permutation codes, and a heuristic based on longest increasing subsequences. Regarding the 2.25-approximation algorithm, we show that previous experimental data on its approximation ratio are incorrect. Regarding the 3-approximation algorithm, we close a missing gap on the proof of its approximation ratio and we show a way to run it in  $O(n \log n)$  time. Regarding the heuristic, we propose a minor adaptation that allow us to prove an approximation bound of 3. We present experimental data obtained by running these algorithms for all permutations with up to 13 elements and by running these algorithms and the best known algorithms based on the cycle graph for large permutations. The data indicate that the 2.25-approximation algorithm is the best of the algorithms based on alternative approaches and that it is the only one comparable to the algorithms based on the cycle graph.

---

\* *Gustavo Rodrigues Galvão and Zanoni Dias. On Alternative Approaches for Approximating the Transposition Distance. Journal of Universal Computer Science. Volume 20, No. 9, pp. 1259-1283, 2014. Copyright 2014 J.UCS. DOI: <http://dx.doi.org/10.3217/jucs-020-09-1259>*

## 4.1 Introduction

A transposition is the rearrangement event that switches the location of two contiguous portions of a genome. The problem of computing the transposition distance between two genomes consists in finding the minimum number of transpositions needed to transform one genome into the other. Such problem finds application in comparative genomics because the transposition distance can be used to estimate the evolutionary distance between two genomes.

Representing the order of the genes in the genomes as permutations, that problem can be reduced to the combinatorial problem of finding the minimum number of transpositions required to sort a permutation, which is referred to as the Problem of Sorting by Transpositions. This problem was recently proven to be NP-hard [22], therefore it is not likely that a polynomial-time algorithm exists. It was introduced by Bafna and Pevzner [10], who presented a 1.5-approximation algorithm that runs in quadratic time. Later, Elias and Hartman [48] improved the approximation bound to 1.375, maintaining quadratic time complexity. Recently, Dias and Dias [36, 37] presented improved versions of Bafna and Pevzner's algorithm and Elias and Hartman's algorithm, which keep the original approximation ratios, and these have been the best known algorithms for the problem of sorting by transpositions.

These approximation algorithms, as well as others [30, 68, 73] with relatively low approximation ratios (*i.e.* less or equal than 1.5), are based on a structure named the cycle graph. This structure is regarded as complex by some authors, therefore they posed alternative approaches in order to bypass it. For a detailed literature survey, the reader is referred to the book of Fertin and colleagues [54].

Walter, Dias, and Meidanis [124] presented a 2.25-approximation algorithm based on a structure named the breakpoint diagram that runs in  $O(n^2)$  time. They ran some experiments in order to observe the approximation ratio of their algorithm in practice, but it was not conclusive. Benoît-Gagné and Hamel [14] developed a 3-approximation algorithm based on permutation codes that runs in  $O(n^2)$  time. According to them, although there exist better algorithms with respect to approximation ratio, their algorithm is faster than any existing one. Besides, their experimental results suggested that the approximation ratio of their algorithm may be lowered. Guyer, Heath, and Vergara [69] devised a heuristic based on the longest increasing subsequence in a permutation that runs in  $O(n^5 \log n)$  time. The experiments they performed suggested that it has the potential to produce near-optimal results.

In this work, we review these algorithms in order to improve their analyses, providing theoretical strengthening whenever possible, and to determine whether they are good alternatives to the algorithms based on the cycle graph. Regarding Walter, Dias, and Meidanis' algorithm [124], we show that previous experimental data on its approximation ratio are incorrect, and then we present new experimental data suggesting that its approximation ratio may be lowered to 2. Regarding Benoît-Gagné and Hamel's algorithm [14], we close a missing gap on the proof of its approximation

ratio and we demonstrate a way to run it in  $O(n \log n)$  time. On the other hand, we present experimental data that contradicts Benoît-Gagné and Hamel's hypothesis that its approximation ratio may be lowered. Regarding Guyer, Heath, and Vergara's heuristic [69], we propose a minor adaptation that allow us to prove an approximation bound of 3. On the other hand, we present experimental data that indicates this algorithm does not produce near-optimal results. Finally, we compare these three algorithms to the best known algorithms based on the cycle graph.

The remainder of this paper is organized as follows. In the next section, we give the basic definitions and notation of the paper. In Section 4.3, we briefly describe the algorithms studied in this paper, close a missing gap on Benoît-Gagné and Hamel's proof [14] for the approximation ratio of their algorithm (Lemma 14), and demonstrate that a constrained version of Guyer, Heath, and Vergara's heuristic [69] still has an approximation bound of 3 (Theorem 4). In Section 4.4, we show how to compute the permutation codes in  $O(n \log n)$  time, what allow us to implement Benoît-Gagné and Hamel's algorithm [14] in such a way that its running time becomes  $O(n \log n)$ . In Section 4.5, we present experimental results along with a discussion on the performance of the algorithms in practice. In the last section, we conclude the paper.

## 4.2 Preliminaries

We represent genomes as permutations, where genes appear as elements. A *permutation*  $\pi$  is a bijection of  $\{1, 2, \dots, n\}$  onto itself. The group of all permutations of  $\{1, 2, \dots, n\}$  is denoted by  $S_n$  and we write a permutation  $\pi \in S_n$  as  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . Sometimes, we extend it with two elements  $\pi_0 = 0$  and  $\pi_{n+1} = n + 1$ . The extended permutation will still be called  $\pi$ .

A *transposition* is an operation  $\rho(i, j, k)$ ,  $1 \leq i < j < k \leq n + 1$ , that moves blocks of contiguous elements of a permutation  $\pi \in S_n$  in such way that  $\rho(i, j, k) \cdot (\pi_1 \dots \pi_{i-1} \pi_i \dots \pi_{j-1} \pi_j \dots \pi_{k-1} \pi_k \dots \pi_n) = (\pi_1 \dots \pi_{i-1} \pi_j \dots \pi_{k-1} \pi_i \dots \pi_{j-1} \pi_k \dots \pi_n)$ . The Problem of Sorting by Transpositions consists in finding the minimum number of transpositions that transform a permutation  $\pi \in S_n$  into the identity permutation  $I_n = (1 \ 2 \ \dots \ n)$ . This number is known as the *transposition distance* of a permutation  $\pi$  and it is denoted by  $d(\pi)$ .

Given a permutation  $\pi \in S_n$ , a *breakpoint* is a pair of adjacent elements that are not consecutive, that is, a pair  $(\pi_i, \pi_{i+1})$  such that  $\pi_{i+1} - \pi_i \neq 1$ ,  $0 \leq i \leq n$ . The number of breakpoints of  $\pi$  is denoted by  $b(\pi)$ . Note that  $I_n$  is the only permutation in  $S_n$  having zero breakpoints. Since a transposition can remove at most three breakpoints, we can state the following lemma.

**Lemma 12.** [10] For any permutation  $\pi \in S_n$ ,  $d(\pi) \geq \frac{b(\pi)}{3}$ .

A *strip* of a permutation  $\pi$  is a maximal series of consecutive elements without a breakpoint. We denote the number of strips in  $\pi$  by  $s(\pi)$ .

**Example 5.** Let  $\pi = (0\ 4\ 5\ 2\ 3\ 1\ 6)$  be the extended permutation of  $(4\ 5\ 2\ 3\ 1)$ . We have that the pairs  $(0, 4)$ ,  $(5, 2)$ ,  $(3, 1)$ , and  $(1, 6)$  are breakpoints, thus  $b(\pi) = 4$ . We also have that  $4, 5, 2, 3$ , and  $1$  are strips of  $\pi$ , thus  $s(\pi) = 3$ .

Let  $\pi \in S_n$ ,  $\pi \neq I_n$ ,  $s_1$  be the first strip of  $\pi$ , and  $s_m$  be the last strip of  $\pi$ ,  $m \leq n$ . If we assume that  $\pi_1 = 1$ , we can transform  $\pi$  into a permutation  $\sigma \in S_{n-|s_1|}$  such that  $\sigma_i = \pi_i - |s_1|$ ,  $i > |s_1|$ . It is not hard to see that  $d(\pi) = d(\sigma)$ . An analogous argument can be used to show that we can transform  $\pi$  into a permutation  $\gamma \in S_{n-|s_m|}$  such that  $d(\pi) = d(\gamma)$  if  $\pi_n = n$ . These transformations are referred to as reductions, and we call *irreducible* any permutation in which such reductions cannot be applied. Furthermore, we denote by  $S_n^*$  the set formed by all irreducible permutations of  $S_n$ .

**Lemma 13.** For any permutation  $\pi \in S_n^*$ ,  $s(\pi) = b(\pi) - 1$ .

*Proof.* Let  $s_1, s_2, \dots, s_{s(\pi)}$  be the strips of a permutation  $\pi \in S_n^*$ . The last element of strip  $s_i$  and the first element of strip  $s_{i+1}$ ,  $1 \leq i \leq s(\pi) - 1$ , form a breakpoint, and the pairs  $(\pi_0, \pi_1)$  and  $(\pi_n, \pi_{n+1})$  are always breakpoints on an irreducible permutation. Therefore, we have that  $b(\pi) = s(\pi) + 1$  and the claim follows.  $\square$

Given a permutation  $\pi \in S_n$ , the *left and right codes of an element*  $\pi_i$ , denoted  $lc(\pi_i)$  and  $rc(\pi_i)$  respectively, are defined as  $lc(\pi_i) = |\{\pi_j : \pi_j > \pi_i \text{ and } 1 \leq j \leq i - 1\}|$  and  $rc(\pi_i) = |\{\pi_j : \pi_j < \pi_i \text{ and } i + 1 \leq j \leq n\}|$ . The *left (resp., right) code of a permutation*  $\pi$  is then defined as the sequence of lc's (resp., rc's) of its elements, and it is denoted by  $lc(\pi)$  (resp.,  $rc(\pi)$ ).

Let us call *plateau* any maximal length sequence of contiguous elements in a number sequence that have the same nonzero value. The number of plateaux in a code  $c$  is denoted  $p(c)$ . We denote by  $p(\pi)$  the minimum of  $p(lc(\pi))$  and  $p(rc(\pi))$ . Note that  $I_n$  is the only permutation in  $S_n$  having zero plateaux.

**Example 6.** Let  $\pi = (5\ 3\ 2\ 4\ 1)$ . We have that  $lc(\pi) = lc(\pi_1)\ lc(\pi_2)\ lc(\pi_3)\ lc(\pi_4)\ lc(\pi_5) = 0\ 1\ 2\ 1\ 4$ , and  $rc(\pi) = rc(\pi_1)\ rc(\pi_2)\ rc(\pi_3)\ rc(\pi_4)\ rc(\pi_5) = 4\ 2\ 1\ 1\ 0$ . Then,  $p(\pi) = \min\{p(lc(\pi)), p(rc(\pi))\} = \min\{4, 3\} = 3$ .

An *increasing subsequence* of a permutation  $\pi$  is a subsequence  $\pi_{i_1}\ \pi_{i_2}\ \dots\ \pi_{i_j}$  of nonnecessarily contiguous elements of  $\pi$  such that for all  $k$ ,  $0 < k < j$ , we have  $i_k < i_{k+1}$  and  $\pi_{i_k} < \pi_{i_{k+1}}$ . A *longest increasing subsequence* is an increasing subsequence of  $\pi$  of maximum length. The set of the elements belonging to a longest increasing subsequence is denoted by  $LIS(\pi)$ . It is easy to see that, for any  $\pi \in S_n$ ,  $|LIS(\pi)| = n$  if and only if  $\pi = I_n$ .

**Example 7.** Let  $\pi = (2\ 3\ 1\ 5\ 4)$ . The increasing subsequences  $2\ 3\ 5$  and  $2\ 3\ 4$  are maximal, therefore either  $LIS(\pi) = \{2, 3, 5\}$  or  $LIS(\pi) = \{2, 3, 4\}$ .



## 4.3 Algorithms

The following sections describe three algorithms for sorting by transpositions based on alternative approaches, namely Walter, Dias, and Meidanis' 2.25-approximation algorithm [124], Benoît-Gagné and Hamel's 3-approximation algorithm [14], and a constrained version of Guyer, Heath, and Vergara's heuristic [69]. Moreover, Section 4.3.2 contains the missing proof for the approximation ratio of Benoît-Gagné and Hamel's algorithm [14] and Section 4.3.3 contains the demonstration that the constrained version of Guyer, Heath, and Vergara's heuristic has an approximation bound of 3.

### 4.3.1 Algorithm based on the breakpoint diagram

Walter, Dias, and Meidanis [124] developed an approximation algorithm based on breakpoints. We will not discuss in detail how this algorithm works because it relies on an extensive case by case analysis that is not relevant to the discussion we set in this paper. Nevertheless, we present below a sketch of this algorithm (Algorithm 8) to make it clear why it is a 2.25-approximation algorithm.

---

**Algorithm 8:** Sketch of the 2.25-approximation algorithm proposed by Walter, Dias, and Meidanis [124].

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of transpositions applied for sorting  $\pi$ .

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq I_n$  do
3   if there exists a transposition  $\rho(i, j, k)$  that removes more than 1 breakpoint
   of  $\pi$  then
4      $\pi \leftarrow \rho(i, j, k) \cdot \pi$ ;
5      $d \leftarrow d + 1$ ;
6   else
7     Find up to 3 transpositions that removes at least 4 breakpoints when
     applied on  $\pi$ ;
8     Apply on  $\pi$  the transpositions found and update  $d$  accordingly;
9   end
10 end
11 return  $d$ ;

```

---

Note that, in the worst case, Algorithm 8 removes 4 breakpoints applying 3 transpositions. Thus, denoting by  $A_8(\pi)$  the number of transpositions applied by Algorithm 8 for sorting  $\pi$ , we have  $A_8(\pi) \leq \frac{3}{4}b(\pi)$ . Since  $d(\pi) \geq \frac{b(\pi)}{3}$  (Lemma 12), we conclude that Algorithm 8 is a 2.25-approximation. As for its time complexity, Walter, Dias and Meidanis [124] showed that it runs in  $O(n^2)$  time.

### 4.3.2 Algorithm based on permutation codes

Benoît-Gagné and Hamel [14] showed that it is always possible to decrease by one unit the number of plateaux of a (right or left) code by applying a transposition. Since  $p(\pi)$  represents the minimum value between  $p(lc(\pi))$  and  $p(rc(\pi))$ , it is possible to sort  $\pi$  applying at most  $p(\pi)$  transpositions. Thus, Benoît-Gagné and Hamel [14] proposed a simple algorithm for approximating the transposition distance that only computes the value of  $p(\pi)$ . Such algorithm is described below (Algorithm 9).

---

**Algorithm 9:** Algorithm proposed by Benoît-Gagné and Hamel [14].

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of transpositions applied for sorting  $\pi$ .

- 1 Compute  $lc(\pi)$ ;
  - 2 Compute  $rc(\pi)$ ;
  - 3  $i \leftarrow p(lc(\pi))$ ;
  - 4  $j \leftarrow p(rc(\pi))$ ;
  - 5  $d \leftarrow \min\{i, j\}$ ;
  - 6 **return**  $d$ ;
- 

Although Benoît-Gagné and Hamel [14] stated that Algorithm 9 is a 3-approximation, we think that they did not provide a complete proof for such claim. That is, they proved that Algorithm 9 has the following approximation ratio

$$\frac{c \cdot p(\pi)}{b(\pi)}, \text{ where } c = \frac{3 \lfloor \frac{b(\pi)}{3} \rfloor + b(\pi) \bmod 3}{\lceil \frac{b(\pi)}{3} \rceil},$$

and they also proved that  $c \leq 3$ , but it lacked the proof that  $p(\pi) \leq b(\pi)$ . Such proof is given by Lemma 14.

**Lemma 14.** *Given a permutation  $\pi \in S_n$ ,  $\pi \neq I_n$ , we have that  $p(\pi) < b(\pi)$ .*

*Proof.* Let  $\pi \in S_n$ ,  $\pi \neq I_n$ ,  $s_1$  be the first strip of  $\pi$ , and  $s_m$  be the last strip of  $\pi$ ,  $m \leq n$ . If  $\pi_1 = 1$ , then  $lc(\pi_i) = rc(\pi_i) = 0$  for any element  $\pi_i \in s_1$ . Thus, the elements belonging to  $s_1$  do not affect  $p(\pi)$ . The same can be observed for the elements belonging to  $s_m$  when  $\pi_n = n$ . It means that if we reduce  $\pi$  to  $\sigma$ , then  $p(\pi) = p(\sigma)$ . Since it is not hard to see that  $b(\pi) = b(\sigma)$ , we can restrict our analysis to irreducible permutations.

Let  $\gamma \in S_n^*$ , and let the series of consecutive elements  $\gamma_i \gamma_{i+1} \dots \gamma_j$  be a strip of  $\gamma$ . We have that  $lc(\gamma_{k+1}) = lc(\gamma_k)$  and  $rc(\gamma_{k+1}) = rc(\gamma_k)$ ,  $i \leq k < j$ . It means that, with respect to  $lc(\gamma)$  and  $rc(\gamma)$ , the elements belonging to a strip of  $\gamma$  either have zero value or are contained in the same plateau. Therefore,  $s(\gamma) \geq p(lc(\gamma))$  and  $s(\gamma) \geq p(rc(\gamma))$ , thus  $s(\gamma) \geq p(\gamma)$ . Since, by Lemma 13,  $b(\gamma) > s(\gamma)$ , the claim follows.  $\square$

Regarding the time complexity of Algorithm 9, Benoît-Gagné and Hamel [14] noted that  $lc(\pi)$  and  $rc(\pi)$  can be easily computed in  $O(n^2)$  time, while  $p(lc(\pi))$

and  $p(rc(\pi))$  can be easily computed in  $O(n)$  time. Therefore, they concluded that Algorithm 9 runs in  $O(n^2)$  time. In Section 4.4, we show how to compute  $lc(\pi)$  and  $rc(\pi)$  in  $O(n \log n)$  time.

### 4.3.3 Algorithm based on the longest increasing subsequence

Guyer, Heath, and Vergara [69] developed a greedy algorithm based on the longest increasing subsequence of a permutation  $\pi \in S_n$ . At each iteration, the algorithm selects, from the  $\binom{n+1}{3}$  possible transpositions, the transposition  $\rho(i, j, k)$  such that  $|\text{LIS}(\rho(i, j, k) \cdot \pi)|$  is maximum. We say that a transposition satisfying this greedy choice is a greedy transposition. Since there may exist more than one greedy transposition, the performance of this algorithm may vary depending on the rule used to choose among greedy transpositions. Guyer, Heath, and Vergara [69] neither pointed out any specific rule nor presented an approximation guarantee, therefore we decided to define a rule that could lead us to determine an approximation guarantee.

One might think of the rule that only greedy transpositions which remove breakpoints should be applied. Note that, using this rule, it would be trivial to prove an approximation bound of 3 due to Lemma 12. It is not true, however, that there always exists a greedy transposition satisfying this rule. For instance, among all the 56 permutations that can be obtained from permutation  $\pi = (7\ 5\ 6\ 4\ 2\ 3\ 1)$  by applying a transposition, there are 8 permutations yielded by greedy transpositions, but none of them has less breakpoints than  $\pi$ .

We say that a transposition  $\rho(i, j, k)$  does not cut a strip of a permutation  $\pi$  if the pairs of adjacent elements  $(\pi_{i-1}, \pi_i)$ ,  $(\pi_{j-1}, \pi_j)$  and  $(\pi_{k-1}, \pi_k)$  are breakpoints. The rule we considered is that only greedy transpositions which do not cut strips of permutations should be applied. Although it is possible that a greedy transposition cuts a strip of permutation (see Example 8), Lemma 17 shows that there is always a greedy transposition satisfying such a rule. Algorithm 10 is the resulting algorithm from this rule.

---

**Algorithm 10:** Constrained version of Guyer, Heath, and Vergara's heuristic [69].

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of transpositions applied for sorting  $\pi$ .

```

1  $d \leftarrow 0$ ;
2 while  $\pi \neq I_n$  do
3    $d \leftarrow d + 1$ ;
4    $\rho_d \leftarrow$  a transposition such that  $|\text{LIS}(\rho_d \cdot \pi)|$  is maximum and that does not
   cut a strip of  $\pi$ ;
5    $\pi \leftarrow \rho_d \cdot \pi$ ;
6 end
7 return  $d$ 

```

---

**Example 8.** Let  $\pi = (5\ 6\ 3\ 4\ 1\ 2)$ . We have that  $\rho(1, 3, 6)$  is a greedy transposition and it cuts the strip 1 2.

**Lemma 15.** Let  $\pi_1\ \pi_2\ \dots\ \pi_r$  be the first strip of a permutation  $\pi \in S_n$ . If  $\pi_1 = 1$ , then a transposition  $\rho(i, j, k)$  where  $i \leq r$  cannot be a greedy transposition.

*Proof.* Let  $\rho(i, j, k)$  be a transposition where  $i \leq r$  and let  $\pi'$  be the permutation  $\pi' = \rho(i, j, k) \cdot \pi$ . There are two cases to consider:

- a)  $j - 1 \leq r$ . In this case, it is not hard to see that  $|\text{LIS}(\pi')| \leq |\text{LIS}(\pi)|$  because the elements  $\pi_i, \pi_{i+1}, \dots, \pi_{j-1}$  are all smaller than the elements  $\pi_j, \pi_{j+1}, \dots, \pi_{k-1}$ , therefore  $\rho(i, j, k)$  could not be a greedy transposition.
- b)  $j - 1 > r$ . In this case, we argue that, if  $\rho(i, j, k)$  was a greedy transposition, then none of the elements  $\pi_i, \pi_{i+1}, \dots, \pi_r$  could belong to  $\text{LIS}(\pi')$ . For the sake of the contradiction, assume they could. Then none of the elements  $\pi_j, \pi_{j+1}, \dots, \pi_{k-1}$  could belong to  $\text{LIS}(\pi')$  because they are greater than the formers. This implies that the elements in  $\text{LIS}(\pi')$  would form an increasing subsequence in  $\pi$ , therefore  $|\text{LIS}(\pi')| \leq |\text{LIS}(\pi)|$  and  $\rho(i, j, k)$  could not be a greedy transposition. But if none of the elements  $\pi_i, \pi_{i+1}, \dots, \pi_r$  could belong to  $\text{LIS}(\pi')$ , then  $\rho(i, j, k)$  could not be a greedy transposition since  $|\text{LIS}(\rho(i+1, j, k) \cdot \pi)| > |\text{LIS}(\pi')|$ .

□

**Lemma 16.** Let  $\pi_s\ \pi_{s+1}\ \dots\ \pi_n$  be the last strip of a permutation  $\pi \in S_n$ . If  $\pi_n = n$ , then a transposition  $\rho(i, j, k)$  where  $k > s$  cannot be a greedy transposition.

*Proof.* Analogous to the proof of Lemma 15. □

**Lemma 17.** Given a permutation  $\pi$ , there exists a greedy transposition which does not cut any of its strips.

*Proof.* Let  $\rho(i, j, k)$  be a greedy transposition, and let  $\pi'$  be the permutation such that  $\pi' = \rho(i, j, k) \cdot \pi$ . If  $\rho(i, j, k)$  does not cut a strip of  $\pi$ , then we are done. Otherwise, we have to basically consider three possibilities:

- (a)  $(\pi_{i-1}, \pi_i)$  is not a breakpoint.

In this case let  $i'$  be the greatest integer such that  $i' < i$  and  $(\pi_{i'-1}, \pi_{i'})$  is a breakpoint (Lemma 15 guarantees that  $i' \geq r$  when  $\pi_1 = 1$ ), and let  $\pi''$  be the permutation such that  $\pi'' = \rho(i', j, k) \cdot \pi$ . Then, we have four subcases to analyze:

- (i)  $\pi_i \in \text{LIS}(\pi')$  and  $\{\pi_{i'}, \pi_{i'+1}, \dots, \pi_{i-1}\} \in \text{LIS}(\pi')$ . In this subcase we have that the elements in  $\text{LIS}(\pi')$  form an increasing subsequence in  $\pi''$ , therefore  $|\text{LIS}(\pi'')| \geq |\text{LIS}(\pi')|$ . Since  $\rho(i, j, k)$  is a greedy transposition by hypothesis, we have that  $|\text{LIS}(\pi'')| \leq |\text{LIS}(\pi')|$ . Therefore  $|\text{LIS}(\pi'')| = |\text{LIS}(\pi')|$  and  $\rho(i', j, k)$  is also a greedy transposition.

- (ii)  $\pi_i \in \text{LIS}(\pi')$  and  $\{\pi_{i'}, \pi_{i'+1}, \dots, \pi_{i-1}\} \notin \text{LIS}(\pi')$ . In this subcase we have that the elements in  $\{\pi_{i'}, \pi_{i'+1}, \dots, \pi_{i-1}\}$  along with the elements in  $\text{LIS}(\pi')$  form an increasing subsequence in  $\pi''$ , therefore  $|\text{LIS}(\pi'')| > |\text{LIS}(\pi')|$  and this contradicts our hypothesis that  $\rho(i, j, k)$  is a greedy transposition.
- (iii)  $\pi_i \notin \text{LIS}(\pi')$  and  $\{\pi_{i'}, \pi_{i'+1}, \dots, \pi_{i-1}\} \in \text{LIS}(\pi')$ . In this subcase we have that  $|\text{LIS}(\rho(i+1, j, k) \cdot \pi)| > |\text{LIS}(\pi')|$  and this contradicts our hypothesis that  $\rho(i, j, k)$  is a greedy transposition.
- (iv)  $\pi_i \notin \text{LIS}(\pi')$  and  $\{\pi_{i'}, \pi_{i'+1}, \dots, \pi_{i-1}\} \notin \text{LIS}(\pi')$ . The proof for this subcase is the same as that in subcase (a.i).

(b)  $(\pi_{j-1}, \pi_j)$  is not a breakpoint.

In this case let  $j'$  be the least integer such that  $j < j'$  and  $(\pi_{j'-1}, \pi_{j'})$  is a breakpoint, and let  $j''$  be the greatest integer such that  $j'' < j$  and  $(\pi_{j''-1}, \pi_{j''})$  is a breakpoint. It may be the case that either  $j' = k$  or  $j'' = i$ , but it is impossible that  $j' = k$  and  $j'' = i$ , otherwise  $\rho(i, j, k)$  would only move elements belonging to the same strip, therefore  $|\text{LIS}(\pi')| \leq |\text{LIS}(\pi)|$  and  $\rho(i, j, k)$  could not be a greedy transposition. Also note that a situation where  $\pi_{j-1} \in \text{LIS}(\pi')$  and  $\pi_j \in \text{LIS}(\pi')$  is not possible given the definition of an increasing subsequence. Then, if we assume that  $j' \neq k$  and let  $\pi''$  be the permutation such that  $\pi'' = \rho(i, j', k) \cdot \pi$ , we have three subcases to analyze:

- (i)  $\pi_{j-1} \in \text{LIS}(\pi')$  and  $\{\pi_j, \pi_{j+1}, \dots, \pi_{j'-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.ii).
- (ii)  $\pi_{j-1} \notin \text{LIS}(\pi')$  and  $\{\pi_j, \pi_{j+1}, \dots, \pi_{j'-1}\} \in \text{LIS}(\pi')$ . In this subcase we have that  $|\text{LIS}(\rho(i, j-1, k) \cdot \pi)| > |\text{LIS}(\pi')|$  and this contradicts our hypothesis that  $\rho(i, j, k)$  is a greedy transposition.
- (iii)  $\pi_{j-1} \notin \text{LIS}(\pi')$  and  $\{\pi_j, \pi_{j+1}, \dots, \pi_{j'-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.iv).

On the other hand, if we assume that  $j'' \neq i$  and let  $\pi''$  be the permutation such that  $\pi'' = \rho(i, j'', k) \cdot \pi$ , we also have three subcases to analyze:

- (i)  $\pi_j \in \text{LIS}(\pi')$  and  $\{\pi_{j''}, \pi_{j''+1}, \dots, \pi_{j-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.ii).
- (ii)  $\pi_j \notin \text{LIS}(\pi')$  and  $\{\pi_{j''}, \pi_{j''+1}, \dots, \pi_{j-1}\} \in \text{LIS}(\pi')$ . In this subcase we have that  $|\text{LIS}(\rho(i, j+1, k) \cdot \pi)| > |\text{LIS}(\pi')|$  and this contradicts our hypothesis that  $\rho(i, j, k)$  is a greedy transposition.
- (iii)  $\pi_j \notin \text{LIS}(\pi')$  and  $\{\pi_{j''}, \pi_{j''+1}, \dots, \pi_{j-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.iv).

(c)  $(\pi_{k-1}, \pi_k)$  is not a breakpoint.

In this case let  $k'$  be the least integer such that  $k < k'$  and  $(\pi_{k'-1}, \pi_{k'})$  is a breakpoint (Lemma 16 guarantees that  $k' \leq s$  when  $\pi_n = n$ ), and let  $\pi''$  be the permutation such that  $\pi'' = \rho(i, j, k') \cdot \pi$ . Then, we have four subcases to analyze:

- (i)  $\pi_{k-1} \in \text{LIS}(\pi')$  and  $\{\pi_k, \pi_{k+1}, \dots, \pi_{k'-1}\} \in \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.i).
- (ii)  $\pi_{k-1} \in \text{LIS}(\pi')$  and  $\{\pi_k, \pi_{k+1}, \dots, \pi_{k'-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.ii).
- (iii)  $\pi_{k-1} \notin \text{LIS}(\pi')$  and  $\{\pi_k, \pi_{k+1}, \dots, \pi_{k'-1}\} \in \text{LIS}(\pi')$ . In this subcase we have that  $|\text{LIS}(\rho(i, j, k-1) \cdot \pi)| > |\text{LIS}(\pi')|$  and this contradicts our hypothesis that  $\rho(i, j, k)$  is a greedy transposition.
- (iv)  $\pi_{k-1} \notin \text{LIS}(\pi')$  and  $\{\pi_k, \pi_{k+1}, \dots, \pi_{k'-1}\} \notin \text{LIS}(\pi')$ . This subcase is analogous to subcase (a.iv).

Although more than one possibility can occur at the same time, they are independent from each other, in such a way that in all possible cases, if transposition  $\rho(i, j, k)$  cuts a strip of  $\pi$ , then it is possible to derive a greedy transposition which does not, thus the claim follows.  $\square$

Based on the fact that Algorithm 10 does not apply greedy transpositions that cut strips, we are able to prove an upper bound on the number of transpositions it applies for sorting permutations (Lemma 18). Using this upper bound, it is possible to prove that Algorithm 10 is a 3-approximation (Theorem 4).

**Lemma 18.** *Let  $A_{10}(\pi)$  be the number of transpositions applied by Algorithm 10 for sorting a permutation  $\pi$ . Then, we have  $A_3(\pi) \leq s(\pi) - 1$ .*

*Proof.* For helping us to determine an upper bound to the number of transpositions applied by Algorithm 10, we define a simple procedure, called *StripSum*, which sums the sizes of the strips of a permutation. It receives as input a permutation  $\pi \in S_n$  and proceeds as follows. Firstly, it sorts all the strips of  $\pi$  with respect to their sizes, obtaining a list of strips  $s^0, s^1, \dots, s^{s(\pi)-1}$  such that  $|s^i| \geq |s^{i+1}|$  for all  $i, 0 \leq i < s(\pi) - 1$ . Secondly, it initializes a variable named SUM to  $|s^0|$ . Finally, starting from  $s^1$ , it iterates over the list of strips such that, at iteration  $i$ , the algorithm increases the value of SUM by  $|s^i|$ . Let  $\text{SUM}_i$  be the value of the variable SUM at iteration  $i$ , with  $\text{SUM}_0 = |s^0|$ . Clearly,  $\text{SUM}_i = \text{SUM}_{i-1} + |s^i|$  for all  $i, 1 \leq i \leq s(\pi) - 1$ . Besides, when the algorithm stops,  $\text{SUM} = \text{SUM}_{s(\pi)-1} = |s^0| + |s^1| + \dots + |s^{s(\pi)-1}| = n$ .

Now, assume that  $\pi$  was given as input to Algorithm 10 and let  $\pi^i$  be the permutation produced after  $i$  iterations, with  $\pi^0 = \pi$ . We can prove by induction that  $|\text{LIS}(\pi^i)| \geq \text{SUM}_i$ . For the base case, we have  $|\text{LIS}(\pi^0)| \geq \text{SUM}_0$  because, by definition,  $|\text{LIS}(\pi^0)|$  must be equal or greater than the size of any strip of  $\pi^0$ . For the

induction step, assume the claim holds for some  $0 < i < A_3(\pi)$ . Since Algorithm 10 never cuts a strip, all the strips of  $\pi^i$  are formed by strips of  $\pi^0$ . Let  $s'$  be the strip of greatest size among all strips of  $\pi^0$  whose elements do not belong to a given LIS( $\pi^i$ ). We have that  $|\text{LIS}(\pi^{i+1})| \geq |\text{LIS}(\pi^i)| + |s'|$  because it is possible to apply a transposition on  $\pi^i$  and obtain a new permutation containing an increasing subsequence formed by the elements of  $s'$  and LIS( $\pi^i$ ). If  $|s'| \geq |s^{i+1}|$ , then  $|\text{LIS}(\pi^{i+1})| \geq |\text{LIS}(\pi^i)| + |s'| \geq \text{SUM}_i + |s^{i+1}| = \text{SUM}_{i+1}$ . Otherwise, if  $|s'| < |s^{i+1}|$ , it means that the elements of all strips  $s^t$ ,  $0 \leq t \leq i + 1$ , belong to LIS( $\pi^i$ ), therefore  $|\text{LIS}(\pi^{i+1})| > |\text{LIS}(\pi^i)| \geq \text{SUM}_{i+1}$ .

The inequality  $|\text{LIS}(\pi^i)| \geq \text{SUM}_i$  implies that Algorithm 10 makes  $|\text{LIS}(\pi)|$  converge to  $n$  applying no more transpositions than the number of iterations that procedure *StripSum* performs. Since it performs  $s(\pi) - 1$  iterations, the lemma follows.  $\square$

**Theorem 4.** *Algorithm 3 is a 3-approximation.*

*Proof.* Lemma 15 guarantees that Algorithm 10 will never apply a transposition which moves the elements of the first strip of  $\pi$  when  $\pi_1 = 1$ . Similarly, Lemma 16 guarantees that Algorithm 10 will never apply a transposition which moves the elements of the last strip of  $\pi$  when  $\pi_n = n$ . For this reason, if we reduce permutation  $\pi$  to a permutation  $\sigma$ , it is not hard to see that  $A_{10}(\pi) = A_3(\sigma)$ . Thus, we can restrict our analysis to irreducible permutations.

Let  $\gamma \in S_n^*$ . By Lemma 18, we have that  $A_3(\gamma) \leq s(\gamma) - 1$ . Since, by Lemma 13,  $s(\gamma) = b(\gamma) - 1$ , we conclude that  $A_3(\gamma) \leq b(\gamma) - 2$ . It means that  $A_3(\gamma) \leq 3d(\gamma)$  once  $d(\gamma) \geq \frac{b(\gamma)}{3}$  (Lemma 12), and the theorem has been proved.  $\square$

Since there are  $O(n^3)$  possible transpositions to consider per iteration, it takes  $O(n \log n)$  time to determine a longest increasing subsequence of a permutation, it takes  $O(1)$  time to determine whether a transposition cuts a strip of a permutation, and the while loop executes  $O(n)$  times, we conclude that Algorithm 10 runs in  $O(n^5 \log n)$  time.

## 4.4 Computing Permutation Codes in $O(n \log n)$ Time

In this section, we describe how to compute the left and right codes of a permutation with  $n$  elements in  $O(n \log n)$  time, what allow us to implement Benoît-Gagné and Hamel's algorithm [14] in such a way that its running time becomes  $O(n \log n)$ . We note that permutation codes are closely related to the Lehmer code [95] (in fact, the Lehmer code is equivalent to the right code of a permutation) and there are known algorithms for computing the Lehmer code in  $O(n \log n)$  time (see [4, page 235]).

Given a permutation  $\pi \in S_n$ , the left code of the element  $\pi_k$  in the interval  $[i, j]$ , denoted by  $lc_{[i,j]}$ , is defined as

$$lc_{[i,j]}(\pi_k) = |\{\pi_l : \pi_l > \pi_k \text{ and } i \leq l \leq k-1\}|$$

for all  $i \leq k \leq j$ . Similarly, the right code of the element  $\pi_k$  in the interval  $[i, j]$ , denoted by  $rc_{[i,j]}$ , is defined as

$$rc_{[i,j]}(\pi_k) = |\{\pi_l : \pi_l < \pi_k \text{ and } k+1 \leq l \leq j\}|$$

for all  $i \leq k \leq j$ . If  $i = j$ , then we define  $lc_{[i,i]}(\pi_k) = rc_{[i,i]}(\pi_k) = 0$ .

It is not hard to realize that  $lc_{[i,j]}(\pi_k) = lc(\pi_k)$  and  $rc_{[i,j]}(\pi_k) = rc(\pi_k)$  when  $i = 1$  and  $j = n$ . By looking at the problem of computing the left/right code of the elements of a permutation  $\pi \in S_n$  as the problem of computing the left/right code of such elements in the interval  $[1, n]$ , it becomes clearer that we can use a divide-and-conquer approach to solve it. Firstly, we compute recursively the left/right code of the elements  $\pi_1 \pi_2 \dots \pi_m$  in the interval  $[1, m]$  and the left/right code of the elements  $\pi_{m+1} \pi_{m+2} \dots \pi_n$  in the interval  $[m+1, n]$  such that  $m = \lfloor \frac{n+1}{2} \rfloor$ . As for the base case, we have  $lc_{[i,i]}(\pi_i) = rc_{[i,i]}(\pi_i) = 0$  for all  $1 \leq i \leq n$ . Once the left/right codes of these elements have been computed in the referred intervals, we have that:

- $lc_{[1,n]}(\pi_k) = lc_{[1,m]}(\pi_k)$  for all  $1 \leq k \leq m$  and  $lc_{[1,n]}(\pi_k) = lc_{[m+1,n]}(\pi_k) + |\{\pi_l : \pi_l > \pi_k \text{ and } 1 \leq l \leq m\}|$  for all  $m+1 \leq k \leq n$ ;
- $rc_{[1,n]}(\pi_k) = rc_{[m+1,n]}(\pi_k)$  for all  $m+1 \leq k \leq n$  and  $rc_{[1,n]}(\pi_k) = rc_{[1,m]}(\pi_k) + |\{\pi_l : \pi_l < \pi_k \text{ and } m+1 \leq l \leq n\}|$  for all  $1 \leq k \leq m$ .

This means that, regarding the left code, the main question is how to efficiently compute  $|\{\pi_l : \pi_l > \pi_k \text{ and } 1 \leq l \leq m\}|$  for every element  $\pi_k$  such that  $m+1 \leq k \leq n$ ; regarding the right code, it is how to efficiently compute  $|\{\pi_l : \pi_l < \pi_k \text{ and } m+1 \leq l \leq n\}|$  for every element  $\pi_k$  such that  $1 \leq k \leq m$ .

The answer for the above questions relies on mergesort, that is, it is possible to adapt the merge step of mergesort so that we can efficiently establish the order relations between the elements and, consequently, we can efficiently compute those values. Firstly, we present an algorithm, called *MergeLeftCodes*, that shows how to accomplish it regarding the left code.

Algorithm *MergeLeftCodes* receives four parameters as input: a vector  $L$  containing the elements  $\pi_i, \pi_{i+1}, \dots, \pi_j$  ordered in ascending order; a vector  $R$  containing the elements  $\pi_{j+1}, \pi_{j+2}, \dots, \pi_k$  ordered in ascending order; a vector  $LC$  such that  $LC[e] = lc_{[i,j]}(\pi_e)$  for all  $i \leq e \leq j$  and  $LC[e] = lc_{[j+1,k]}(\pi_e)$  for all  $j+1 \leq e \leq k$ ; and the inverse permutation of  $\pi$ ,  $\pi^{-1}$ . As a result, it returns a vector  $M$  containing the elements of vectors  $L$  and  $R$  ordered in ascending order and updates vector  $LC$  in such a way that  $LC[e] = lc_{[i,k]}(\pi_e)$  for all  $i \leq e \leq k$ .

We will prove the correctness of Algorithm *MergeLeftCodes* by proving that the following loop invariants hold for the while loop of lines 7-18:

- vector  $M$  contains the  $m-1$  smallest elements of  $L$  and  $R$  ordered in ascending order;



- $LC[e] = lc_{[i,k]}(\pi_e)$ , where  $e = \pi_{M[t]}^{-1}$ , for all  $1 \leq t \leq m - 1$ ;
- $L[l]$  and  $R[r]$  are the smallest elements of vectors  $L$  and  $R$  that have not been copied to  $M$ .

---

**Algorithm 11:** MergeLeftCodes.
 

---

**Data:** Three vectors  $L$ ,  $R$  and  $LC$ , and permutation  $\pi^{-1} \in S_n$ .

**Result:** Returns a vector containing the elements of  $L$  and  $R$  ordered in ascending order and updates vector  $LC$  in such a way that  $LC[e] = lc_{[i,k]}(\pi_e)$  for all  $i \leq e \leq k$ .

```

1  $l \leftarrow 1$ ;
2  $r \leftarrow 1$ ;
3  $m \leftarrow 1$ ;
4 Let  $M$  be a vector of size  $|L| + |R|$ ;
5  $R[|R| + 1] \leftarrow n + 1$ ;
6  $L[|L| + 1] \leftarrow n + 1$ ;
7 while  $m \leq |M|$  do
8   if  $R[r] < L[l]$  then
9      $M[m] \leftarrow R[r]$ ;
10     $e \leftarrow \pi_{M[m]}^{-1}$ ;
11     $LC[e] \leftarrow LC[e] + |L| - l$ ;
12     $r \leftarrow r + 1$ ;
13  else
14     $M[m] \leftarrow L[l]$ ;
15     $l \leftarrow l + 1$ ;
16  end
17   $m \leftarrow m + 1$ ;
18 end
19 return  $M$ ;

```

---

It not hard to see that these loop invariants hold before the first iteration of the while loop of lines 7-18. At each iteration, we have to consider two possibilities: either  $R[r] < L[l]$  or  $R[r] > L[l]$ .

If  $R[r] < L[l]$ , then  $R[r]$  is the smallest element that has not been copied to  $M$ , therefore it is copied to  $M$  (line 9). Since the elements of  $L$  are to the left of the elements of  $R$  in the permutation  $\pi$ , it means that there exists  $|L| - l$  elements greater than  $M[m]$  and to its left in  $\pi$  considering just the elements of  $L$  (note that the element  $n + 1$  must not be considered). Given that  $LC[e]$ ,  $e = \pi_{M[m]}^{-1}$ , equals the number of elements greater than  $M[m]$  and to its left in  $\pi$  considering just the elements of  $R$ , we conclude that  $LC[e] + |L| - l$  equals the number of elements greater than  $M[m]$  and to its left in  $\pi$  considering both the elements of  $L$  and  $R$ . In other words, we have  $LC[e] = lc_{[i,k]}(\pi_e)$  after line 11. Finally, the variables  $r$  and  $m$  are incremented (lines 12 and 17), thus the loop invariants still hold.

If  $R[r] > L[l]$ , then  $L[l]$  is the smallest element that has not been copied to  $M$ , therefore it is copied to  $M$  (line 14). Since the elements of  $L$  are to the left of the elements of  $R$  in the permutation  $\pi$ , we have  $LC[e] = lc_{[i,k]}(\pi_e)$ ,  $e = \pi_{M[m]}^{-1}$ , therefore it is not necessary to update vector  $LC$ . Finally, the variables  $l$  and  $m$  are incremented (lines 15 and 17), thus the loop invariants still hold.

After the while loop of lines 7-18 terminates, it is clear that vector  $M$  will contain the elements of vectors  $L$  and  $R$  ordered in ascending order. Besides, vector  $LC$  will have been updated in such a way that  $LC[e] = lc_{[i,k]}(\pi_e)$  for all  $i \leq e \leq k$ .

As for the time complexity of Algorithm *MergeLeftCodes*, we have that each of the lines 1, 2, 3, 5, and 6 runs in time  $O(1)$ , line 4 runs in time  $O(n)$ , and the while loop executes  $O(n)$  times. Therefore, Algorithm *MergeLeftCodes* runs in  $O(n)$  time.

With Algorithm *MergeLeftCodes* in hand, the recursive algorithm that computes the left code of the elements  $\pi_i \pi_{i+1} \dots \pi_j$  in the interval  $[i, j]$  can be trivially derived from the discussion held at the beginning of this section. This algorithm is called *RecursiveLeftCode* and it is presented below. In order to obtain the left code of a permutation  $\pi$ , we simply execute Algorithm *RecursiveLeftCode* as described by Algorithm 13.

---

**Algorithm 12:** RecursiveLeftCode.

---

**Data:** A permutation  $\pi \in S_n$  and its inverse,  $\pi^{-1}$ , vector  $LC$ , and indexes  $i$  and  $j$ .

**Result:** Returns a vector containing the elements  $\pi_i \pi_{i+1} \dots \pi_j$  ordered in ascending order and updates vector  $LC$  in such a way that  $LC[e] = lc_{[i,j]}(\pi_e)$  for all  $i \leq e \leq j$ .

```

1  $m \leftarrow \lfloor \frac{i+j}{2} \rfloor$ ;
2 if  $i < j$  then
3    $L \leftarrow \text{RecursiveLeftCode}(\pi, \pi^{-1}, LC, i, m)$ ;
4    $R \leftarrow \text{RecursiveLeftCode}(\pi, \pi^{-1}, LC, m+1, j)$ ;
5    $M \leftarrow \text{MergeLeftCodes}(L, R, LC, \pi^{-1})$ ;
6 else
7   Let  $M$  be a vector of size 1;
8    $M[1] \leftarrow \pi_i$ ;
9    $LC[i] \leftarrow 0$ ;
10 end
11 return  $M$ ;

```

---

The time complexity of Algorithm *RecursiveLeftCode* equals the time complexity of mergesort, which is  $O(n \log n)$ . Computing  $\pi^{-1}$  as well as creating a vector of size  $n$  takes  $O(n)$  time, therefore Algorithm 13 runs in  $O(n \log n)$  time.

In the case of the right code, the adaptation to the merge step of mergesort is very similar to the one made in the case of the left code, except that the elements are ordered in descending order rather than in ascending order. Moreover, it is not hard to see that the correctness and complexity analyses of the algorithms for computing

---

**Algorithm 13:** Computing the left code of a permutation.

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Returns a vector containing the left codes of the elements of  $\pi$ .

- 1 Compute  $\pi^{-1}$ ;
  - 2 Let  $LC$  be a vector of size  $n$ ;
  - 3 `RecursiveLeftCode`( $\pi, \pi^{-1}, LC, 1, n$ );
  - 4 **return**  $LC$ ;
- 

the right code of a permutation are analogous to the ones performed for algorithms *MergeLeftCodes* and *RecursiveLeftCode*, therefore we omit them.

## 4.5 Experimental Results and Discussion

The following sections describe the experiments we have performed and discuss the results we have obtained. The algorithms described in this paper were implemented in Java, while the algorithms based on the cycle graph were implemented in Python (we used Dias and Dias [36, 37] implementations). The experiments were performed on an Intel<sup>®</sup> Core<sup>™</sup> i7-2600K CPU at 3.40GHz with 16GB of RAM running Ubuntu 12.04.2 LTS operating system.

### 4.5.1 Experiments on small permutations

The approximation algorithms presented in Section 4.3 were implemented and tested by their authors for verifying their performance in practice. One kind of test was to compare the distance computed by the algorithm with  $d(\pi)$  for every  $\pi \in S_n$  in order to obtain the real approximation ratio of the respective approximation algorithm for small permutations. More specifically, Walter, Dias, and Meidanis [124] ran this test for  $1 \leq n \leq 11$ , Benoît-Gagné and Hamel [14] ran it for  $1 \leq n \leq 9$ , and Guyer, Heath, and Vergara [69] ran it just for  $n = 6$ .

We ran this kind of test for  $1 \leq n \leq 13$  for all algorithms using GRAAu [58], and the results are presented in tables 4.1, 4.2, and 4.3, where  $n$  is the size of the permutations, *Max. Dist.* is the greatest distance outputted by the algorithm, *Avg. Dist.* is the average of the distances outputted by the algorithm, *Avg. Ratio* is the average of the ratios between the distance outputted by the algorithm and the transposition distance, *Max. Ratio* is the greatest ratio among all the ratios between the distance outputted by the algorithm and the transposition distance, and *Equals* is the percentage of distances outputted by the algorithm that is equal to the transposition distance.

Firstly, we note that the results obtained by Walter, Dias, and Meidanis for their 2.25-approximation algorithm are incorrect. For instance, the maximum value of  $\frac{A_8(\pi)}{d(\pi)}$  they observed for  $n = 11$  was  $\frac{10}{5}$ . But this result cannot be right because  $A_8(\pi) \leq 9$  for every  $\pi \in S_{11}$ . Given a permutation  $\pi \in S_n$ , it is easy to see that  $0 \leq b(\pi) \leq$

Table 4.1: Results obtained from the audit of the implementation of Walter, Dias, and Meidanis' algorithm.

<b>n</b>	<b>Max. Dist.</b>	<b>Avg. Dist.</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Equals</b>
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	2	1.00	1.00	1.00	100.00%
4	3	1.54	1.00	1.00	100.00%
5	3	2.08	1.00	1.00	100.00%
6	4	2.61	1.00	1.33	99.17%
7	5	3.14	1.00	1.33	98.57%
8	6	3.66	1.01	1.50	97.12%
9	6	4.19	1.01	1.50	96.06%
10	7	4.70	1.01	1.50	94.15%
11	8	5.22	1.01	1.60	92.84%
12	9	5.73	1.02	1.60	90.68%
13	9	6.24	1.02	1.60	89.30%

Table 4.2: Results obtained from the audit of the implementation of Benoît-Gagné and Hamel's algorithm.

<b>n</b>	<b>Max. Dist.</b>	<b>Avg. Dist.</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Equals</b>
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	2	1.00	1.00	1.00	100.00%
4	3	1.54	1.00	1.00	100.00%
5	4	2.13	1.02	1.50	95.00%
6	5	2.75	1.06	1.67	85.00%
7	6	3.42	1.10	2.00	71.77%
8	7	4.13	1.14	2.00	56.41%
9	8	4.87	1.18	2.00	41.62%
10	9	5.63	1.22	2.25	28.80%
11	10	6.42	1.25	2.25	18.74%
12	11	7.22	1.29	2.25	11.57%
13	12	8.05	1.32	2.40	6.77%

Table 4.3: Results obtained from the audit of the implementation of Algorithm 3, which is a constrained version of Guyer, Heath, and Vergara’s heuristic.

n	Max. Dist.	Avg. Dist.	Avg. Ratio	Max. Ratio	Equals
1	0	0.00	1.00	1.00	100.00%
2	1	0.50	1.00	1.00	100.00%
3	2	1.00	1.00	1.00	100.00%
4	3	1.54	1.00	1.00	100.00%
5	4	2.10	1.01	1.50	97.50%
6	5	2.67	1.03	1.50	92.78%
7	6	3.26	1.05	1.67	86.45%
8	7	3.86	1.06	1.67	77.93%
9	8	4.48	1.08	2.00	69.06%
10	9	5.10	1.10	2.00	58.94%
11	10	5.73	1.12	2.00	49.61%
12	11	6.38	1.14	2.00	40.23%
13	12	7.03	1.15	2.25	32.18%

Table 4.4: Permutations  $\pi^m$  of size  $3m + 1$ ,  $m \in \{5, 6, 7\}$ , for which  $\frac{p(\pi^m)}{d(\pi^m)} = \frac{3m}{m+1}$ . Note that  $d(\pi^m) \geq \frac{b(\pi^m)}{3} \geq m + 1$ .

Permutation	Transposition Sorting Sequence
$\pi^5 = (16\ 9\ 4\ 11\ 6\ 15\ 8\ 2\ 12\ 7\ 5\ 3\ 14\ 13\ 10\ 1)$	$\rho(5, 9, 12), \rho(1, 7, 10), \rho(3, 9, 14), \rho(5, 10, 17), \rho(6, 10, 14), \rho(1, 7, 11)$
$\pi^6 = (19\ 11\ 4\ 18\ 6\ 14\ 8\ 13\ 10\ 2\ 15\ 5\ 9\ 7\ 3\ 17\ 16\ 12\ 1)$	$\rho(4, 12, 17), \rho(7, 12, 16), \rho(6, 9, 15), \rho(1, 5, 11), \rho(3, 8, 20), \rho(4, 13, 17), \rho(1, 5, 13)$
$\pi^7 = (22\ 13\ 4\ 21\ 6\ 17\ 8\ 16\ 10\ 15\ 12\ 2\ 18\ 5\ 11\ 9\ 7\ 3\ 20\ 19\ 14\ 1)$	$\rho(4, 14, 20), \rho(8, 13, 19), \rho(7, 10, 18), \rho(6, 9, 17), \rho(1, 5, 11), \rho(3, 8, 23), \rho(4, 16, 20), \rho(1, 5, 15)$

$n + 1$ . As discussed in Section 4.3.1,  $A_8(\pi) \leq \frac{3}{4}b(\pi)$ , therefore  $A_8(\pi) \leq \frac{3n+3}{4}$ , and the claim follows.

The real approximation ratios observed for the 2.25-approximation algorithm seem to increase in a progression that converges to 2, that is,  $\frac{2}{2}, \frac{4}{3}, \frac{6}{4}, \frac{8}{5}, \dots, \frac{2k}{k+1}$ . This may indicate that a deeper analysis of this algorithm could lead one to prove that it is in fact a 2-approximation.

Regarding the 3-approximation algorithm developed by Benoît-Gagné and Hamel, if we just consider the real approximation ratios obtained for  $n \in \{7, 10, 13\}$ , we can observe that they seem to follow the progression  $\frac{6}{3}, \frac{9}{4}, \frac{12}{5}, \dots, \frac{3k}{k+1}$ . We ran further experiments to verify the strength of this assumption, and we found permutations  $\pi^m$  of size  $3m + 1$ ,  $m \in \{5, 6, 7\}$ , for which  $\frac{p(\pi^m)}{d(\pi^m)} = \frac{3m}{m+1}$  (these permutations are presented

in Table 4.4). Note that, for  $m = 7$ , the real approximation ratio of Benoît-Gagné and Hamel’s algorithm equals  $\frac{21}{8} = 2.625$ . This is an indication that the approximation ratio of this algorithm may not be lowered, contradicting the hypothesis raised by Benoît-Gagné and Hamel that its approximation ratio “tends to a number significantly smaller than 3”.

Figure 4.1 illustrates that, of the three algorithms, Walter, Dias, and Meidanis’ algorithm has the best practical performance.

### 4.5.2 Experiments on large permutations

In order to investigate what happens in practice for large permutations and to compare the algorithms regarded in this paper against the best known algorithms based on the cycle graph (namely Bafna and Pevzner’s 1.5-approximation algorithm [10], Elias and Hartman’s 1.375-approximation algorithm [48], Dias and Dias’ [36] extension of Bafna and Pevzner’s algorithm [10], and Dias and Dias’ [37] extension of Elias and Hartman’s algorithm [48]) we tested all these algorithms on the same set of arbitrarily

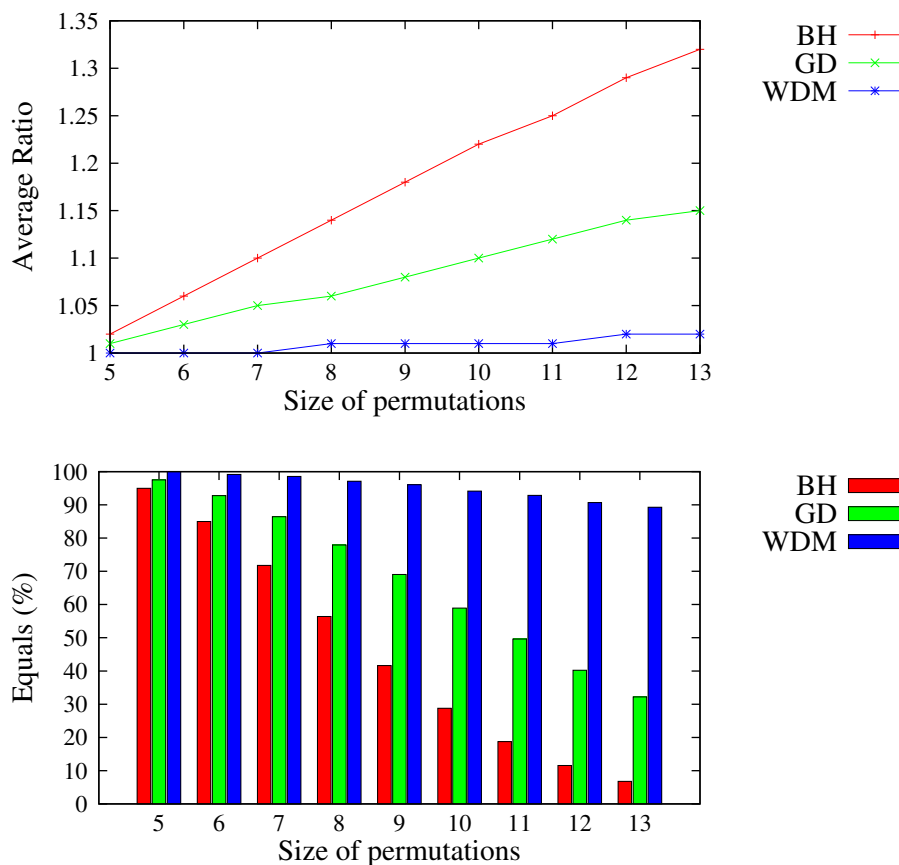


Figure 4.1: Comparison of Walter, Dias, and Meidanis’ algorithm (WDM), Benoît-Gagné and Hamel’s algorithm (BH), and the constrained version of Guyer, Heath, and Vergara’s heuristic (GD) based on the results provided by GRAAu.

large permutations. This set consisted of 59,000 random permutations of sizes varying between 10 and 300 in intervals of 5, with 1,000 permutations of each size.

Figure 4.2 shows the average distance computed for all algorithms. As can be seen, these data corroborate with the data obtained for small permutations, that is, of the three algorithms studied in this paper, Walter, Dias, and Meidanis' algorithm has the best practical performance. Figure 4.2 also shows that Walter, Dias, and Meidanis' algorithm provided results comparable to those provided by the algorithms based on the cycle graph. For the purpose of further verifying how good the algorithms studied in this paper performed in comparison to the algorithms based on the cycle graph, we computed how often each algorithm provided the best distance. The results are presented in figures 4.3 and 4.4.

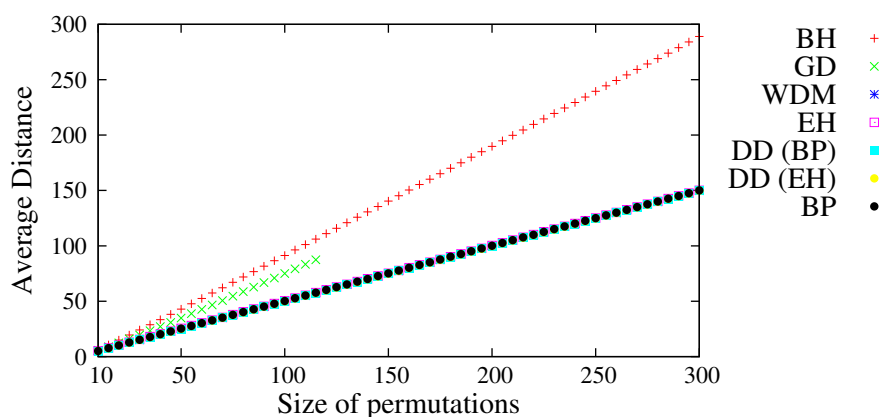


Figure 4.2: Comparison of Walter, Dias, and Meidanis' algorithm (WDM), Benoît-Gagné and Hamel's algorithm (BH), the constrained version of Guyer, Heath, and Vergara's heuristic (GD), Bafna and Pevzner's algorithm (BP), Elias and Hartman's algorithm (EH), and Dias and Dias' algorithms (DD (BP) and DD (EH)) based on the average distance. Due to time constraints, we could not compute the average distance of the constrained version of Guyer, Heath, and Vergara's heuristic for permutations with more than 115 elements (note that this algorithm runs in  $O(n^5 \log n)$  time). The average distances computed for algorithms WDM, EH, DD (BP), DD (EH) and BP were about equal, therefore they are overlapping in the graph.

We can notice that the results were consistently the same regardless of the size of the permutations. Benoît-Gagné and Hamel's algorithm and the constrained version of Guyer, Heath, and Vergara's heuristic provided the best distance less times than the other algorithms (for permutations with more than 20 elements, they did not provide the best distance even once). Walter, Dias, and Meidanis' algorithm provided the best distance more times than Bafna and Pevzner's algorithm and Elias and Hartman's algorithm, but less times than Dias and Dias' algorithms. Although Walter, Dias, and Meidanis' algorithm did not outperform Dias and Dias' algorithms, which are the best known algorithms for sorting by transpositions, it is remarkable that it outperformed two approximation algorithms with much better approximation ratios.

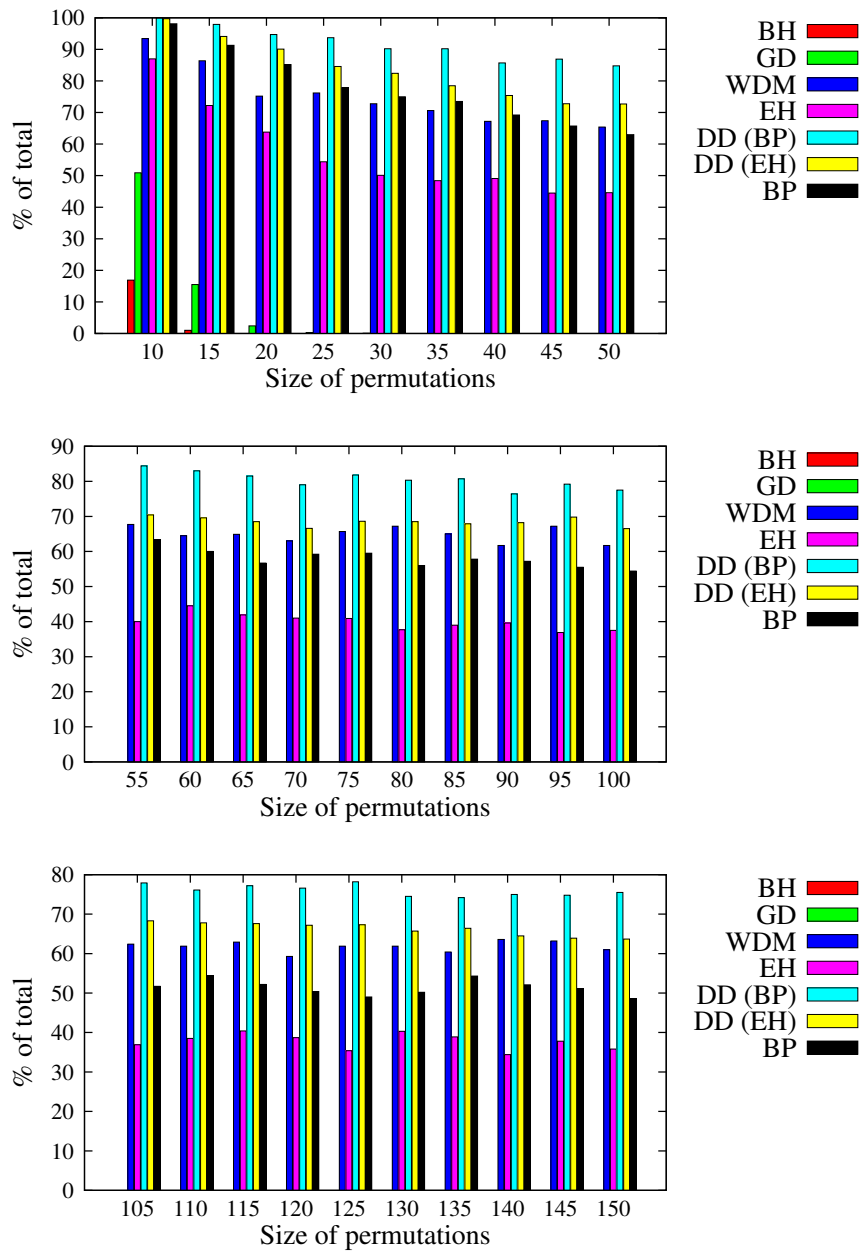


Figure 4.3: Relative number of times each algorithm provided the best distance. Note that more than one algorithm can have provided the best distance.



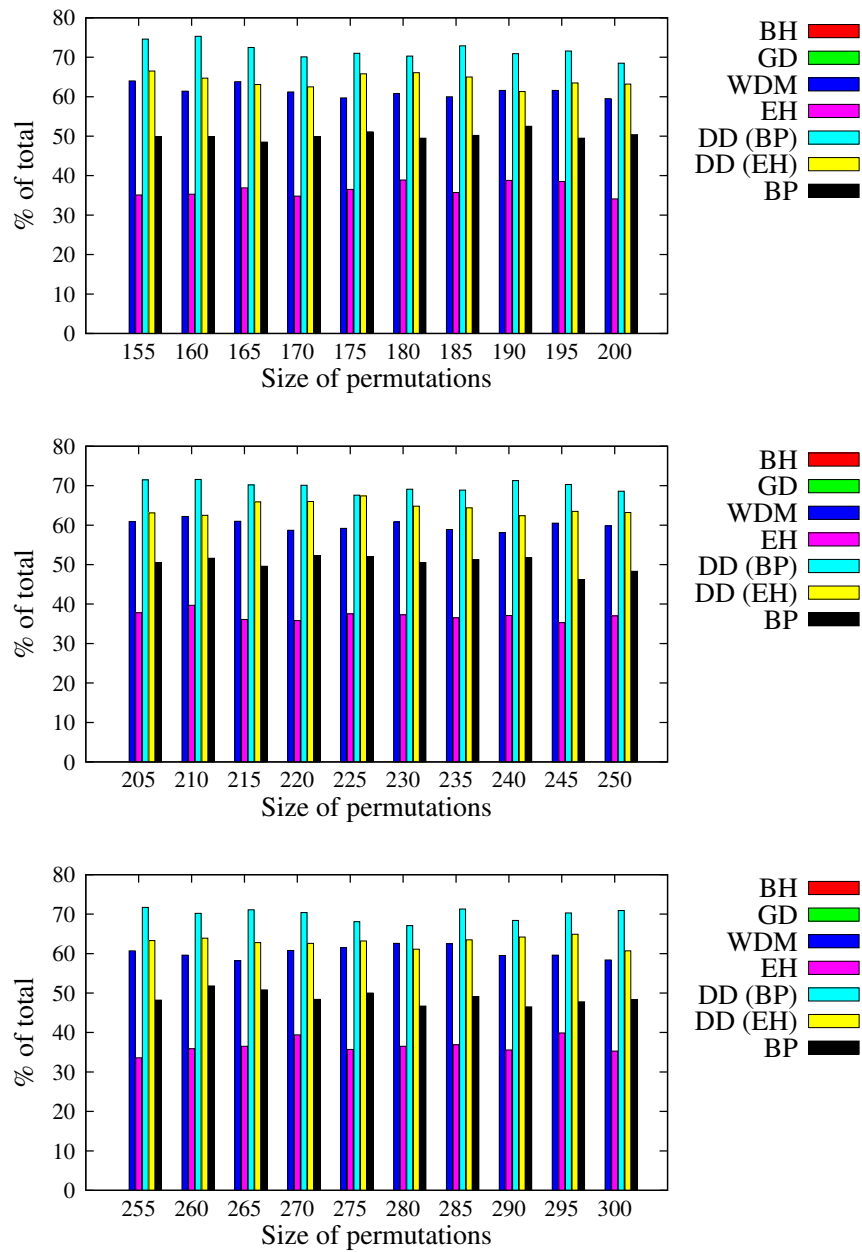


Figure 4.4: Relative number of times each algorithm provided the best distance. Note that more than one algorithm can have provided the best distance.

## 4.6 Conclusions

In this paper, we revisited three algorithms for the problem of sorting by transpositions: Walter, Dias, and Meidanis' 2.25-approximation algorithm [124], Benoît-Gagné and Hamel's 3-approximation algorithm [14], and Guyer, Heath, and Vergara's heuristic [69]. These algorithms are based on alternative approaches to the cycle graph, which is the standard tool for tackling permutation sorting problems.

Regarding theoretical aspects, we closed a missing gap on the proof of the approximation ratio of Benoît-Gagné and Hamel's algorithm [14] and we demonstrated a way to run their algorithm in  $O(n \log n)$  time. This latter reinforces Benoît-Gagné and Hamel's argument that, although there does exist better algorithms with respect to approximation ratio, their algorithm is fast. We proposed a minor adaptation to Guyer, Heath, and Vergara's heuristic [69] that allowed us to prove an approximation bound of 3. Finally, with respect to Walter, Dias, and Meidanis' algorithm [124], we did not present any theoretical improvement, but we demonstrated that previous experimental data on its approximation ratio are incorrect.

Regarding practical aspects, we performed an experimental investigation of these three algorithms for small and large permutations. For the experiments on small permutations, we considered all permutations with up to 13 elements. To the best of our knowledge, this was the first time these algorithms were tested for all permutations with more than 11 elements. For the experiments on large permutations, we also taken into account approximation algorithms based on the cycle graph, namely Bafna and Pevzner's 1.5-approximation algorithm [10], Elias and Hartman's 1.375-approximation algorithm [48], Dias and Dias' [36] extension of Bafna and Pevzner's algorithm [10], and Dias and Dias' [37] extension of Elias and Hartman's algorithm [48]. The latter two are the best known algorithms for the problem of sorting by transpositions.

The experimental data yielded by the experiments on small permutations gave some insights on the approximation ratio of the algorithms under study. It indicated that the approximation ratio of Benoît-Gagné and Hamel's algorithm [14] may not be lowered, contradicting a first hypothesis [14] that it could be, and that the approximation ratio of Walter, Dias, and Meidanis' algorithm [124] may be lowered to 2. Unfortunately, we could not obtain any proof regarding the tightness of the approximation ratio of the studied algorithms.

Both the experiments on small and large permutations pointed out Walter, Dias, and Meidanis' algorithm [124] as the best algorithm out of the three algorithms based on alternative approaches. Moreover, the experiments on large permutations showed that Walter, Dias, and Meidanis' algorithm [124] provided results comparable to the ones provided by the algorithms based on the cycle graph. In fact, Walter, Dias, and Meidanis' algorithm [124] outperformed on average both Bafna and Pevzner's algorithm [10] and Elias and Hartman's algorithm [48], what is remarkable since these algorithms have much better approximation ratios.

We conclude that, although the algorithms based on alternative approaches have

worse approximation ratios, Benoît-Gagné and Hamel's algorithm [14] is a good alternative due to its simplicity and its practical and asymptotic speed, while Walter, Dias, and Meidanis' algorithm [124] is a good alternative in terms of practical results. The constrained version of Guyer, Heath, and Vergara's heuristic [69] proposed by us does not figure as a good alternative because it did not present good practical results and it has a prohibitive time complexity, just as the original heuristic.

Although the experimental data on small permutations suggested that none of the studied algorithms are promising alternatives in terms of approximation ratios, it is still not clear whether the approaches they rely on can or cannot yield algorithms with low approximation ratios. Therefore, searching for results that could help make progress on this question either way is an interesting direction to follow for future work.



## Chapter 5

# Sorting Signed Permutations by Short Operations \*

**Abstract:** During evolution, global mutations may alter the order and the orientation of the genes in a genome. Such mutations are referred to as rearrangement events, or simply operations. In unichromosomal genomes, the most common operations are reversals, which are responsible for reversing the order and orientation of a sequence of genes, and transpositions, which are responsible for switching the location of two contiguous portions of a genome. The problem of computing the minimum sequence of operations that transforms one genome into another – which is equivalent to the problem of sorting a permutation into the identity permutation – is a well-studied problem that finds application in comparative genomics. There are a number of works concerning this problem in the literature, but they generally do not take into account the length of the operations (*i.e.* the number of genes affected by the operations). Since it has been observed that short operations are prevalent in the evolution of some species, algorithms that efficiently solve this problem in the special case of short operations are of interest. In this paper, we investigate the problem of sorting a signed permutation by short operations. More precisely, we study four flavors of this problem: (i) the problem of sorting a signed permutation by reversals of length at most 2; (ii) the problem of sorting a signed permutation by reversals of length at most 3; (iii) the problem of sorting a signed permutation by reversals and transpositions of length at most 2; and (iv) the problem of sorting a signed permutation by reversals and transpositions of length at most 3. We present polynomial-time solutions for problems (i) and (iii), a 5-approximation for problem (ii), and a 3-approximation for problem (iv). Moreover, we show that the expected approximation ratio of the 5-approximation algorithm is not greater than 3 for random signed permutations with more than 12 elements. Finally, we present experimental results that show that the approximation ratios of the approximation algorithms cannot be smaller than 3. In particular, this means that the approximation ratio of the 3-approximation algorithm is tight.

---

\* *Gustavo Rodrigues Galvão, Orlando Lee, and Zanoni Dias. Sorting signed permutations by short operations. Algorithms for Molecular Biology, Volume 10, Article 12, 2015. Copyright 2015 Rodrigues Galvão et al.; licensee BioMed Central. DOI: <http://dx.doi.org/10.1186/s13015-015-0040-x>*

## 5.1 Background

One of the challenges of modern science is to understand how species evolve. As evolution can be viewed as a branching process, whereby new species arise from changes occurring in living organisms, the study of the evolutionary history of a group of species is commonly made by analyzing trees whose nodes represent species and edges represent evolutionary relationships. Since these relationships are referred to as phylogeny, such trees are called phylogenetic trees.

Phylogenies can be inferred from different kinds of data, from geographic and ecological, through behavioral, morphological, and metabolic, to molecular data, such as DNA. Molecular data have the advantage of being exact and reproducible, at least within experimental error, not to mention fairly easy to obtain [63, Chapter 12]. Among the existing methods for phylogenetic reconstruction from molecular data, we focus on those referred to as distance-based methods. These methods build the phylogenetic tree corresponding to a group of species as follows. First, the evolutionary distance between each pair of species is estimated in order to generate a distance matrix  $M$  such that each entry  $M_{i,j}$  contains the evolutionary distance between species  $i$  and  $j$ . Then, the phylogenetic tree is constructed from this matrix using a specific algorithm, such as *Neighbor-Joining* [108]. Therefore, a key point of distance-based methods is how to estimate the evolutionary distance between two species.

A well-accepted approach for estimating the evolutionary distance is the genome rearrangement approach [54]. It proposes to estimate the evolutionary distance between two species using the rearrangement distance between their genomes, which is the length of the shortest sequence of genome-wide mutations, called rearrangement events, that transforms one genome into the other. Assuming genomes consist of a single linear chromosome, share the same set of genes, and contain no duplicated genes, we can represent them as permutations of integers where each integer corresponds to a gene. Besides, each integer may have a sign,  $+$  or  $-$ , indicating the gene orientation. Permutations whose elements have signs are called signed permutations and permutations whose elements do not have signs are called unsigned permutations.

By representing genomes as permutations, the problem of finding the shortest sequence of rearrangement events that transforms one genome into another can be reduced to the combinatorial problem of calculating the minimum number of operations necessary to transform one permutation into another. By algebraic properties of permutations, this problem can be equivalently stated as the problem of calculating the minimum number of operations necessary to transform one permutation into the identity permutation  $(+1 +2 \dots +n)$ . This problem is commonly referred to as the permutation sorting problem.

Depending on the operations allowed to sort a permutation, we have a different variant of the permutation sorting problem. Reversals and transpositions are the most often considered operations for phylogenetic reconstruction. A reversal is responsible for reversing the order and flipping the signs of a sequence of elements within

a permutation, while a transposition is responsible for switching the location of two contiguous portions of a permutation. The problem of sorting an unsigned permutation by reversals is an NP-hard problem [23]. It was introduced by Watterson *et al.* [126] and the best known result is due to Berman, Hannenhalli and Karpinski [15], who presented a 1.375-approximation algorithm. The problem of sorting a signed permutation by reversals was introduced by Bafna and Pevzner [9], who presented a 1.5-approximation algorithm. Hannenhalli and Pevzner [72] presented the first polynomial algorithm for this problem, which was further improved by Tannier, Bergeron and Sagot [119] to run in subquadratic time. Barder, Moret and Yan [8] showed how to determine the minimum number of reversals that sorts a signed permutation (without actually sorting) in linear time. The problem of sorting an unsigned permutation by transpositions is an NP-hard problem [22]. It was introduced by Bafna and Pevzner [10], who presented a 1.5-approximation algorithm. Later, Elias and Hartman [48] improved the approximation bound to 1.375. Variants of the permutation sorting problem which allow both reversals and transpositions are also regarded in the literature [68, 107, 123].

Simultaneously with the study of the aforementioned variants of the permutation sorting problem, some researchers have investigated variants in which bounds are imposed on the lengths of the operations. Jerrum [80] proved that the problem of sorting an unsigned permutation by reversals (or transpositions) of length 2 is solvable in polynomial time. Later, Heath and Vergara [78] considered the problem of sorting an unsigned permutation by reversals of length at most 3 and presented the best known result for it, a 2-approximation algorithm. Heath and Vergara [76, 77] also considered the problem of sorting an unsigned permutation by transpositions of length at most 3 and presented a  $\frac{4}{3}$ -approximation algorithm. Jiang *et al.* [82] presented a  $(1+\epsilon)$ -approximation for unsigned permutations with many inversions and, more recently, Jiang *et al.* [81] also devised an  $\frac{5}{4}$ -approximation algorithm for sorting general unsigned permutations by transpositions of length at most 3. Finally, Vergara [121] showed that the  $\frac{4}{3}$ -approximation algorithm for the problem of sorting by transpositions of length at most 3 is a 2-approximation algorithm for the problem of sorting by reversals and transpositions of length at most 3.

The biological relevance of these bounded variants is grounded on the assumption that rearrangement events affecting large portions of a genome are less likely to occur. In the past, corroborating evidence has emerged, that is, separate sets of observations have shown the prevalence and significance of short reversals (*i.e.* reversals involving one or a few genes) in the evolution of bacterial genomes [32, 94] and lower eukaryotes genomes [100, 110]. This fact, together with the realization that signed permutations constitute a more biologically relevant model for genomes, motivated us to investigate the problem of sorting a signed permutation by short operations.

In preliminary work, Galvão and Dias [59] investigated the problem of sorting a signed permutation by reversals of length at most 3 and presented three approxima-

tion algorithms, the best one having an approximation factor of 9. In this paper, we not only present an approximation algorithm with a better approximation factor, but also consider other bounded variants. More precisely, we study four variants of the permutation sorting problem: (i) the problem of sorting a signed permutation by reversals of length at most 2, (ii) the problem of sorting a signed permutation by reversals of length at most 3, (iii) the problem of sorting a signed permutation by reversals and transpositions of length at most 2, and (iv) the problem of sorting a signed permutation by reversals and transpositions of length at most 3. We present polynomial-time solutions for problems (i) and (iii), a 5-approximation for problem (ii), and a 3-approximation for problem (iv). Moreover, we show that the expected approximation factor of the 5-approximation algorithm is not greater than 3 for random signed permutations with more than 12 elements. Finally, we present experimental results that show that the approximation factors of the approximation algorithms cannot be smaller than 3. In particular, this means that the approximation factor of the 3-approximation algorithm is tight.

## 5.2 Preliminaries

In this section, we present basic definitions that are used throughout this paper, generally following [59]. Let  $n$  be a positive integer.

A *signed permutation*  $\pi$  is a bijection of  $\{-n, \dots, -2, -1, 1, 2, \dots, n\}$  onto itself that satisfies  $\pi(-i) = -\pi(i)$  for all  $i \in \{1, 2, \dots, n\}$ . The two-row notation for a signed permutation is

$$\pi = \begin{pmatrix} -n & \dots & -2 & -1 & 1 & 2 & \dots & n \\ -\pi_n & \dots & -\pi_2 & -\pi_1 & \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \dots, n\}$  for  $1 \leq i \leq n$ . The notation used in genome rearrangement literature, which is the one we will adopt, is the one-row notation  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . Note that we drop the mapping of the negative elements since  $\pi(-i) = -\pi(i)$  for all  $i \in \{1, 2, \dots, n\}$ . By abuse of notation, we say that  $\pi$  has size  $n$ . The set of all signed permutations of size  $n$  is  $S_n^\pm$ .

A signed reversal  $\rho(i, j)$ ,  $1 \leq i \leq j \leq n$ , is an operation that transforms a signed permutation  $\pi = (\pi_1 \pi_2 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n)$  into the signed permutation  $\pi \cdot \rho(i, j) = (\pi_1 \pi_2 \dots \pi_{i-1} \underline{-\pi_j \ -\pi_{j-1} \dots \ -\pi_{i+1} \ -\pi_i} \pi_{j+1} \dots \pi_n)$ . A signed reversal  $\rho(i, j)$  is called a *signed  $k$ -reversal* if  $k = j - i + 1$ . A signed  $k$ -reversal is called *short* if  $k \leq 3$ . It is called *super short* if  $k \leq 2$ .

The problem of sorting by signed short reversals consists in finding the minimum number of signed short reversals that transform a permutation  $\pi \in S_n^\pm$  into the *identity permutation*  $\iota_n = (+1 \ +2 \ \dots \ +n)$ . This number is referred to as the *signed short reversal distance* of permutation  $\pi$  and it is denoted by  $d_{ssr}(\pi)$ . Similarly, the problem of sorting by signed super short reversals consists in finding the minimum number of



signed super short reversals that transform a permutation  $\pi \in S_n^\pm$  into  $\iota_n$ . This number is referred to as the *signed super short reversal distance* of permutation  $\pi$  and it is denoted by  $d_{ssr}(\pi)$ .

A *transposition*  $\rho(i, j, k)$ ,  $1 \leq i < j < k \leq n + 1$ , is an operation that transforms a signed permutation  $\pi = (\pi_1 \dots \pi_{i-1} \overline{\pi_i \dots \pi_{j-1}} \overline{\pi_j \dots \pi_{k-1}} \pi_k \dots \pi_n)$  into the signed permutation  $\pi \cdot \rho(i, j, k) = (\pi_1 \dots \pi_{i-1} \overline{\pi_j \dots \pi_{k-1}} \overline{\pi_i \dots \pi_{j-1}} \pi_k \dots \pi_n)$ . A transposition  $\rho(i, j, k)$  is called an  $(x, y)$ -transposition, where  $x = j - i$  and  $y = k - j$ . An  $(x, y)$ -transposition is called *short* if  $x + y \leq 3$ . It is called *super short* if  $x + y = 2$ .

The problem of sorting by signed short operations consists in finding the minimum number of signed short reversals and short transpositions that transform a permutation  $\pi \in S_n^\pm$  into  $\iota_n$ . This number is referred to as the *signed short operation distance* of permutation  $\pi$  and it is denoted by  $d_{sso}(\pi)$ . Similarly, the problem of sorting by signed super short operations consists in finding the minimum number of signed super short reversals and super short transpositions that transform a permutation  $\pi \in S_n^\pm$  into  $\iota_n$ . This number is referred to as the *signed super short operation distance* of a permutation  $\pi$  and it is denoted by  $d_{ssso}(\pi)$ .

We say that a pair of elements  $(\pi_i, \pi_j)$  of a signed permutation  $\pi$  is an *inversion* if  $i < j$  and  $|\pi_i| > |\pi_j|$ . The number of inversions in a signed permutation  $\pi$  is denoted by  $\text{Inv}(\pi)$ .

**Lemma 19.** *Let  $\pi$  be a signed permutation. If  $\text{Inv}(\pi) > 0$ , then there exists an inversion  $(\pi_i, \pi_j)$  such that  $j = i + 1$ .*

*Proof.* Let  $\pi_1, \pi_2, \dots, \pi_i$  be a maximal subsequence such that  $|\pi_1| < |\pi_2| < \dots < |\pi_i|$ . Since  $\text{Inv}(\pi) > 0$ , we have that  $i < n$ . So  $|\pi_{i+1}| < |\pi_i|$  and the result follows.  $\square$

Let  $\Delta\text{Inv}(\pi, \rho)$  denote the change in the number of inversions in a signed permutation  $\pi$  due to the application of an operation  $\rho$ , that is,  $\Delta\text{Inv}(\pi, \rho) = \text{Inv}(\pi) - \text{Inv}(\pi \cdot \rho)$ . The following lemma provides bounds on the value of  $\Delta\text{Inv}(\pi, \rho)$  considering that  $\rho$  is a short operation.

**Lemma 20.** *Let  $\pi$  be a signed permutation. Then, we have that*

- i)  $-1 \leq \Delta\text{Inv}(\pi, \rho) \leq 1$  if  $\rho$  is a super short operation,
- ii)  $-2 \leq \Delta\text{Inv}(\pi, \rho) \leq 2$  if  $\rho$  is a short transposition, and
- iii)  $-3 \leq \Delta\text{Inv}(\pi, \rho) \leq 3$  if  $\rho$  is a signed short reversal.

*Proof.* Suppose first that  $\rho$  is a super short operation. If  $\rho$  is a 1-reversal, then  $\Delta\text{Inv}(\pi, \rho) = 0$ . Moreover, if  $\rho$  is a signed 2-reversal  $\rho(i, i + 1)$  or a  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$ , then  $\Delta\text{Inv}(\pi, \rho) = 1$  if  $(\pi_i, \pi_{i+1})$  is an inversion and  $\Delta\text{Inv}(\pi, \rho) = -1$  otherwise.

Now, suppose that  $\rho$  is a  $(1, 2)$ -transposition  $\rho(i, i + 1, i + 2)$ . We have that if  $(\pi_i, \pi_{i+1})$  and  $(\pi_i, \pi_{i+2})$  are inversions, then  $\Delta\text{Inv}(\pi, \rho) = 2$ . On the other hand, if  $(\pi_i,$

$\pi_{i+1}$ ) and  $(\pi_i, \pi_{i+2})$  are not inversions, then  $\Delta\text{Inv}(\pi, \rho) = -2$ . Finally, if either  $(\pi_i, \pi_{i+1})$  or  $(\pi_i, \pi_{i+2})$  is an inversion, then  $\Delta\text{Inv}(\pi, \rho) = 0$ . Note that a similar argument holds if  $\rho$  is a  $(2, 1)$ -transposition.

Finally, suppose that  $\rho$  is a signed 3-reversal  $\rho(i, i + 2)$ . We have that if  $|\pi_i| > |\pi_{i+1}| > |\pi_{i+2}|$ , then  $\Delta\text{Inv}(\pi, \rho) = 3$ . On the other hand, if  $|\pi_i| < |\pi_{i+1}| < |\pi_{i+2}|$ , then  $\Delta\text{Inv}(\pi, \rho) = -3$ . Since in the other subcases we have that  $-1 \leq \Delta\text{Inv}(\pi, \rho) \leq 1$ , the lemma follows.  $\square$

### 5.3 Sorting by Bounded Signed Reversals

In this section, we present a polynomial-time solution for the problem of sorting by super short signed reversals and a 5-approximation algorithm for the problem of sorting by signed short reversals. Before we present the main results, we first introduce a useful tool for tackling these problems, the *vector diagram*. This tool was also used by Heath and Vergara [78,121] for the problem of sorting by (unsigned) short reversals.

#### 5.3.1 The Vector Diagram

For each element  $\pi_i$  of a signed permutation  $\pi$ , we define a *vector*  $v(\pi_i)$  whose length is given by  $|v(\pi_i)| = ||\pi_i| - i|$ . If  $|v(\pi_i)| > 0$ , the vector  $v(\pi_i)$  has a direction indicated by the sign of  $|\pi_i| - i$ . The vector  $v(\pi_i)$  is a *right vector* if  $|\pi_i| - i > 0$  while it is a *left vector* if  $|\pi_i| - i < 0$ . If the length of  $v(\pi_i)$  is zero, then  $v(\pi_i)$  is said to be a *positive zero vector* if  $\pi_i = i$  and a *negative zero vector* if  $\pi_i = -i$ . A vector diagram  $V_\pi$  of  $\pi$  is the set of vectors of the elements of  $\pi$ . The sum of the lengths of all the vectors in  $V_\pi$  is denoted by  $\text{Vec}(\pi)$ . See Figure 5.1 for an example.

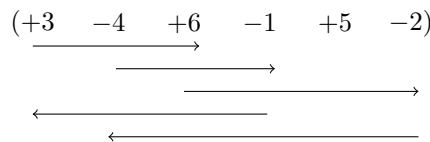


Figure 5.1: Vector diagram of the signed permutation  $\pi = (+3 -4 +6 -1 +5 -2)$ . Note that  $\text{Vec}(\pi) = 14$ .

Two elements  $\pi_i$  and  $\pi_j$ ,  $i < j$ , of a signed permutation  $\pi$  are said to be *vector-opposite* if the vectors  $v(\pi_i)$  and  $v(\pi_j)$  differ in direction,  $|v(\pi_i)| \geq j - i$ , and  $|v(\pi_j)| \geq j - i$ . Besides, they are said to be *m-vector-opposite* if  $j - i = m$ . Note that  $m$  specifies the distance between vector-opposite elements. For instance, in Figure 5.1 the elements  $\pi_2 = -4$  and  $\pi_4 = -1$  are 2-vector-opposite elements.

**Lemma 21.** *Let  $\pi$  be a signed permutation. If  $\text{Inv}(\pi) > 0$ , then  $\pi$  contains at least a pair of vector-opposite elements.*

*Proof.* We say that an element  $\pi_e$  in  $\pi$  is out-of-place if  $|\pi_e| \neq e$ . Note that there must exist out-of-place elements in  $\pi$  if  $\text{Inv}(\pi) > 0$ . Among all out-of-place elements in  $\pi$ , let  $\pi_i$  be the one with the greatest absolute value. We first show by contradiction that  $v(\pi_i)$  is a right vector. Suppose  $v(\pi_i)$  is a left vector, that is,  $|\pi_i| - i < 0$ . Then the element  $\pi_k$  such that  $|\pi_k| = i$  is an out-of-place element with absolute value greater than  $|\pi_i|$ , a contradiction.

Now since there is at least one right vector in  $V_\pi$ , there exists a rightmost right vector in  $V_\pi$ , that is, a right vector  $v(\pi_i)$  such that  $i$  is as large as possible. The element  $\pi_k$  such that  $k = |\pi_i|$  is out-of-place since  $|\pi_k| \neq k$ . The vector  $v(\pi_k)$  is therefore a left vector as it occurs to the right of  $v(\pi_i)$ , the rightmost right vector. Consider the elements  $\pi_{i+1}, \pi_{i+2}, \dots, \pi_k$ . At least one of these elements corresponds to a left vector. Select the leftmost left vector from these elements, that is, select the vector  $v(\pi_j)$  such that  $i + 1 \leq j \leq k$  and  $j$  is as small as possible.

We claim that  $\pi_i$  and  $\pi_j$  are vector-opposite elements. Since  $|v(\pi_i)| = k \geq j$ , all that remains to be shown is that  $|v(\pi_j)| \leq i$ . In other words, we need to show that the correct position of element  $\pi_j$  does not occur to the right of position  $i$ . For a contradiction, suppose this is the case. Then the element  $\pi_t$  such that  $t = |\pi_j|$  is out-of-place and therefore  $v(\pi_t)$  is either a right or left vector. It is not a right vector since it occurs on the right of  $v(\pi_i)$ , the rightmost right vector. It is not a left vector since it occurs on the left of  $v(\pi_j)$ , the leftmost left vector from a set that includes  $v(\pi_t)$ . Then we have a contradiction since we have found an out-of-place element that corresponds to a zero vector. The lemma follows.  $\square$

**Lemma 22.** *Let  $\pi \in S_n^\pm$  be a signed permutation such that  $\text{Inv}(\pi) > 0$  and let  $\pi_i$  and  $\pi_j$  be  $m$ -vector-opposite elements. Moreover, let  $\pi' \in S_n^\pm$  be a signed permutation such that  $|\pi'_i| = |\pi_j|$ ,  $|\pi'_j| = |\pi_i|$ , and  $|\pi'_k| = |\pi_k|$  for all  $k \notin \{i, j\}$ . Then  $\text{Vec}(\pi) - \text{Vec}(\pi') = 2m$ .*

*Proof.* We have that

$$\begin{aligned} \text{Vec}(\pi) - \text{Vec}(\pi') &= \sum_{k=1}^n (|v(\pi_k)| - |v(\pi'_k)|) \\ &= |v(\pi_i)| - |v(\pi'_i)| + |v(\pi_j)| - |v(\pi'_j)| \\ &= m + m \\ &= 2m, \end{aligned}$$

and therefore the lemma follows.  $\square$

Let  $\Delta\text{Vec}(\pi, \rho)$  denote the change in the sum of the lengths of all the vectors in  $V_\pi$  due to the application of a signed reversal  $\rho$ , that is,  $\Delta\text{Vec}(\pi, \rho) = \text{Vec}(\pi) - \text{Vec}(\pi \cdot \rho)$ . The following lemma provides bounds on the value of  $\Delta\text{Vec}(\pi, \rho)$  considering that  $\rho$  is a signed short reversal.

**Lemma 23.** *Let  $\pi$  be a signed permutation. Then, we have that*

$$i) \Delta\text{Vec}(\pi, \rho) = 0 \text{ if } \rho \text{ is a signed 1-reversal,}$$

ii)  $-2 \leq \Delta \text{Vec}(\pi, \rho) \leq 2$  if  $\rho$  is a signed 2-reversal, and

iii)  $-4 \leq \Delta \text{Vec}(\pi, \rho) \leq 4$  if  $\rho$  is a signed 3-reversal.

*Proof.* Suppose first that  $\rho$  is a signed 1-reversal  $\rho(i, i)$ . In this case,  $\rho$  does not affect the length of the vector  $v(\pi_i)$ , therefore  $\Delta \text{Vec}(\pi, \rho) = 0$ .

Now, suppose that  $\rho$  is a signed 2-reversal  $\rho(i, i + 1)$ . If the elements  $\pi_i$  and  $\pi_{i+1}$  are 1-vector-opposite, then  $\Delta \text{Vec}(\pi, \rho) = 2$ . On the other hand, if  $v(\pi_i)$  is a zero or a left vector and  $v(\pi_{i+1})$  is a zero or a right vector, then  $\Delta \text{Vec}(\pi, \rho) = -2$ . Note that  $\Delta \text{Vec}(\pi, \rho)$  cannot be greater than 2 and cannot be less than  $-2$  because  $\rho(i, i + 1)$  can increase or decrease the length of  $v(\pi_i)$  and  $v(\pi_{i+1})$  by just one unit.

Finally, suppose that  $\rho$  is a signed 3-reversal  $\rho(i, i + 2)$ . Note that  $\rho$  does not affect the length of the vector  $v(\pi_{i+1})$ . Now, if the elements  $\pi_i$  and  $\pi_{i+2}$  are 2-vector-opposite, then  $\Delta \text{Vec}(\pi, \rho) = 4$ . On the other hand, if  $v(\pi_i)$  is a zero or a left vector and  $v(\pi_{i+2})$  is a zero or a right vector, then  $\Delta \text{Vec}(\pi, \rho) = -4$ . Note that  $\Delta \text{Vec}(\pi, \rho)$  cannot be greater than 4 and cannot be less than  $-4$  because  $\rho(i, i + 2)$  can increase or decrease the length of  $v(\pi_i)$  and  $v(\pi_{i+2})$  by just two units.  $\square$

### 5.3.2 Sorting by Signed Super Short Reversals

From the proof of Lemma 20, we have that a signed 1-reversal does not change the number of inversions in a signed permutation and a signed 2-reversal can eliminate at most one inversion. This means that, for sorting a signed permutation  $\pi$ , we have to apply  $\text{Inv}(\pi)$  signed 2-reversals plus a given number of signed 1-reversals in order to flip the signs of the remaining negative elements. The question is: how many signed 1-reversals do we have to apply?

Intuitively, if an element  $\pi_i$  is in  $t$  distinct pairs of inversions in a signed permutation  $\pi$ , then its sign will be flipped  $t$  times, one time per signed 2-reversal applied. Therefore, if  $\pi_i$  is negative and  $t$  is even, then  $\pi_i$  will remain negative after we apply the  $t$  signed 2-reversals. The same is true when  $\pi_i$  is positive and  $t$  is odd. We can make use of the vector diagram in order to capture this intuition formally.

Let  $V_\pi^{\text{even}^-}$  be a subset of  $V_\pi$  such that  $V_\pi^{\text{even}^-} = \{v(\pi_i) : \pi_i < 0 \text{ and } |v(\pi_i)| \text{ is even}\}$  and let  $V_\pi^{\text{odd}^+}$  be a subset of  $V_\pi$  such that  $V_\pi^{\text{odd}^+} = \{v(\pi_i) : \pi_i > 0 \text{ and } |v(\pi_i)| \text{ is odd}\}$ . The elements of a signed permutation  $\pi$  whose vectors belong to either  $V_\pi^{\text{even}^-}$  or  $V_\pi^{\text{odd}^+}$  are precisely the elements which will be negative after we apply the  $\text{Inv}(\pi)$  signed 2-reversals (Lemma 24). Using this fact, we can obtain an exact formula for the signed super short reversal distance of a signed permutation  $\pi$  (Theorem 5).

**Lemma 24.** *Let  $\pi$  be a signed permutation and let  $\pi' = \pi \cdot \rho(i, i + 1)$ . Then, we have that  $|V_{\pi'}^{\text{even}^-}| + |V_{\pi'}^{\text{odd}^+}| = |V_\pi^{\text{even}^-}| + |V_\pi^{\text{odd}^+}|$ .*

*Proof.* The signed 2-reversal  $\rho(i, i + 1)$  changes the signs of  $\pi_i$  and  $\pi_{i+1}$  along with the parities of  $|v(\pi_i)|$  and  $|v(\pi_{i+1})|$ . For this reason, if  $\pi_i$  (or  $\pi_{i+1}$ ) belongs to either  $V_\pi^{\text{even}^-}$  or  $V_\pi^{\text{odd}^+}$ , then  $\pi'_{i+1} = -\pi_i$  (or  $\pi'_i = -\pi_{i+1}$ ) belongs to either  $V_{\pi'}^{\text{even}^-}$  or  $V_{\pi'}^{\text{odd}^+}$ .

On the other hand, if  $\pi_i$  (or  $\pi_{i+1}$ ) does not belong to neither  $V_\pi^{even^-}$  nor  $V_\pi^{odd^+}$ , then  $\pi'_{i+1} = -\pi_i$  (or  $\pi'_i = -\pi_{i+1}$ ) does not belong to either  $V_{\pi'}^{even^-}$  or  $V_{\pi'}^{odd^+}$ . Therefore the lemma follows.  $\square$

**Lemma 25.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{sssr}(\pi) \leq \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}|$ .*

*Proof.* It suffices to prove that it is always possible to apply signed super short reversals on  $\pi \neq \iota_n$  in such a way that the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + |V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| \leq \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}| - 1. \quad (5.1)$$

If  $\text{Inv}(\pi) = 0$ , then  $|v(\pi_i)| = 0$  for every  $\pi_i$  of  $\pi$ . This means that  $|V_\pi^{odd^+}| = 0$ , and therefore we can sort  $\pi$  with  $|V_\pi^{even^-}|$  signed 1-reversals and (5.1) holds.

If  $\text{Inv}(\pi) > 0$ , then there exists a signed 2-reversal  $\rho(i, i + 1)$  that removes an inversion in  $\pi$  (Lemma 19). So, apply such signed 2-reversal on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ . Moreover, we have that  $|V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| = |V_\pi^{even^-}| + |V_\pi^{odd^+}|$  (Lemma 24). Summing both equalities we obtain (5.1), therefore the lemma follows.  $\square$

**Lemma 26.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{sssr}(\pi) \geq \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}|$ .*

*Proof.* It suffices to prove that if we apply an arbitrary signed super short reversal on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + |V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| \geq \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}| - 1. \quad (5.2)$$

Suppose first that we apply a signed 1-reversal  $\rho(i, i)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi)$ . Moreover, since the sign of  $\pi_i$  is flipped without changing the parity of  $|v(\pi_i)|$ , we have that  $|V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| \geq |V_\pi^{even^-}| + |V_\pi^{odd^+}| - 1$ . Summing the previous equality with this inequality we obtain (5.2).

Now, suppose that we apply a signed 2-reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $|V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| = |V_\pi^{even^-}| + |V_\pi^{odd^+}|$  (Lemma 24). Moreover, since a signed 2-reversal can remove at most one inversion, we have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 1$ . Summing the previous equality with this inequality we obtain (5.2). Therefore the lemma follows.  $\square$

**Theorem 5.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{sssr}(\pi) = \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}|$ .*

*Proof.* Immediate from Lemmas 25 and 26.  $\square$

From the proof of Lemma 25, we can derive the following optimal algorithm for sorting a signed permutation by signed super short reversals. First, perform signed

2-reversals on the inversions until the permutation has no inversions. Then, perform signed 1-reversals on the negative elements until the permutation has no negative elements. Since a signed permutation  $\pi \in S_n^\pm$  can have at most  $\binom{n}{2}$  inversions and at most  $n$  negative elements, we have that this algorithm runs in  $O(n^2)$  time. We remark that the value of  $d_{ssr}(\pi)$  can be computed in  $O(n\sqrt{\log n})$  time because computing  $|V_\pi^{even^-}| + |V_\pi^{odd^+}|$  takes  $O(n)$  time and computing  $\text{Inv}(\pi)$  takes  $O(n\sqrt{\log n})$  time [26].

### 5.3.3 Sorting by Signed Short Reversals

A trivial algorithm for the problem of sorting by signed short reversals is the optimal algorithm for the problem of sorting by signed super short reversals. From the lower bound of Lemma 27, it follows that this trivial algorithm is a 6-approximation algorithm. Moreover, we have that this approximation bound is tight. For instance, we need 6 signed super short reversals for sorting the signed permutation  $(-3 \ -2 \ -1)$ , but one signed 3-reversal is sufficient for sorting it.

**Lemma 27.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{ssr}(\pi) \geq \frac{\text{Inv}(\pi) + |V_\pi^-| + |V_\pi^+|}{6}$ .*

*Proof.* It suffices to prove that if we apply an arbitrary signed short reversal on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + |V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| \geq \text{Inv}(\pi) + |V_\pi^{even^-}| + |V_\pi^{odd^+}| - 6. \quad (5.3)$$

From the proof of Lemma 26, we have that (5.3) holds when we apply a signed super short reversal on  $\pi$ . So, suppose that we apply the signed 3-reversal  $\rho(i, i+2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 3$ . Moreover, we have that  $|V_{\pi'}^{even^-}| + |V_{\pi'}^{odd^+}| \geq |V_\pi^{even^-}| + |V_\pi^{odd^+}| - 3$ . Summing both inequalities we obtain (5.3), and the lemma follows.  $\square$

Let  $V_\pi^{odd}$  be a subset of  $V_\pi$  such that  $V_\pi^{odd} = \{v(\pi_i) : |v(\pi_i)| \text{ is odd}\}$  and let  $V_\pi^{0-}$  be a subset of  $V_\pi$  such that  $V_\pi^{0-} = \{v(\pi_i) : v(\pi_i) \text{ is a negative zero vector}\}$ . By using these two subsets of  $V_\pi$ , we can obtain better bounds on the signed short reversal distance of a signed permutation  $\pi$  (Lemmas 29 and 30). These bounds lead to a 5-approximation for the problem of sorting by signed short reversals (Theorem 6). We note that the upper bound given in Lemma 29 relies on the fact that it is always possible to switch the positions of a pair of  $m$ -vector-opposite elements (without affecting the elements between them) applying  $m$  signed short reversals (Lemma 28).

**Lemma 28.** *Let  $\pi \in S_n^\pm$  be a signed permutation such that  $\text{Inv}(\pi) > 0$  and let  $\pi_i$  and  $\pi_j$  be  $m$ -vector-opposite elements. It is possible to transform  $\pi$  into  $\pi' \in S_n^\pm$  such that  $|\pi'_i| = |\pi_j|$ ,  $|\pi'_j| = |\pi_i|$ , and  $|\pi'_k| = |\pi_k|$  for all  $k \notin \{i, j\}$  applying  $d$  signed short reversals, where*

$$d = \begin{cases} m - 1 & \text{if } m \text{ is even,} \\ m & \text{if } m \text{ is odd.} \end{cases}$$

*Proof.* We have two cases to consider:

- a)  $m$  is even. In this case, we can transform  $\pi$  into a signed permutation  $\pi' \in S_n^\pm$  such that  $|\pi'_i| = |\pi_j|$ ,  $|\pi'_j| = |\pi_i|$ ,  $\pi'_{j-1} = -\pi_{j-1}$ , and  $\pi'_k = \pi_k$  for all  $k \notin \{i, j-1, j\}$  applying the sequence of signed short reversals  $\rho(i, i+2), \rho(i+2, i+4), \dots, \rho(j-4, j-2), \rho(j-2, j), \rho(j-4, j-2), \dots, \rho(i, i+2)$ . Therefore, to transform  $\pi$  into  $\pi'$ , we can apply  $m-1$  signed 3-reversals.
- b)  $m$  is odd. In this case, we can transform  $\pi$  into a signed permutation  $\pi' \in S_n^\pm$  such that  $|\pi'_i| = |\pi_j|$ ,  $|\pi'_j| = |\pi_i|$ , and  $\pi'_k = \pi_k$  for all  $k \notin \{i, j\}$  applying the sequence of signed short reversals  $\rho(i, i+2), \rho(i+2, i+4), \dots, \rho(j-3, j-1), \rho(j-1, j), \rho(j-3, j-1), \dots, \rho(i, i+2)$ . Therefore, to transform  $\pi$  into  $\pi'$ , we can apply  $m-1$  signed 3-reversals and one signed 2-reversal, totalizing  $m$  signed short reversals.

Since in both cases we can transform  $\pi$  into  $\pi'$  applying  $2\lceil \frac{m}{2} \rceil - 1$ , the lemma follows.  $\square$

**Lemma 29.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{ssr}(\pi) \leq \text{Vec}(\pi) + |V_\pi^{odd}| + |V_\pi^{0-}|$ .*

*Proof.* It suffices to prove that it is always possible to apply a sequence of  $t > 0$  signed short reversals on  $\pi \neq \iota_n$  in such a way that the resulting permutation  $\pi'$  satisfies

$$\text{Vec}(\pi') + |V_{\pi'}^{odd}| + |V_{\pi'}^{0-}| \leq \text{Vec}(\pi) + |V_\pi^{odd}| + |V_\pi^{0-}| - t. \quad (5.4)$$

If  $\text{Vec}(\pi) = 0$ , then  $|v(\pi_i)| = 0$  for every  $\pi_i$  in  $\pi$ . This means that  $|V_\pi^{odd}| = 0$ . Therefore we can sort  $\pi$  with  $|V_\pi^{0-}|$  signed 1-reversals and (5.4) holds.

If  $\text{Vec}(\pi) > 0$ , then  $\pi$  contains at least one pair of vector-opposite elements (Lemma 21). Let  $\pi_i$  and  $\pi_j$ ,  $i < j$ , be  $m$ -vector-opposite elements. Now, suppose that we apply the  $d$  signed reversals described in Lemma 28 on  $\pi$  and let  $\pi'$  denote the resulting permutation. We will show that the application of this sequence of signed short reversals results in an average decrease in

$$\begin{aligned} \Delta(\pi, \pi') &= \text{Vec}(\pi) + |V_\pi^{odd}| + |V_\pi^{0-}| - (\text{Vec}(\pi') + |V_{\pi'}^{odd}| + |V_{\pi'}^{0-}|) \\ &= 2m + (|V_\pi^{odd}| - |V_{\pi'}^{odd}|) + (|V_\pi^{0-}| - |V_{\pi'}^{0-}|) \end{aligned}$$

of at least 1 unit per signed short reversal. In other words, we need to show that  $\frac{\Delta(\pi, \pi')}{d} \geq 1$ .

In order to evaluate the value of  $\Delta(\pi, \pi')$ , we divide our analysis in two cases:

- a)  $m$  is even. In this case, we have that the parities of the lengths of the vectors do not change, therefore  $|V_\pi^{odd}| - |V_{\pi'}^{odd}| = 0$ . In order to evaluate the value of  $|V_\pi^{0-}| - |V_{\pi'}^{0-}|$ , we further divide our analysis into three subcases:
- i)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  are even. In this subcase, we have that the vectors  $v(\pi_i)$ ,  $v(\pi_{j-1})$ , and  $v(\pi_j)$  may become negative zero vectors, therefore  $|V_\pi^{0-}| - |V_{\pi'}^{0-}| \geq -3$ . This means that  $\Delta(\pi, \pi') \geq 2m - 3$ .

- ii)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  have distinct parities. In this subcase, we have that the vector  $v(\pi_{j-1})$  and one of the vectors  $v(\pi_i)$  and  $v(\pi_j)$  (precisely the one whose length is even) may become negative zero vectors, therefore  $|V_\pi^{0^-}| - |V_{\pi'}^{0^-}| \geq -2$ . This means that  $\Delta(\pi, \pi') \geq 2m - 2$ .
- iii)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  are odd. In this subcase, we have that none of the vectors  $v(\pi_i)$  and  $v(\pi_j)$  can become a negative zero vector, but the vector  $v(\pi_{j-1})$  can. Therefore  $|V_\pi^{0^-}| - |V_{\pi'}^{0^-}| \geq -1$ . This means that  $\Delta(\pi, \pi') \geq 2m - 1$ .

b)  $m$  is odd. In this case, we further divide our analysis into three subcases:

- i)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  are even. In this subcase, we have that none of the vectors  $v(\pi_i)$  and  $v(\pi_j)$  can become a negative zero vector, therefore  $|V_\pi^{0^-}| - |V_{\pi'}^{0^-}| = 0$ . Moreover,  $|v(\pi_i)|$  and  $|v(\pi_j)|$  become odd, therefore  $|V_\pi^{odd}| - |V_{\pi'}^{odd}| = -2$ . This means that  $\Delta(\pi, \pi') = 2m - 2$ .
- ii)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  have distinct parities. In this subcase, we have that the parities of the lengths of the vectors  $v(\pi_i)$  and  $v(\pi_j)$  are switched, therefore  $|V_\pi^{odd}| - |V_{\pi'}^{odd}| = 0$ . Moreover, one of the vectors  $v(\pi_i)$  and  $v(\pi_j)$  (precisely the one whose length is odd) may become a negative zero vector, therefore  $|V_\pi^{0^-}| - |V_{\pi'}^{0^-}| \geq -1$ . This means that  $\Delta(\pi, \pi') \geq 2m - 1$ .
- iii)  $|v(\pi_i)|$  and  $|v(\pi_j)|$  are odd. In this subcase, we have that  $|v(\pi_i)|$  and  $|v(\pi_j)|$  become even, therefore  $|V_\pi^{odd}| - |V_{\pi'}^{odd}| = 2$ . On the other hand, we have that the vectors  $v(\pi_i)$  and  $v(\pi_j)$  may become negative zero vectors, therefore  $|V_\pi^{0^-}| - |V_{\pi'}^{0^-}| \geq -2$ . This means that  $\Delta(\pi, \pi') \geq 2m$ .

Note that the only subcase in which we have  $\frac{\Delta(\pi, \pi')}{d} < 1$  is subcase (b.i), precisely when  $m = 1$ . So, assume that we have no choice other than selecting a pair of 1-vector-opposite elements  $\pi_i$  and  $\pi_j$  such that  $|v(\pi_i)|$  and  $|v(\pi_j)|$  are even. We will show that it is still possible to apply a sequence of signed short reversals on  $\pi$  in such a way that (5.4) holds.

Let  $v(\pi_i)$  be the rightmost right vector of  $\pi$ , that is,  $i$  is the largest integer for which  $v(\pi_i)$  is a right vector. As shown in the proof of Lemma 21, there exists an element  $\pi_j$ ,  $j > i$ , such that  $\pi_i$  and  $\pi_j$  form a pair of vector-opposite elements. Combining this fact with our initial assumption, we can conclude that  $j = i + 1$ .

Now, suppose that we apply the signed short reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. From our previous case-by-case analysis, we have that  $\Delta(\pi, \pi') = 0$ . Moreover, we have that  $v(\pi'_{i+1})$  is the rightmost right vector of  $\pi'$ . Therefore, there exists an element  $\pi'_k$ ,  $k > i + 1$ , such that  $\pi'_{i+1}$  and  $\pi'_k$  form a pair of  $m$ -vector-opposite elements, as shown in the proof of Lemma 21. This means that we can apply the  $d$  short signed reversals described in Lemma 28 on  $\pi'$ , obtaining permutation  $\pi''$ . Given that  $|v(\pi'_{i+1})|$  is odd, we can conclude from our previous case-by-case analysis that  $\Delta(\pi', \pi'') \geq 2m - 1$  if  $m$  is odd and  $\Delta(\pi', \pi'') \geq 2m - 2$  if  $m$  is even. Hence, the average decrease in  $\Delta(\pi, \pi'')$  is of at least  $\frac{2m-1}{m+1}$  units per signed



short reversal if  $m$  is odd and of at least  $\frac{2m-2}{m}$  units per signed short reversal if  $m$  is even.

Note that  $\frac{2m-1}{m+1} < 1$  when  $m = 1$ , but in this case we show that the average decrease in  $\Delta(\pi, \pi'')$  is of at least 1 unit per signed short reversal. We have two cases to consider:

- 1)  $|v(\pi'_k)|$  is odd. In this case, we have that  $\Delta(\pi', \pi'') \geq 2$ , therefore the average decrease in  $\Delta(\pi, \pi'')$  is of at least 1 unit per signed short reversal.
- 2)  $|v(\pi'_k)|$  is even. We show that this case cannot happen. For the sake of contradiction, assume that  $|v(\pi'_k)|$  is even. Then, we have that  $|v(\pi'_k)| \geq 2$ . Besides, since  $m = 1$ , we have that  $k = i + 2$ . These two facts imply that  $\pi_i$  and  $\pi_{i+2}$  are 2-vector-opposite elements, but it contradicts our initial hypothesis that we had no choice other than selecting a pair of 1-vector-opposite elements.

Since it is always possible to apply a sequence of  $t$  signed short reversals on  $\pi$  in such a way that the resulting permutation  $\pi'$  satisfies (5.4), the lemma follows.  $\square$

**Lemma 30.** *Let  $\pi$  be a signed permutation. Then, we have that  $d_{ssr}(\pi) \geq \frac{\text{Vec}(\pi) + |V_{\pi}^{odd}| + |V_{\pi}^{0-}|}{5}$ .*

*Proof.* It suffices to prove that if we apply an arbitrary signed short reversal on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Vec}(\pi') + |V_{\pi'}^{odd}| + |V_{\pi'}^{0-}| \geq \text{Vec}(\pi) + |V_{\pi}^{odd}| + |V_{\pi}^{0-}| - 5. \quad (5.5)$$

Suppose first that we apply a signed 1-reversal  $\rho(i, i)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Vec}(\pi') = \text{Vec}(\pi)$  and  $|V_{\pi'}^{odd}| = |V_{\pi}^{odd}|$ . Moreover, since the sign of  $\pi_i$  is flipped without changing the parity of  $|v(\pi_i)|$ , we have that  $|V_{\pi'}^{0-}| \geq |V_{\pi}^{0-}| - 1 \geq |V_{\pi}^{0-}| - 5$ . Summing the previous equalities with this inequality we obtain (5.5).

Suppose now that we apply a signed 2-reversal  $\rho(i, i+1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Vec}(\pi') \geq \text{Vec}(\pi) - 2$ . Moreover, we have that  $|V_{\pi'}^{odd}| \geq |V_{\pi}^{odd}| - 2$  and  $|V_{\pi'}^{0-}| \geq |V_{\pi}^{0-}| - 2$ , but since  $V_{\pi}^{odd} \cap V_{\pi}^{0-} = \emptyset$ , we conclude that  $|V_{\pi'}^{odd}| + |V_{\pi'}^{0-}| \geq |V_{\pi}^{odd}| + |V_{\pi}^{0-}| - 2 \geq |V_{\pi}^{odd}| + |V_{\pi}^{0-}| - 3$ . Summing the previous inequalities we obtain (5.5).

Finally, suppose that we apply a signed 3-reversal  $\rho(i, i+2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that the parities of the lengths of the vectors do not change and hence  $|V_{\pi'}^{odd}| = |V_{\pi}^{odd}|$ . Moreover, we have that  $\text{Vec}(\pi') \geq \text{Vec}(\pi) - 4$  and  $|V_{\pi'}^{0-}| \geq |V_{\pi}^{0-}| - 3$ . It should be noted, however, that if  $v(\pi_i)$  (or  $v(\pi_{i+2})$ ) belongs to  $V_{\pi}^{0-}$ , then  $\text{Vec}(\pi') \geq \text{Vec}(\pi) - 2$  because the length of  $v(\pi_i)$  (or  $v(\pi_{i+2})$ ) increases by 2 units. On the other hand, if neither  $v(\pi_i)$  nor  $v(\pi_{i+2})$  belongs to  $V_{\pi}^{0-}$ , then  $|V_{\pi'}^{0-}| \geq |V_{\pi}^{0-}| - 1$ . Therefore  $\text{Vec}(\pi') + |V_{\pi'}^{0-}| \geq \text{Vec}(\pi) + |V_{\pi}^{0-}| - 5$ . Summing the previous equality with this inequality we obtain (5.5) and the lemma follows.  $\square$

**Theorem 6.** *The problem of sorting by short signed reversals is 5-approximable.*

*Proof.* Immediate from Lemmas 29 and 30. □

Heath and Vergara [78] have described an algorithm for finding vector-opposite elements which runs in linear time on  $n$ , the size of the input permutation. Basically, what their algorithm does is to find vector-opposite elements  $\pi_i$  and  $\pi_j$  such that  $v(\pi_i)$  is the rightmost right vector of  $\pi$ . Algorithm 14 is an adaptation of that algorithm. The difference between the two algorithms is that, given a signed permutation  $\pi \neq \iota_n$ , Algorithm 14 guarantees that, if it returns a pair  $(\pi_i, \pi_{i+1})$ , then  $\pi_i$  and  $\pi_{i+2}$  are not 2-vector-opposite. Note that Algorithm 14 also runs in linear time on  $n$ .

---

**Algorithm 14:** Returns a pair of vector-opposite elements.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .  
**Result:** A pair of vector-opposite elements.

```

1  $i \leftarrow n$ 
2 while  $|\pi_i| \leq i$  do
3   |  $i \leftarrow i - 1$ 
4 end
5  $j \leftarrow i + 1$ 
6 while  $|\pi_j| = j$  do
7   |  $j \leftarrow j + 1$ 
8 end
9 if  $j < n$  and  $j - i = 1$  then
10  | if  $|\pi_{i+2}| < i + 2$  and  $|v(\pi_i)| \geq 2$  and  $|v(\pi_{i+2})| \geq 2$  then
11  |   |  $j \leftarrow i + 2$ 
12  |   end
13 end
14 return  $(\pi_i, \pi_j)$ 

```

---



---

**Algorithm 15:** Algorithm for sorting by signed short reversals.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .  
**Result:** Number of signed short reversals applied for sorting  $\pi$ .

```

1  $d \leftarrow 0$ 
2 while  $\text{Vec}(\pi) > 0$  do
3   | Let  $\pi_i$  and  $\pi_j$  be  $m$ -vector opposite elements returned by Algorithm 14
4   | Apply signed short reversals on  $\pi$  such as described in Lemma 28
5   |  $d \leftarrow d + 2^{\lceil \frac{m}{2} \rceil} - 1$ 
6 end
7 Apply signed 1-reversals on  $\pi$  until it has no negative elements and update  $d$  accordingly
8 return  $d$ 

```

---

Algorithm 15 sorts a signed permutation in two steps. While the signed permutation has vector-opposite elements, the algorithm finds a pair of them using Algorithm 14 and then switches their positions applying the signed short reversals described in Lemma 28. When the signed permutation has no vector-opposite elements, the algorithm applies signed 1-reversals until the signed permutation has no negative elements.

It follows from Theorem 6 that Algorithm 15 is a 5-approximation algorithm for the problem of sorting by short signed reversals. Regarding its time complexity, it suffices to compute the total cost of calls to lines 3, 4, and 7. The total cost of calls in line 3 equals the total cost for all calls to Algorithm 14. Although it runs in  $O(n)$  time and there are  $O(n^2)$  vector-opposite elements in a signed permutation, we can provide the Algorithm 14 with enough information so that the costs of calls to this algorithm can be significantly reduced. Note that Algorithm 14 performs two scans in the signed permutation, one for each vector of the vector-opposite elements returned. By observing that a rightmost right vector remains a rightmost vector until it becomes a zero vector, it need not be searched again if the vector has not been zeroed. Thus, the scan for the rightmost vector needs to be performed only  $O(n)$  times. In addition, the total cost of scans for the left vector for the same right vector is bounded by the length of the right vector, also  $O(n)$ . The total cost for all calls to Algorithm 14 with this refinement is thus  $O(n^2)$ . Each call to line 4 takes  $O(m)$  time, where  $m = j - i$ , and causes a strict decrease in  $\text{Vec}(\pi)$  of  $2m$  units. Thus, the cost in this case is bounded by  $\text{Vec}(\pi)$  rather than the number of iterations performed in the while loop. As each vector has length at most  $n$ , we have that  $\text{Vec}(\pi) \leq n^2$ , meaning a cost of  $O(n^2)$  time for the calls to line 4. Finally, we have that line 3 runs in  $O(n)$  time, therefore Algorithm 15 runs in  $O(n^2)$  time.

We finish by noting that there exists a large class of signed permutations for which the approximation ratio of Algorithm 15 is much lower than its worst-case approximation ratio (Lemma 31). Moreover, based on the fact that the expected value of  $\text{Vec}(\pi)$  of a random signed permutation  $\pi \in S_n^\pm$  is  $\frac{n^2-1}{3}$  (Lemma 33), we can conclude that the expected approximation ratio of Algorithm 15 for sorting a random signed permutation is also lower than the worst-case approximation ratio (Theorem 7). Just to make things clear, we define a random signed permutation as a random ordering of the elements  $\{1, 2, \dots, n\}$ , with the added characteristic that the sign,  $+$  or  $-$ , of each element is also randomly chosen.

**Lemma 31.** *Let  $A_{15}(\pi)$  be the number of signed short reversals applied by Algorithm 15 for sorting a signed permutation  $\pi \in S_n^\pm$ . We have that  $\frac{A_{15}(\pi)}{d_{ssr}(\pi)} \leq 3$  when  $\text{Vec}(\pi) = 0$  or  $\text{Vec}(\pi) \geq 4n$ .*

*Proof.* We have two cases to consider:

- a)  $\text{Vec}(\pi) = 0$ . In this case, we have that Algorithm 15 sorts  $\pi$  with  $|V_\pi^{0^-}|$  signed 1-reversals. On the other hand, we have that  $d_{ssr}(\pi) \geq \frac{|V_\pi^{0^-}|}{3}$  because a signed

short reversal cannot affect more than 3 elements at once. Therefore  $\frac{A_{15}(\pi)}{d_{ssr}(\pi)} \leq 3$ .

- b)  $\text{Vec}(\pi) \geq 4n$ . In this case, we have seen that Algorithm 15 sorts  $\pi$  in two steps. First it applies signed 2-reversals and signed 3-reversals on  $\pi$  until  $\text{Vec}(\pi) = 0$  and then it applies signed 1-reversals on  $\pi$  until  $|V_{\pi}^{0-}| = 0$ . Note that, in the first step, each signed short reversal applied by Algorithm 15 results in an average decrease in  $\text{Vec}(\pi)$  of at least 2 units. Hence Algorithm 15 applies at most  $\frac{\text{Vec}(\pi)}{2}$  signed short reversals in the first step. Moreover, Algorithm 15 applies at most  $n$  signed 1-reversals in the second step because  $|V_{\pi}^{0-}| \leq n$ . On the other hand, we have that  $d_{ssr}(\pi) \geq \frac{\text{Vec}(\pi)}{4}$  (Lemma 23). This analysis lead us to conclude that  $\frac{A_{15}(\pi)}{d_{ssr}(\pi)} \leq 2 + \frac{4n}{\text{Vec}(\pi)}$ . Therefore  $\frac{A_{15}(\pi)}{d_{ssr}(\pi)} \leq 3$ .

Since  $\frac{A_{15}(\pi)}{d_{ssr}(\pi)} \leq 3$  in both cases, the lemma follows.  $\square$

In what follows, let  $\Pr(|v(\pi_i)| = j)$  denote the probability that  $|v(\pi_i)|$  is equal to  $j$  and  $\mathbb{E}(X)$  denote the expected value of a random variable  $X$ .

**Lemma 32.** *Let  $\pi \in S_n^{\pm}$  be a random signed permutation. Then  $\sum_{i=1}^n \Pr(|v(\pi_i)| = j) = \frac{2(n-j)}{n}$  for  $1 \leq j \leq n-1$ .*

*Proof.* We have that  $|S_n^{\pm}| = n!2^n$  and for each  $1 \leq k \leq n$ , there are  $(n-1)!2^n$  signed permutations for which  $|\pi_i| = k$ . Then

$$\Pr(|v(\pi_i)| = j) = \begin{cases} \frac{1}{n} & \text{if } j = 0, \\ \frac{2}{n} & \text{if } i + j \leq n \text{ and } i - j \geq 1, \\ \frac{1}{n} & \text{if } i + j > n \text{ or } i - j < 1 \text{ but not both,} \\ 0 & \text{otherwise,} \end{cases}$$

for  $0 \leq j \leq n-1$ . In order to evaluate  $\sum_{i=1}^n \Pr(|v(\pi_i)| = j)$  for a given  $j$ , we consider two cases:

- a)  $1 \leq j < \frac{n}{2}$ . In this case, we have that

$$\Pr(|v(\pi_i)| = j) = \begin{cases} \frac{1}{n} & \text{if } 1 \leq i \leq j, \\ \frac{1}{n} & \text{if } n - j + 1 \leq i \leq n, \\ \frac{2}{n} & \text{otherwise.} \end{cases}$$

Therefore, we have that  $\sum_{i=1}^n \Pr(|v(\pi_i)| = j) = \frac{j}{n} + \frac{j}{n} + \frac{2(n-2j)}{n} = \frac{2(n-j)}{n}$ .

- b)  $\frac{n}{2} \leq j \leq n$ . In this case, we have that

$$\Pr(|v(\pi_i)| = j) = \begin{cases} \frac{1}{n} & \text{if } 1 \leq i \leq n - j, \\ \frac{1}{n} & \text{if } j + 1 \leq i \leq n, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, we have that  $\sum_{i=1}^n \Pr(|v(\pi_i)| = j) = \frac{n-j}{n} + \frac{n-j}{n} = \frac{2(n-j)}{n}$ .

Since in both cases  $\sum_{i=1}^n \Pr(|v(\pi_i)| = j) = \frac{2(n-j)}{n}$  holds, the lemma follows.  $\square$

**Lemma 33.** *Let  $\pi \in S_n^\pm$  be a random signed permutation. Then  $\mathbb{E}(\text{Vec}(\pi)) = \frac{n^2-1}{3}$ .*

*Proof.* Given that  $\mathbb{E}(|v(\pi_i)|) = \sum_{j=0}^{n-1} j \Pr(|v(\pi_i)| = j)$ , we have that

$$\begin{aligned}
\mathbb{E}(\text{Vec}(\pi)) &= \mathbb{E}(\sum_{i=1}^n |v(\pi_i)|) \\
&= \sum_{i=1}^n \mathbb{E}(|v(\pi_i)|) \\
&= \sum_{i=1}^n \sum_{j=0}^{n-1} j \Pr(|v(\pi_i)| = j) \\
&= \sum_{j=1}^{n-1} j \sum_{i=1}^n \Pr(|v(\pi_i)| = j) \\
&= \sum_{j=1}^{n-1} j \frac{2(n-j)}{n} \\
&= 2 \sum_{j=1}^{n-1} j - \frac{2}{n} \sum_{j=1}^{n-1} j^2 \\
&= 2 \left( \frac{n^2-n}{2} \right) - \frac{2}{n} \left( \frac{(n-1)n(2n-1)}{6} \right) \\
&= n^2 - n - \frac{2n^2-3n+1}{3} \\
&= \frac{n^2-1}{3},
\end{aligned}$$

and the lemma follows.  $\square$

**Theorem 7.** *The expected approximation ratio of Algorithm 15 for sorting a random signed permutation  $\pi \in S_n^\pm$  is no greater than 3 for  $n \geq 13$ .*

*Proof.* According to Lemma 31, we have that the approximation ratio of Algorithm 15 for sorting a given signed permutation  $\sigma \in S_n^\pm$  is no greater than 3 when  $\text{Vec}(\sigma) \geq 4n$ . Since we know that the expected value of  $\text{Vec}(\pi)$  of a random signed permutation  $\pi \in S_n^\pm$  is  $\frac{n^2-1}{3}$  (Lemma 33), we conclude that the expected approximation ratio of Algorithm 15 for sorting  $\pi$  is no greater than 3 if  $\frac{n^2-1}{3} \geq 4n$ . This inequality holds when  $n \geq 13$ , and the theorem follows.  $\square$

## 5.4 Sorting by Bounded Operations

In this section, we present a polynomial-time solution for the problem of sorting by super short operations and a 3-approximation algorithm for the problem of sorting by short operations. Before we present the main results, we first introduce a useful tool for tackling these problems, the *permutation graph*. This tool was also used by Heath and Vergara [77] for dealing with the problem of sorting by short transpositions.

### 5.4.1 The Permutation Graph

The *permutation graph* of a permutation  $\pi \in S_n^\pm$  is the undirected graph  $G_\pi = (V, E)$ , where  $V = \{\pi_1, \pi_2, \dots, \pi_n\}$  and  $E = \{(\pi_i, \pi_j) : i < j \text{ and } |\pi_i| > |\pi_j|\}$ . In other words,  $G_\pi$  is an undirected graph whose vertex set is formed by the elements of  $\pi$  and edge set is formed by the inversions in  $\pi$ . Figure 5.2 illustrates  $G_\pi$  for  $\pi = (+3 -4 +6 -1 +5 -2)$ .

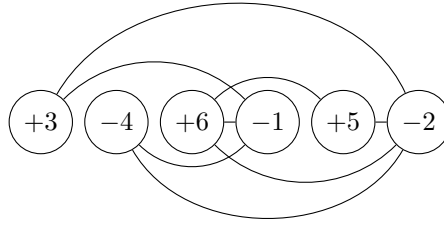


Figure 5.2: Permutation graph of the signed permutation  $(+3 -4 +6 -1 +5 -2)$ .

Given a signed permutation  $\pi$ , we denote the number of connected components (or simply components) of  $G_\pi$  by  $c(\pi)$ . Moreover, we say that a component of  $G_\pi$  is *odd* if it contains an odd number of negative elements (vertices) and we say it is *even* otherwise. The number of odd components of  $G_\pi$  is denoted by  $c_{\text{odd}}(\pi)$ . Lastly, we say that an edge of  $G_\pi$  is a *cut-edge* if its deletion increases the number of components of  $G_\pi$ .

### 5.4.2 Sorting by Signed Super Short Operations

From the proof of Lemma 20, we have that a super short operation can eliminate at most one inversion of a signed permutation. This means that, for sorting a signed permutation  $\pi$ , we have to apply  $\text{Inv}(\pi)$  super short operations (*i.e.* 2-reversals and  $(1, 1)$ -transpositions) plus a given number of signed 1-reversals in order to flip the signs of the remaining negative elements. As before, the question is: how many signed 1-reversals do we have to apply? As Lemmas 34 and 35 show, the answer is  $c_{\text{odd}}(\pi)$ .

**Lemma 34.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then, we have that  $d_{\text{ssso}}(\pi) \leq \text{Inv}(\pi) + c_{\text{odd}}(\pi)$ .*

*Proof.* It suffices to prove that it is always possible to apply a signed super short operation on  $\pi \neq \iota_n$  in such a way that the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + c_{\text{odd}}(\pi') \leq \text{Inv}(\pi) + c_{\text{odd}}(\pi) - 1. \quad (5.6)$$

If  $\text{Inv}(\pi) = 0$ , then each component of  $G_\pi$  is a single vertex. Therefore, we can sort  $\pi$  with  $c_{\text{odd}}(\pi)$  signed 1-reversals and (5.6) holds.

If  $\text{Inv}(\pi) > 0$ , then there exists an edge  $e = (\pi_i, \pi_{i+1})$  in  $G_\pi$  (Lemma 19). Suppose first that  $e$  is not a cut-edge and that we apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$ , obtaining the permutation  $\pi'$ . We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ . Moreover, since  $e$  is not a cut-edge, we have that the vertex sets of the components of  $G_{\pi'}$  are the same as of the components of  $G_\pi$ . This means that  $c_{\text{odd}}(\pi') = c_{\text{odd}}(\pi)$ . Summing both equalities we obtain (5.6).

Now, suppose that  $e$  is a cut-edge and let  $C$  denote the component of  $G_\pi$  which contains  $e$ . Moreover, let  $C_1$  and  $C_2$  denote the components of  $C - e$  and assume, without loss of generality, that  $\pi_i \in C_1$ . We have three cases to consider:

- a)  $C_1$  and  $C_2$  are both even. Note that  $C$  is even. Apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$  and that  $c_{\text{odd}}(\pi') = c_{\text{odd}}(\pi)$ . Summing both equalities we obtain (5.6).
- b)  $C_1$  and  $C_2$  have distinct parities. Note that  $C$  is odd. Apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$  and that  $c_{\text{odd}}(\pi') = c_{\text{odd}}(\pi)$ . Summing both equalities we obtain (5.6).
- c)  $C_1$  and  $C_2$  are both odd. Note that  $C$  is even. Apply the signed 2-reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ . Moreover, we have that  $c_{\text{odd}}(\pi') = c_{\text{odd}}(\pi)$  because  $C_1$  and  $C_2$  become even after the signed reversal is applied on  $\pi$ . Summing both equalities we obtain (5.6).

Since it is always possible to apply a signed super short operation on  $\pi$  in such a way that the resulting permutation  $\pi'$  satisfies (5.6), the lemma follows.  $\square$

**Lemma 35.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then  $d_{\text{ssso}}(\pi) \geq \text{Inv}(\pi) + c_{\text{odd}}(\pi)$ .*

*Proof.* It suffices to prove that if we apply an arbitrary super short operation on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + c_{\text{odd}}(\pi') \geq \text{Inv}(\pi) + c_{\text{odd}}(\pi) - 1. \quad (5.7)$$

Suppose first that we apply a signed 1-reversal  $\rho(i, i)$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi)$ . Moreover, since the component containing  $\pi_i$  may become even, we have that  $c_{\text{odd}}(\pi') \geq c_{\text{odd}}(\pi) - 1$ . Summing the previous equality with this inequality we obtain (5.7).

Now, suppose that we apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have two cases to consider:

- a)  $(\pi_i, \pi_{i+1})$  is not an inversion. In this case, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) + 1$ . On the other hand, by adding a new edge, we may eliminate two odd components, therefore  $c_{\text{odd}}(\pi') \geq c_{\text{odd}}(\pi) - 2$ . Summing the previous equality with this inequality we obtain (5.7).
- b)  $(\pi_i, \pi_{i+1})$  is an inversion. In this case, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ . Moreover, let  $e = (\pi_i, \pi_{i+1})$  be an edge of  $G_\pi$  and let  $C$  be the component of  $G_\pi$  containing  $e$  and. We further divide our analysis into two subcases:
- i)  $e$  is not a cut-edge. In this case, we have that  $c_{\text{odd}}(\pi') = c_{\text{odd}}(\pi)$  because the parity of the component  $C - e$  is the same as of  $C$ , therefore (5.7) holds.

- ii)  $e$  is a cut-edge. In this case, let  $C_1$  and  $C_2$  denote the components of  $C - e$ . If  $C$  is odd, then either  $C_1$  or  $C_2$  is odd. If  $C$  is even, then either  $C_1$  and  $C_2$  are both odd or  $C_1$  and  $C_2$  are both even. In any case, we have that  $c_{odd}(\pi') \geq c_{odd}(\pi)$ , therefore (5.7) holds.

Finally, suppose that we apply the signed 2-reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. By making use of an argument analogous to the one in the previous paragraph, we conclude that  $\pi'$  satisfies (5.7) and the lemma follows.  $\square$

**Theorem 8.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then,  $d_{ssso}(\pi) = \text{Inv}(\pi) + c_{odd}(\pi)$ .*

*Proof.* Immediate from Lemmas 34 and 35.  $\square$

Let  $\pi$  be a signed permutation. From the proof of Lemma 35, we can conclude that a super short operation cannot decrease the value of  $c_{odd}(\pi)$  if it is applied on an inversion in  $\pi$ . Moreover, from the proof of Lemma 34, we can conclude that if a  $(1, 1)$ -transposition increases the value of  $c_{odd}(\pi)$  when applied on an inversion in  $\pi$ , then it is possible to apply a signed 2-reversal on this inversion in such a way that  $c_{odd}(\pi)$  remains unaltered. These observations lead us to the following optimal algorithm for sorting by signed super short operations (Algorithm 16).

The time complexity of Algorithm 16 depends on the time complexity of the algorithm used to compute the value of  $c_{odd}(\pi)$ . A straightforward algorithm is to traverse  $G_\pi$  with a depth-first search and count the number of odd components. Such an algorithm runs in  $O(n^2)$  time. It is possible, however, to count the number of odd components in  $G_\pi$  in  $O(n)$  time.

Koh and Ree [86] have studied the permutation graph of unsigned permutations and have demonstrated some useful properties about them. Since the permutation graph of the signed permutation  $\pi \in S_n^\pm$  is isomorphic to the permutation graph of the unsigned permutation  $(|\pi_1| |\pi_2| \dots |\pi_n|)$ , we are able to translate those properties to the permutation graph of signed permutations. In particular, Lemma 36 represents the translation of one of those properties.

**Lemma 36.** *Let  $\pi \in S_n^\pm$  be a signed permutation. The vertex sets of the components of  $G_\pi$  are of the form  $C_1 = \{\pi_1, \pi_2, \dots, \pi_k\}$ ,  $C_2 = \{\pi_{k+1}, \pi_{k+2}, \dots, \pi_l\}$ ,  $\dots$ ,  $C_t = \{\pi_{m+1}, \pi_{m+2}, \dots, \pi_n\}$ . Moreover, we have that  $\{|\pi_1|, |\pi_2|, \dots, |\pi_k|\} = \{1, 2, \dots, k\}$ ,  $\{|\pi_{k+1}|, |\pi_{k+2}|, \dots, |\pi_l|\} = \{k + 1, k + 2, \dots, l\}$ ,  $\dots$ ,  $\{|\pi_{m+1}|, |\pi_{m+2}|, \dots, |\pi_n|\} = \{m + 1, m + 2, \dots, n\}$ .*

We say that a contiguous sequence of elements  $\pi_i \pi_{i+1} \dots \pi_j$ ,  $i \leq j$ , of a signed permutation  $\pi$  is a *complete substring* if  $\{|\pi_i|, |\pi_{i+1}|, \dots, |\pi_j|\} = \{i, i + 1, \dots, j\}$ . From Lemma 36, we have that the vertex set of a component of  $G_\pi$  forms a complete substring. Furthermore, assume that  $\{\pi_i, \pi_{i+1}, \dots, \pi_j\}$  is the vertex set of a component of  $G_\pi$ . We claim that  $\pi_i \pi_{i+1} \dots \pi_j$  is the minimum complete substring that starts



---

**Algorithm 16:** Optimal algorithm for sorting by super short operations.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .  
**Result:** Number of super short operations applied for sorting  $\pi$ .

```

1  $d \leftarrow 0$ 
2  $c_{odd} \leftarrow c_{odd}(\pi)$ 
3 while  $\text{Inv}(\pi) > 0$  do
4   Let  $(\pi_i, \pi_{i+1})$  be an inversion in  $\pi$ 
5    $\pi \leftarrow \pi \cdot \rho(i, i+1, i+2)$ 
6   if  $c_{odd}(\pi) > c_{odd}$  then
7      $\pi \leftarrow \pi \cdot \rho(i, i+1, i+2)$      $\triangleright$  undo the previous  $(1, 1)$ -transposition
8      $\pi \leftarrow \pi \cdot \rho(i, i+1)$ 
9   end
10   $d \leftarrow d + 1$ 
11 end
12 Apply signed 1-reversals on  $\pi$  until it has no negative elements and update  $d$ 
   accordingly
13 return  $d$ 

```

---

with  $\pi_i$ . For the sake of contradiction, suppose that there exists a complete substring  $\pi_i \pi_{i+1} \dots \pi_k$  such that  $k < j$ . We have that  $\pi_l > \pi_m$  for every  $i \leq l \leq k$  and  $k+1 \leq m \leq j$ . Therefore there does not exist any edge in  $G_\pi$  connecting the elements in  $\{\pi_i, \pi_{i+1}, \dots, \pi_k\}$  with the elements in  $\{\pi_{k+1}, \pi_{k+2}, \dots, \pi_j\}$ . But this contradicts our hypothesis that  $\{\pi_i, \pi_{i+1}, \dots, \pi_j\}$  is the vertex set of a component of  $G_\pi$ .

From the discussion of the last paragraph, we can design the following algorithm for finding the vertex sets of the components of the permutation graph of a signed permutation  $\pi \in S_n^\pm$ . Find the minimum complete substring  $\pi_1 \pi_2 \dots \pi_k$  starting with  $\pi_1$  and let  $C_1 = \{\pi_1, \pi_2, \dots, \pi_k\}$  be a component of  $G_\pi$ . If  $k < n$ , then find the minimum complete substring  $\pi_{k+1} \pi_{k+2} \dots \pi_l$  starting with  $\pi_{k+1}$  and let  $C_2 = \{\pi_{k+1}, \pi_{k+2}, \dots, \pi_l\}$  be another component of  $G_\pi$ . Continue with this process until all elements have been assigned to a component. It remains to show how to find the minimum complete substring  $\pi_i \pi_{i+1} \dots \pi_j$  starting with  $\pi_i$ . Note that  $i$  is the least element and  $j$  is the largest element of the set  $S = \{|\pi_i|, |\pi_{i+1}|, \dots, |\pi_j|\}$ . Since all integers in the interval  $[i, j]$  are in  $S$ , we have that  $|S| = j - i + 1$ . This fact give us the necessary and sufficient condition for knowing when we have found the last element of the minimum complete substring starting with  $\pi_i$ . The complete algorithm is detailed below (Algorithm 17).

Algorithm 17 performs a linear scan on the positions of the permutation  $\pi \in S_n^\pm$ , and so it runs in  $O(n)$ . With the vertex sets of the components of  $G_\pi$ , it is easy to count the number of odd components in  $G_\pi$  in  $O(n)$  time. Returning to Algorithm 16, we can see that lines 4-9 run in  $O(n)$  time. Since the while loop iterates a total of  $O(n^2)$  times and line 12 runs in  $O(n)$  time, we can conclude that Algorithm 16 runs in  $O(n^3)$  time. We remark that the value of  $d_{ssso}(\pi)$  can be computed in  $O(n \sqrt{\log n})$  time

---

**Algorithm 17:** Find the vertex sets of the components of a permutation graph.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .

**Result:** The vertex sets of the components of  $G_\pi$ .

```

1  $C \leftarrow \emptyset$ 
2  $S \leftarrow \emptyset$ 
3  $i \leftarrow 1$ 
4 while  $i \leq n$  do
5    $C \leftarrow C \cup \{\pi_i\}$ 
6    $\min \leftarrow i$ 
7    $\max \leftarrow |\pi_i|$ 
8   while  $(\max - \min + 1) > |C|$  do
9      $i \leftarrow i + 1$ 
10     $C \leftarrow C \cup \{\pi_i\}$ 
11    if  $|\pi_i| > \max$  then
12       $\max \leftarrow |\pi_i|$ 
13    end
14  end
15   $S \leftarrow S \cup C$ 
16   $C \leftarrow \emptyset$ 
17   $i \leftarrow i + 1$ 
18 end
19 return  $S$ 

```

---

because computing  $c_{\text{odd}}(\pi)$  takes  $O(n)$  time and computing  $\text{Inv}(\pi)$  takes  $O(n\sqrt{\log n})$  time [26].

### 5.4.3 Sorting by Signed Short Operations

A trivial algorithm for the problem of sorting by signed short operations is the optimal algorithm for the problem of sorting by signed super short operations. From the lower bound of Lemma 37, it follows that this algorithm is a 4-approximation algorithm. In addition, we have that this approximation bound is tight. For instance, we need 4 signed super short operations for sorting the signed permutation  $(-3 -2 -1)$ , but one signed 3-reversal is sufficient for sorting it.

**Lemma 37.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then,  $d_{\text{SSO}}(\pi) \geq \frac{\text{Inv}(\pi) + c_{\text{odd}}(\pi)}{4}$ .*

*Proof.* It suffices to prove that if we apply an arbitrary short operation on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + c_{\text{odd}}(\pi') \geq \text{Inv}(\pi) + c_{\text{odd}}(\pi) - 4. \quad (5.8)$$

From the proof of Lemma 35, we have that (5.8) holds in case we apply a super short operation on  $\pi$ . So, suppose that we apply a short operation  $\rho$  on  $\pi$  which acts

on the elements  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$ . Moreover, let  $\pi'$  denote the resulting permutation. We have three cases to consider:

- a)  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$  belong to the same component. In this case, we have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 3$  and  $c_{\text{odd}}(\pi') \geq c_{\text{odd}}(\pi) - 1$ , therefore (5.8) holds.
- b) two elements in  $\{\pi_i, \pi_{i+1}, \pi_{i+2}\}$  belong to a component  $C_1$  and the remaining element belongs to a component  $C_2$ . In this case, we have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 1$  and  $c_{\text{odd}}(\pi') \geq c_{\text{odd}}(\pi) - 2$ , therefore (5.8) holds.
- c)  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$  belong to distinct components. In this case, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) + 3$  and  $c_{\text{odd}}(\pi') \geq c_{\text{odd}}(\pi) - 3$ , therefore (5.8) holds.

Since (5.8) holds in any case, the lemma follows.  $\square$

Given a signed permutation  $\pi$ , let  $c_{\text{odd}}^t(\pi)$  be the number of odd components of  $G_\pi$  which have exactly  $t$  vertices. By just considering the odd components having at most two vertices, we can obtain better bounds on the signed short operation distance of a signed permutation  $\pi$  (Lemmas 39 and 40). These bounds lead to a 3-approximation for the problem of sorting by signed short reversals (Theorem 9). We note that the upper bound given in Lemma 39 relies on the fact that we can establish an isomorphism between a component with  $m$  vertices and the permutation graph of a signed permutation  $\sigma \in S_m^\pm$  (Lemma 38).

**Lemma 38.** *Let  $\pi \in S_n^\pm$  be a signed permutation and let  $C = (V_C, E_C)$  be a component of  $G_\pi$  with  $m$  vertices. Then, there exists a signed permutation  $\sigma \in S_m^\pm$  such that  $G_\sigma$  is isomorphic to  $C$ .*

*Proof.* By Lemma 36, we have that if  $V_C = \{\pi_{i+1}, \pi_{i+2}, \dots, \pi_{i+m}\}$ , then  $\{|\pi_{i+1}|, |\pi_{i+2}|, \dots, |\pi_{i+m}|\} = \{i+1, i+2, \dots, i+m\}$ . Let  $\sigma \in S_m^\pm$  be a signed permutation such that

$$\sigma_j = \begin{cases} \pi_{i+j} - i & \text{if } \pi_{i+j} > 0 \\ \pi_{i+j} + i & \text{if } \pi_{i+j} < 0 \end{cases}$$

for all  $j \in \{1, 2, \dots, m\}$ . We claim that the bijective function  $f(\pi_{i+x}) = \sigma_x$  is an isomorphism between  $C$  and  $G_\sigma$ . To see this, firstly note that  $\pi_{i+x}$  is a negative vertex if, and only if,  $\sigma_x$  is a negative vertex. Secondly, let  $k$  and  $l$  be to integers such that  $1 \leq k < l \leq m$ . Note that  $(\pi_{i+k}, \pi_{i+l})$  is an edge of  $C$  if, and only if,  $(\sigma_k, \sigma_l)$  is an edge of  $G_\sigma$ , and so the lemma follows.  $\square$

**Lemma 39.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then  $d_{\text{SSO}}(\pi) \leq \text{Inv}(\pi) + c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)$ .*

*Proof.* It suffices to prove that it is always possible to apply a sequence of  $t > 0$  signed short operations on  $\pi \neq \iota_n$  in such a way that the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \leq \text{Inv}(\pi) + c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - t. \quad (5.9)$$

If  $\text{Inv}(\pi) = 0$ , then each component of  $G_\pi$  is a single vertex. Therefore, we can apply  $c_{\text{odd}}^1(\pi)$  signed 1-reversals and (5.9) holds.

If  $\text{Inv}(\pi) > 0$ , then there exists an edge  $e = (\pi_i, \pi_{i+1})$  in  $G_\pi$  (Lemma 19). Let  $C$  denote the component of  $G_\pi$  which contains  $e$  and assume that  $C$  contains  $m$  vertices. We have four cases to consider:

- a)  $m \geq 5$ . In this case, we further divide our analysis into two subcases:
  - i)  $e$  is not a cut-edge. In this case, apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ . Therefore (5.9) holds.
  - ii)  $e$  is a cut-edge. In this case, let  $C_1$  and  $C_2$  denote the components of  $C - e$ . Moreover, let  $m_1$  be the number of vertices in  $C_1$  and let  $m_2$  be the number of vertices in  $C_2$ . If  $m_1 \geq 3$  and  $m_2 \geq 3$ , then apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ . So, without loss of generality, assume that  $m_1 \leq 2$ . Note that  $m_2 \geq 3$  because  $m_1 + m_2 = m \geq 5$ . If  $C_1$  is even, then apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ . Otherwise, if  $C_1$  is odd, apply the signed the 2-reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ . In any case, we have that the resulting permutation  $\pi'$  satisfies (5.9).
- b)  $m = 4$ . According to Lemma 38, there exists a signed permutation  $\sigma \in S_4^\pm$  such that  $G_\sigma$  is isomorphic to  $C$ . We have verified that every permutation  $\sigma \in S_4^\pm$  for which  $c(\sigma) = 1$  can be sorted with at most  $\text{Inv}(\sigma)$  signed short operations, therefore it is possible to apply a sequence of signed short operations on  $C$  in such a way that the resulting permutation  $\pi'$  satisfies (5.9).
- c)  $m = 3$ . Analogous to case b).
- d)  $m = 2$ . In this case, we further divide our analysis into three subcases:
  - i)  $\pi_i$  and  $\pi_{i+1}$  are both negatives. In this case, apply the signed the 2-reversal  $\rho(i, i + 1)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. We have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ , therefore (5.9) holds.
  - ii)  $\pi_i$  and  $\pi_{i+1}$  have distinct signs. In this case, apply the  $(1, 1)$ -transposition  $\rho(i, i + 1, i + 2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi) - 1$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi) + 1$ , therefore (5.9) holds.

- iii)  $\pi_i$  and  $\pi_{i+1}$  are both positives. In this case, apply the  $(1, 1)$ -transposition  $\rho(i, i+1, i+2)$  on  $\pi$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$ ,  $c_{\text{odd}}^2(\pi') = c_{\text{odd}}^2(\pi)$ , and  $c_{\text{odd}}^1(\pi') = c_{\text{odd}}^1(\pi)$ , therefore (5.9) holds.

Since it is always possible to apply a sequence of signed short operations on  $\pi$  in such a way that the resulting permutation  $\pi'$  satisfies (5.9), the lemma follows.  $\square$

**Lemma 40.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then, we have that  $d_{\text{sso}}(\pi) \geq \frac{\text{Inv}(\pi) + c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)}{3}$ .*

*Proof.* It suffices to prove that if we apply an arbitrary short operation on  $\pi$ , then the resulting permutation  $\pi'$  satisfies

$$\text{Inv}(\pi') + c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq \text{Inv}(\pi) + c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - 3. \quad (5.10)$$

Suppose first that we apply a signed 1-reversal  $\rho(i, i)$  and let  $\pi'$  denote the resulting permutation. Then, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi)$ . Moreover, since  $\pi_i$  can belong to an odd component with at most two vertices, we have that  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - 1$ , therefore (5.10) holds.

Now, suppose that we apply a super short operation  $\rho$  on  $\pi$  which acts on the elements  $\pi_i$  and  $\pi_{i+1}$ , and let  $\pi'$  denote the resulting permutation. We have two cases to consider:

- a)  $\pi_i$  and  $\pi_{i+1}$  belong to the same component. In this case, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) - 1$  and  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)$ , and (5.10) holds.
- b)  $\pi_i$  and  $\pi_{i+1}$  belong to distinct components. In this case, we have that  $\text{Inv}(\pi') = \text{Inv}(\pi) + 1$  and  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - 2$ . Therefore (5.10) holds.

Finally, suppose that we apply a short operation  $\rho$  on  $\pi$  which acts on the elements  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$ . Moreover, let  $\pi'$  denote the resulting permutation. We have three cases to consider:

- a)  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$  belong to the same component. In this case, we have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 3$  and  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)$ . Therefore (5.10) holds.
- b) two elements in  $\{\pi_i, \pi_{i+1}, \pi_{i+2}\}$  belong to the component  $C_1$  and the remaining element belongs to the component  $C_2$ . In this case, we have that  $\text{Inv}(\pi') \geq \text{Inv}(\pi) - 1$  and  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - 2$ , and (5.10) holds.
- c)  $\pi_i$ ,  $\pi_{i+1}$ , and  $\pi_{i+2}$  belong to distinct components. In this case, we have that  $\pi_i < \pi_{i+1} < \pi_{i+2}$ , thus  $\text{Inv}(\pi') = \text{Inv}(\pi) + 3$ . Moreover, we have that  $c_{\text{odd}}^2(\pi') + c_{\text{odd}}^1(\pi') \geq c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) - 3$ . Therefore (5.10) holds.

Since (5.10) holds in every case, the lemma follows.  $\square$

**Theorem 9.** *The problem of sorting by short signed operations is 3-approximable.*

*Proof.* Immediate from Lemmas 39 and 40.  $\square$

Let  $\pi$  be a signed permutation. From the proof of Lemma 39, we can conclude that as long as  $\text{Inv}(\pi) > 0$ , we can apply a sequence of short operations that eliminates inversions and keeps the value of  $c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)$  unchanged. When  $\text{Inv}(\pi) = 0$ , we can sort  $\pi$  applying  $c_{\text{odd}}^1(\pi)$  signed 1-reversals. This is precisely what Algorithm 18 does.

---

**Algorithm 18:** Algorithm for sorting by short operations.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .

**Result:** Number of short operations applied for sorting  $\pi$ .

```

1  $d \leftarrow 0$ 
2  $c_{\text{odd}} \leftarrow c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi)$ 
3 while  $\text{Inv}(\pi) > 0$  do
4   Let  $(\pi_i, \pi_{i+1})$  be an inversion in  $\pi$ 
5   Let  $C = (V_C, E_C)$  be the component of  $G_\pi$  such that  $\pi_i, \pi_{i+1} \in V_C$ 
6   if  $|V_C| \geq 5$  then
7      $\pi \leftarrow \pi \cdot \rho(i, i + 1, i + 2)$ 
8     if  $c_{\text{odd}}^2(\pi) + c_{\text{odd}}^1(\pi) > c_{\text{odd}}$  then
9        $\pi \leftarrow \pi \cdot \rho(i, i + 1, i + 2) \triangleright$  undo the previous (1, 1)-transposition
10       $\pi \leftarrow \pi \cdot \rho(i, i + 1)$ 
11    end
12     $d \leftarrow d + 1$ 
13  else if  $|V_C| = 4$  or  $|V_C| = 3$  then
14    Let  $m = |V_C|$  and let  $\sigma \in S_m^\pm$  be a signed permutation such that  $G_\sigma \simeq C$  (Lemma 38)
15    Apply on  $C$  the sequence of short operations that optimally sorts  $\sigma$ 
16     $d \leftarrow d + d_{\text{SSO}}(\sigma)$ 
17  else
18    if  $\pi_i < 0$  and  $\pi_{i+1} < 0$  then
19       $\pi \leftarrow \pi \cdot \rho(i, i + 1)$ 
20    else
21       $\pi \leftarrow \pi \cdot \rho(i, i + 1, i + 2)$ 
22    end
23     $d \leftarrow d + 1$ 
24  end
25 end
26 Apply signed 1-reversals on  $\pi$  until it has no negative elements and update  $d$  accordingly
27 return  $d$ 

```

---

It follows from Theorem 9 that Algorithm 18 is a 3-approximation algorithm for the problem of sorting by short reversals. Regarding its time complexity, we have that each iteration of the while loop takes  $O(n)$  time. Since the while loop iterates a total of  $O(n^2)$  times and line 26 runs in  $O(n)$  time, we can conclude that Algorithm 18 runs in  $O(n^3)$  time.

## 5.5 Experimental Results

We have implemented Algorithms 15 and 18, and we have audited them using GRAAU [60]. The audit consists of comparing the distance computed by an algorithm with the rearrangement distance for every  $\pi \in S_n^\pm$ ,  $1 \leq n \leq 10$ . The results are presented in Tables 5.1 and 5.2, where  $n$  is the size of the permutations, *Avg. Ratio* is the average of the ratios between the distance returned by an algorithm and the rearrangement distance, *Max. Ratio* is the greatest ratio among all the ratios between the distance returned by an algorithm and the rearrangement distance, and *Exact* is the percentage of distances returned by the algorithm that is exactly the rearrangement distance.

Besides providing the *Max. Ratio*, GRAAU also provides up to 50 permutations for which the algorithms achieved this ratio. These permutations can be used to obtain lower bounds on the theoretical approximation ratios of Algorithms 15 and 18. This is precisely what Lemmas 41 and 42 do. Observe that, in the case of Algorithm 18, the lower bound matches the upper bound, so we can conclude that its approximation ratio is tight (Lemma 43).

**Lemma 41.** *The approximation ratio of Algorithm 15 is at least 3.*

*Proof.* Let  $\pi = (+3 +4 -1 -2)$  be a signed permutation. On one hand, we have that Algorithm 15 applies the sequence of signed short reversals  $\rho(2, 4)$ ,  $\rho(1, 3)$ ,  $\rho(1, 1)$ ,  $\rho(2, 2)$ ,  $\rho(3, 3)$ , and  $\rho(4, 4)$  for sorting  $\pi$ . On the other hand, we have that the sequence of signed short reversals  $\rho(1, 3)$  and  $\rho(2, 4)$  sorts  $\pi$ , and the lemma follows.  $\square$

**Lemma 42.** *The approximation ratio of Algorithm 18 is at least 3.*

*Proof.* Let  $\pi = (-3 -2 -5 -4 +1)$  be a signed permutation. On one hand, we have that Algorithm 18 applies the sequence of signed short operations  $\rho(1, 2, 3)$ ,  $\rho(3, 4, 5)$ ,  $\rho(4, 5)$ ,  $\rho(3, 4)$ ,  $\rho(2, 3)$ , and  $\rho(1, 2)$  for sorting  $\pi$ . On the other hand, we have that the sequence of signed short operations  $\rho(3, 5)$  and  $\rho(1, 3)$  sorts  $\pi$ . Therefore the lemma follows.  $\square$

**Lemma 43.** *The approximation ratio of Algorithm 18 is tight.*

*Proof.* Immediate from Theorem 9 and Lemma 42.  $\square$

Table 5.1: Results obtained from the audit of the implementation of Algorithm 15.

<b>n</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Exact</b>
1	1.00	1.00	100.00%
2	1.00	1.00	100.00%
3	1.13	2.50	77.08%
4	1.18	3.00	60.16%
5	1.24	3.00	41.04%
6	1.28	3.00	26.04%
7	1.31	3.00	15.06%
8	1.34	3.00	8.00%
9	1.35	3.00	3.93%
10	1.37	3.00	1.79%

Table 5.2: Results obtained from the audit of the implementation of Algorithm 18.

<b>n</b>	<b>Avg. Ratio</b>	<b>Max. Ratio</b>	<b>Exact</b>
1	1.00	1.00	100.00%
2	1.00	1.00	100.00%
3	1.04	1.50	91.67%
4	1.02	1.50	93.75%
5	1.31	3.00	46.41%
6	1.54	3.00	19.11%
7	1.73	3.00	7.13%
8	1.87	3.00	2.50%
9	1.99	3.00	0.75%
10	2.08	3.00	0.20%

## 5.6 Conclusions

In this article, we have presented optimal algorithms for sorting by signed super short reversals and for sorting by signed super short operations, a 5-approximation algorithm for sorting by signed short reversals, and a 3-approximation algorithm for sorting by signed short operations. We have shown that the expected approximation ratio of the 5-approximation algorithm is not greater than 3 for random signed permutations with more than 12 elements. Moreover, the experimental results on small signed permutations have led us to conclude that the approximation ratio of both approximation algorithms cannot be smaller than 3. In particular, this means that the approximation ratio of the 3-approximation algorithm is tight.

We make two remarks. The first remark is that bounding the length of the operations is not the only approach yielded by the assumption that rearrangement



events affecting large portions of a genome are less likely to occur. Some researchers [13, 106, 118] have proposed to assign weights to the operations according to their length. The second remark is that, as opposed to the unbounded variants of the permutation sorting problem, sorting a linear permutation by short operations is not equivalent to sorting a circular permutation by short operations (see [47] for details). To the best of our knowledge, the only bounded variant considered in the literature that involves circular permutations is the problem of sorting an unsigned circular permutation by reversals of length 2. Jerrum [80] and Egri-Nagy *et al.* [47] demonstrated how to solve this problem in polynomial time.

We see some possible directions for future work. One is to develop polynomial time solutions for the problem of sorting by signed short reversals and for the problem of sorting by signed short operations. Another possibility is to study the problem of sorting signed circular permutations by short operations. In particular, we think that the ideas used to solve the problem of sorting by signed super short reversals can also be used to tackle the problem of sorting a signed circular permutation by reversals of length of at most 2. Finally, one could apply the methods discussed in this work to inferring phylogenies. For instance, Egri-Nagy *et al.* [47] applied their method (*i.e.* sorting unsigned circular permutations by reversals of length 2) to reconstruct the phylogenetic history of some published *Yersinia* genomes. As a result, they produced a phylogenetic tree that is broadly consistent with the phylogenetic tree of Bos *et al.* [17].



# Chapter 6

## Sorting Signed Circular Permutations by Super Short Reversals \*

**Abstract:** We consider the problem of sorting a circular permutation by reversals of length at most 2, a problem that finds application in comparative genomics. Polynomial-time solutions for the unsigned version of this problem are known, but the signed version remained open. In this paper, we present the first polynomial-time solution for the signed version of this problem. Moreover, we perform an experiment for inferring distances and phylogenies for published *Yersinia* genomes and compare the results with the phylogenies presented in previous works.

---

\* *Gustavo Rodrigues Galvão, Christian Baudet, and Zanoni Dias. Sorting signed circular permutations by super short reversals. In Bioinformatics Research and Applications, series Lecture Notes in Bioinformatics, Volume 9096, pp. 272-283, 2015. Copyright 2015 Springer International Publishing Switzerland. DOI: [http://dx.doi.org/10.1007/978-3-319-19048-8\\_23](http://dx.doi.org/10.1007/978-3-319-19048-8_23)*

## 6.1 Introduction

Distance-based methods form one of the three large groups of methods to infer phylogenetic trees from sequence data [96, Chapter 5]. Such methods proceed in two steps. First, the evolutionary distance is computed for every sequence pair and this information is stored in a matrix of pairwise distances. Then, a phylogenetic tree is constructed from this matrix using a specific algorithm, such as *Neighbor-Joining* [108]. Note that, in order to complete the first step, we need some method to estimate the evolutionary distance between a sequence pair. Assuming the sequence data correspond to complete genomes, we can resort to the genome rearrangement approach [54] in order to estimate the evolutionary distance.

In genome rearrangements, one estimates the evolutionary distance between two genomes by finding the rearrangement distance between them, which is the length of the shortest sequence of rearrangement events that transforms one genome into the other. Assuming genomes consist of a single chromosome, share the same set of genes, and contain no duplicated genes, we can represent them as permutations of integers, where each integer corresponds to a gene. If, besides the order, the orientation of the genes is also regarded, then each integer has a sign,  $+$  or  $-$ , and the permutation is called a signed permutation (similarly, we also refer to a permutation as an unsigned permutation when its elements do not have signs). Moreover, if the genomes are circular, then the permutations are also circular; otherwise, they are linear.

A number of publications address the problem of finding the rearrangement distance between two permutations, which can be equivalently stated as a problem of sorting a permutation into the identity permutation (for a detailed survey, the reader is referred to the book of Fertin *et al.* [54]). This problem varies according to the rearrangement events allowed to sort a permutation. Reversals are the most common rearrangement event observed in genomes. They are responsible for reversing the order and orientation of a sequence of genes within a genome. Although the problem of sorting a permutation by reversals is a well-studied problem, most of the works concerning it do not take into account the length of the reversals (*i.e.* the number of genes affected by it). Since it has been observed that short reversals are prevalent in the evolution of some species [32, 33, 94, 110], recent efforts have been made to address this issue [47, 61].

In this paper, we add to those efforts and present a polynomial-time solution for the problem of sorting a signed circular permutation by super short reversals, that is, reversals which affect at most 2 elements (genes) of a permutation (genome). This solution closes a gap in the literature since polynomial-time solutions are known for the problem of sorting an unsigned circular permutation [47, 80], for the problem of sorting an unsigned linear permutation [80], and for the problem of sorting a signed linear permutation [61]. Moreover, we reproduce the experiment performed by Egri-Nagy *et al.* [47] to infer distances and phylogenies for published *Yersinia* genomes, but this time we consider the orientation of the genes (they have ignored it in order

to treat the permutations as unsigned).

The rest of this paper is organized as follows. Section 6.2 succinctly presents the solution developed by Jerrum [80] for the problem of sorting by cyclic super short reversals. Section 6.3 builds upon the previous section and presents the solution for the problem of sorting by signed cyclic super short reversals. Section 6.4 briefly explains how we can use the solutions described in Sect(s). 6.2 and 6.3 to solve the problem of sorting a (signed) circular permutation by super short reversals. Section 6.5 presents experimental results performed on *Yersinia pestis* data. Finally, Sect. 6.6 concludes the paper.

## 6.2 Sorting by Cyclic Super Short Reversals

A *permutation*  $\pi$  is a bijection of  $\{1, 2, \dots, n\}$  onto itself. A classical notation used in combinatorics for denoting a permutation  $\pi$  is the two-row notation

$$\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \dots, n\}$  for  $1 \leq i \leq n$ . This notation indicates that  $\pi(1) = \pi_1, \pi(2) = \pi_2, \dots, \pi(n) = \pi_n$ . The notation used in genome rearrangement literature, which is the one we will adopt, is the one-row notation  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . We say that  $\pi$  has size  $n$ . The set of all permutations of size  $n$  is  $S_n$ .

A *cyclic reversal*  $\rho(i, j)$  is an operation that transforms a permutation  $\pi = (\pi_1 \pi_2 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n)$  into the permutation  $\pi \cdot \rho(i, j) = (\pi_1 \pi_2 \dots \pi_{i-1} \pi_j \pi_{j-1} \dots \pi_{i+1} \pi_i \pi_{j+1} \dots \pi_n)$  if  $1 \leq i < j \leq n$  and transforms a permutation  $\pi = (\pi_1 \pi_2 \dots \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n)$  into the permutation  $\pi \cdot \rho(i, j) = (\pi_j \pi_{j+1} \dots \pi_n \pi_{i+1} \dots \pi_{j-1} \pi_i \pi_{i-1} \dots \pi_1)$  if  $1 \leq j < i \leq n$ . The cyclic reversal  $\rho(i, j)$  is called a *cyclic  $k$ -reversal* if  $k \equiv j - i + 1 \pmod{n}$ . It is called *super short* if  $k = 2$ .

The problem of sorting by cyclic super short reversals consists in finding the minimum number of cyclic super short reversals that transform a permutation  $\pi \in S_n$  into  $\iota_n = (1 \ 2 \ \dots \ n)$ . This number is referred to as the *cyclic super short reversal distance* of permutation  $\pi$  and it is denoted by  $d(\pi)$ .

Let  $S(\pi_i, \pi_j)$  denote the act of switching the positions of the elements  $\pi_i$  and  $\pi_j$  in a permutation  $\pi$ . Note that the cyclic 2-reversal  $\rho(i, j)$  can be alternatively denoted by  $S(\pi_i, \pi_j)$ . Given a sequence  $S$  of cyclic super short reversals and a permutation  $\pi \in S_n$ , let  $R_S(\pi_i)$  be the number of cyclic 2-reversals of the type  $S(\pi_i, \pi_j)$  and let  $L_S(\pi_i)$  be the number of cyclic 2-reversals of the type  $S(\pi_k, \pi_i)$ . In other words,  $R_S(\pi_i)$  denotes the number of times a cyclic 2-reversal moves the element  $\pi_i$  to the right and  $L_S(\pi_i)$  denotes the number of times a cyclic 2-reversal moves the element  $\pi_i$  to the left. We define the *net displacement* of an element  $\pi_i$  with respect to  $S$  as  $d_S(\pi_i) = R_S(\pi_i) - L_S(\pi_i)$ . The *displacement vector* of  $\pi$  with respect to  $S$  is defined as  $d_S(\pi) = (d_S(\pi_1), d_S(\pi_2), \dots, d_S(\pi_n))$ .

**Lemma 44.** *Let  $S = \rho_1, \rho_2, \dots, \rho_t$  be a sequence of cyclic super short reversals that sorts a permutation  $\pi \in S_n$ . Then, we have that*

$$\sum_{i=1}^n d_S(\pi_i) = 0, \tag{6.1}$$

$$\pi_i - d_S(\pi_i) \equiv i \pmod{n}. \tag{6.2}$$

*Proof.* Let  $L_S$  be the number of times a cyclic super short reversal of  $S$  moves an element to the left and let  $R_S$  be the number of times a cyclic super short reversal of  $S$  moves an element to the right. Then,  $L_S = R_S$  because a cyclic super reversal always moves two elements, one for each direction. Therefore, we have that  $\sum_{i=1}^n d_S(\pi_i) = \sum_{i=1}^n (R_S(\pi_i) - L_S(\pi_i)) = R_S - L_S = 0$  and equation 6.1 follows. The equation 6.2 follows from the fact that, once the permutation is sorted, all of its elements must be in the correct position.  $\square$

Note that, in one hand, we can think of a sequence of cyclic super short reversals as specifying a displacement vector. On the other hand, we can also think of a displacement vector as specifying a sequence of cyclic super short reversals. Let  $x = (x_1, x_2, \dots, x_n) \in Z^n$  be a vector and  $\pi \in S_n$  be a permutation. We say that  $x$  is a *valid vector* for  $\pi$  if  $\sum_i x_i = 0$  and  $\pi_i - x_i \equiv i \pmod{n}$ . Given a vector  $x = (x_1, x_2, \dots, x_n) \in Z^n$  and two distinct integers  $i, j \in \{1, 2, \dots, n\}$ , let  $r = i - j$  and  $s = (i + x_i) - (j + x_j)$ . The *crossing number* of  $i$  and  $j$  with respect to  $x$  is defined by

$$c_{ij}(x) = \begin{cases} |\{k \in [r, s] : k \equiv 0 \pmod{n}\}| & \text{if } r \leq s, \\ -|\{k \in [s, r] : k \equiv 0 \pmod{n}\}| & \text{if } r > s. \end{cases}$$

The crossing number of  $x$  is defined by  $C(x) = \frac{1}{2} \sum_{i,j} |c_{ij}(x)|$ . Intuitively, if  $S$  is a sequence of cyclic super short reversals that sorts a permutation  $\pi$  and  $d_S(\pi) = x$ , then  $c_{ij}(x)$  measures the number of times the elements  $\pi_i$  and  $\pi_j$  must “cross”, that is, the number of cyclic 2-reversals of type  $S(\pi_i, \pi_j)$  minus the number of cyclic 2-reversals of type  $S(\pi_j, \pi_i)$ . Using the notion of crossing number, Jerrum [80] was able to prove the following fundamental lemma.

**Lemma 45** (Jerrum [80]). *Let  $S$  be a minimum-length sequence of cyclic super short reversals that sorts a permutation  $\pi \in S_n$  and let  $x \in Z^n$  be a valid vector for  $\pi$ . If  $d_S(\pi) = x$ , then  $d(\pi) = C(x)$ .*

The Lemma 45 allows the problem of sorting a permutation  $\pi$  by cyclic super short reversals to be recast as the optimisation problem of finding a valid vector  $x \in Z^n$  for  $\pi$  with minimum crossing number. More specifically, as Jerrum [80] pointed out, this problem can be formulated as the integer program:

$$\begin{aligned} & \text{Minimize } C(x) \text{ over } Z^n \\ & \text{subject to } \sum_i x_i = 0, \pi_i - x_i \equiv i \pmod{n}. \end{aligned}$$

Although solving an integer program is NP-hard in the general case, Jerrum [80] presented a polynomial-time algorithm for solving this one.

Firstly, Jerrum [80] introduced a transformation  $T_{ij} : Z^n \rightarrow Z^n$  defined as follows. For any vector  $x \in Z^n$ , the result,  $x' = T_{ij}(x)$ , of applying  $T_{ij}$  to  $x$  is given by  $x'_k = x_k$  for  $k \notin \{i, j\}$ ,  $x'_i = x_i - n$ , and  $x'_j = x_j + n$ . Lemma 46 shows what is the effect of this transformation on the crossing number of a vector.

**Lemma 46.** *Let  $x$  and  $x'$  be two vectors over  $Z^n$  such that  $x' = T_{ij}(x)$ . Then,  $C(x') - C(x) = 2(n + x_j - x_i)$ .*

*Proof.* The proof of this lemma is given by Jerrum [80, Theorem 3.9]. We note, however, that he mistakenly wrote that  $C(x') - C(x) = 4(n + x_j - x_i)$ . In other words, he forgot to divide the result by 2. This division is necessary because the crossing number of a vector is the half of the sum of the crossing numbers of its indices.  $\square$

Let  $\max(x)$  and  $\min(x)$  respectively denote the maximum and minimum component values of a vector  $x \in Z^n$ . The transformation  $T_{ij}$  is said to *contract*  $x$  iff  $x_i = \max(x)$ ,  $x_j = \min(x)$  and  $x_i - x_j \geq n$ . Moreover,  $T_{ij}$  is said to *strictly contract*  $x$  iff, in addition, the final inequality is strict. The algorithm proposed by Jerrum [80] starts with a feasible solution to the integer program and performs a sequence of strictly contracting transformations which decrease the value of the crossing number. When no further strictly contracting transformation can be performed, the solution is guaranteed to be optimal. This is because, as showed by Jerrum [80], any two local optimum solutions (*i.e.* solutions which admit no strictly contracting transformation) can be brought into agreement with each other via a sequence of contracting transformations. The detailed algorithm is given below (Algorithm 19).

---

**Algorithm 19:** Algorithm for sorting by cyclic super short reversals.

---

**Data:** A permutation  $\pi \in S_n$ .

**Result:** Number of cyclic super short reversals applied for sorting  $\pi$ .

```

1 Let  $x$  be a  $n$  dimension vector
2 for  $k = 1$  to  $n$  do
3   |  $x_k \leftarrow \pi_k - k$ 
4 end
5 while  $\max(x) - \min(x) > n$  do
6   | Let  $i, j$  be two integers such that  $x_i = \max(x)$  and  $x_j = \min(x)$ 
7   |  $x_i \leftarrow x_i - n$ 
8   |  $x_j \leftarrow x_j + n$ 
9 end
10 return  $C(x)$ 

```

---

Regarding the time complexity of Algorithm 19, we have that line 1 and the for loop of lines 2-4 take  $O(n)$  time. Jerrum [80] observed that none of the variables  $x_i$

changes value more than once, therefore the while loop iterates only  $O(n)$  times. As the lines 6-8 take  $O(n)$  time, the while loop takes  $O(n^2)$  time to execute. Since we can compute the value of  $C(x)$  in  $O(n^2)$  time, the overall complexity of the algorithm is  $O(n^2)$ .

Note that, in this section, we have focused on the problem of computing the cyclic super short reversal distance of a permutation rather than finding the minimum number of cyclic super short reversals that sorts it. As Jerrum [80] remarked, his proofs are constructive and directly imply algorithms for finding the sequence of cyclic super short reversals.

### 6.3 Sorting by Signed Cyclic Super Short Reversals

A *signed permutation*  $\pi$  is a bijection of  $\{-n, \dots, -2, -1, 1, 2, \dots, n\}$  onto itself that satisfies  $\pi(-i) = -\pi(i)$  for all  $i \in \{1, 2, \dots, n\}$ . The two-row notation for a signed permutation is

$$\pi = \begin{pmatrix} -n & \dots & -2 & -1 & 1 & 2 & \dots & n \\ -\pi_n & \dots & -\pi_2 & -\pi_1 & \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \dots, n\}$  for  $1 \leq i \leq n$ . The notation used in genome rearrangement literature, which is the one we will adopt, is the one-row notation  $\pi = (\pi_1 \pi_2 \dots \pi_n)$ . Note that we drop the mapping of the negative elements since  $\pi(-i) = -\pi(i)$  for all  $i \in \{1, 2, \dots, n\}$ . By abuse of notation, we say that  $\pi$  has size  $n$ . The set of all signed permutations of size  $n$  is  $S_n^\pm$ .

A *signed cyclic reversal*  $\rho(i, j)$  is an operation that transforms a signed permutation  $\pi = (\pi_1 \pi_2 \dots \pi_{i-1} \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n)$  into the signed permutation  $\pi \cdot \rho(i, j) = (\pi_1 \pi_2 \dots \pi_{i-1} \pi_j \pi_{j+1} \dots \pi_{i+1} \pi_i \pi_{i-1} \dots \pi_n)$  if  $1 \leq i \leq j \leq n$  and transforms a signed permutation  $\pi = (\pi_1 \pi_2 \dots \pi_i \pi_{i+1} \dots \pi_{j-1} \pi_j \pi_{j+1} \dots \pi_n)$  into the signed permutation  $\pi \cdot \rho(i, j) = (\pi_1 \pi_2 \dots \pi_{j+1} \pi_j \pi_{j-1} \dots \pi_i \pi_{i+1} \dots \pi_n)$  if  $1 \leq j < i \leq n$ . The signed cyclic reversal  $\rho(i, j)$  is called a *signed cyclic  $k$ -reversal* if  $k \equiv j - i + 1 \pmod{n}$ . It is called *super short* if  $k \leq 2$ .

The problem of sorting by signed cyclic super short reversals consists in finding the minimum number of signed cyclic super short reversals that transform a permutation  $\pi \in S_n^\pm$  into  $\iota_n$ . This number is referred to as the *signed cyclic super short reversal distance* of permutation  $\pi$  and it is denoted by  $d^\pm(\pi)$ .

Let  $S(|\pi_i|, |\pi_j|)$  denote the act of switching the positions and flipping the signs of the elements  $\pi_i$  and  $\pi_j$  in a signed permutation  $\pi$ . Note that the signed cyclic 2-reversal  $\rho(i, j)$  can be alternatively denoted by  $S(|\pi_i|, |\pi_j|)$ . Given a sequence  $S$  of cyclic signed super short reversals and a signed permutation  $\pi \in S_n^\pm$ , let  $R_S(\pi_i)$  be the number of signed cyclic 2-reversals of the type  $S(|\pi_i|, |\pi_j|)$  and let  $L_S(\pi_i)$  be the number of signed cyclic 2-reversals of the type  $S(|\pi_k|, |\pi_i|)$ . We define the *net*



*displacement* of an element  $\pi_i$  with respect to  $S$  as  $d_S(\pi_i) = R_S(\pi_i) - L_S(\pi_i)$ . The *displacement vector* of  $\pi$  with respect to  $S$  is defined as  $d_S(\pi) = (d_S(\pi_1), d_S(\pi_2), \dots, d_S(\pi_n))$ . The following lemma is the signed analog of Lemma 44. We omit the proof because it is the same as of the proof of Lemma 44.

**Lemma 47.** *Let  $S = \rho_1, \rho_2, \dots, \rho_t$  be a sequence of signed cyclic super short reversals that sorts a signed permutation  $\pi \in S_n^\pm$ . Then, we have that*

$$\sum_{i=1}^n d_S(\pi_i) = 0, \quad (6.3)$$

$$|\pi_i| - d_S(\pi_i) \equiv i \pmod{n}. \quad (6.4)$$

Let  $x \in Z^n$  be a vector and  $\pi \in S_n^\pm$  be a signed permutation. We say that  $x$  is a *valid vector* for  $\pi$  if  $\sum_i x_i = 0$  and  $|\pi_i| - x_i \equiv i \pmod{n}$ . Given a valid vector  $x$  for the signed permutation  $\pi$ , we define the set  $\text{podd}(\pi, x)$  as  $\text{podd}(\pi, x) = \{i : \pi_i > 0 \text{ and } |x_i| \text{ is odd}\}$  and we define the set  $\text{neven}(\pi, x)$  as  $\text{neven}(\pi, x) = \{i : \pi_i < 0 \text{ and } |x_i| \text{ is even}\}$ . Moreover, let  $U(\pi, x)$  denote the union of these sets, that is,  $U(\pi, x) = \text{podd}(\pi, x) \cup \text{neven}(\pi, x)$ . The following lemma is the signed analog of Lemma 45.

**Lemma 48.** *Let  $S$  be a minimum-length sequence of signed cyclic super short reversals that sorts a signed permutation  $\pi \in S_n^\pm$  and let  $x \in Z^n$  be a valid vector for  $\pi$ . If  $d_S(\pi) = x$ , then  $d^\pm(\pi) = C(x) + |U(\pi, x)|$ .*

*Proof.* Note that the sequence  $S$  can be decomposed into two distinct subsequences  $S_1$  and  $S_2$  such that  $S_1$  is formed by the signed cyclic 1-reversals of  $S$  and  $S_2$  is formed by the signed cyclic 2-reversals of  $S$ . Moreover, we can assume without loss of generality that the signed cyclic reversals of subsequence  $S_2$  are applied first. We argue that  $|S_1| = |U(\pi, x)|$  regardless the size of  $S_2$ . To see this, suppose that we apply a signed cyclic 2-reversal  $\rho(i, j)$  of  $S_2$  in  $\pi$ , obtaining a signed permutation  $\pi'$ . Moreover, let  $S'$  be the resulting sequence after we remove  $\rho(i, j)$  from  $S$ . We have that  $d_{S'}(\pi'_k) = d_S(\pi_k)$  for  $k \notin \{i, j\}$ ,  $d_{S'}(\pi'_i) = d_S(\pi_i) - 1$ , and  $d_{S'}(\pi'_j) = d_S(\pi_j) + 1$ . Then, assuming the vector  $x' \in Z^n$  is equal to  $d_{S'}(\pi')$ , we can conclude that  $U(\pi', x') = U(\pi, x)$  because  $\rho(i, j)$  has changed both the parities of  $|x_i|$  and  $|x_j|$  and the signs of  $\pi_i$  and  $\pi_j$ . Since  $|S_1| = |U(\pi, x)|$  regardless the size of  $S_2$  and we know from Lemma 45 that  $|S_2| \geq C(x)$ , we can conclude that  $|S_2| = C(x)$ , therefore the lemma follows.  $\square$

The Lemma 48 allows the problem of sorting a signed permutation  $\pi$  by signed cyclic super short reversals to be recast as the optimisation problem of finding a valid vector  $x \in Z^n$  for  $\pi$  which minimizes the sum  $C(x) + |U(\pi, x)|$ . The next theorem shows how to solve this problem in polynomial time.

**Theorem 10.** *Let  $\pi \in S_n^\pm$  be a signed permutation. Then, we can find a valid vector  $x \in Z^n$  which minimizes the sum  $C(x) + |U(\pi, x)|$  in polynomial time.*

*Proof.* We divide our analysis into two cases:

- i)  $n$  is even. In this case, we have that the value of  $|U(\pi, x)|$  is the same for any feasible solution  $x$ . This is because, in order to be a feasible solution, a vector  $x$  has to satisfy the restriction  $|\pi_i| - x_i \equiv i \pmod{n}$ . This means that  $x_i$  is congruent modulo  $n$  with  $a = |\pi_i| - i$  and belongs to the equivalent class  $\{\dots, a - 2n, a - n, a, a + n, a + 2n, \dots\}$ . Since  $n$  is even, the parities of the absolute values of the elements in this equivalence class are the same, therefore the value of  $|U(\pi, x)|$  is the same for any feasible solution  $x$ . It follows that we can only minimize the value of  $C(x)$  and this can be done by performing successive strictly contracting transformations.
- ii)  $n$  is odd. In this case, it is possible to minimize the values of  $|U(\pi, x)|$  and  $C(x)$ . Firstly, we argue that minimizing  $C(x)$  leads to a feasible solution  $x''$  such that  $C(x'') + |U(\pi, x'')$  is at least as low as  $C(x') + |U(\pi, x')|$ , where  $x'$  can be any feasible solution such that  $C(x')$  is not minimum. To see this, let  $x'$  be a feasible solution such that  $C(x')$  is not minimum. Then, we can perform a sequence of strictly contracting transformations which decrease the value of  $C(x)$ . When no further strictly contracting transformation can be performed, we obtain a solution  $x''$  such that  $C(x'')$  is minimum. On one hand, we know from Lemma 46 that each strictly contracting transformation  $T_{ij}$  decreases  $C(x)$  by at least 2 units. On the other hand, since  $n$  is odd, its possible that the parities of  $|x_i|$  and  $|x_j|$  have been changed in such a way that the value of  $|U(\pi, x)|$  increases by 2 units. Therefore, in the worst case, each strictly contracting transformation does not change the value of  $C(x) + |U(\pi, x)|$ , so  $C(x') + |U(\pi, x')| \geq C(x'') + |U(\pi, x'')$ . Now, we argue that, if there exists more than one feasible solution  $x$  such that  $C(x)$  is minimum, then it is still may be possible to minimize the value of  $|U(\pi, x)|$ .

Jerrum [80, Theorem 3.9] proved that if there is more than one feasible solution such that  $C(x)$  is minimum, then each of these solutions can be brought into agreement with each other via a sequence of contracting transformations. Note that a contracting transformation  $T_{ij}$  does not change the value of  $C(x)$ , but it can change the value of  $|U(\pi, x)|$  because  $n$  is odd and the parities of  $|x_i|$  and  $|x_j|$  change when  $T_{ij}$  is performed. This means that, among all feasible solutions such that  $C(x)$  is minimum, some of them have minimum  $|U(\pi, x)|$  and these solutions are optimal. Therefore, we can obtain an optimal solution by first obtaining a feasible solution with minimum  $C(x)$  (this can be done by performing successive strictly contracting transformations) and then we can apply on it every possible contracting transformation  $T_{ij}$  which decreases the value of  $|U(\pi, x)|$ .

□

The proof of Theorem 10 directly implies an exact algorithm for sorting by signed cyclic super short reversals. Such an algorithm is described below (Algorithm 20). Regarding its time complexity, we know from previous section that lines 1-9 take  $O(n^2)$  time. Since lines 13-23 take  $O(1)$  time, we can conclude that the nested for loops take  $O(n^2)$  times to execute. Finally, we can compute  $C(x) + |U(\pi, x)|$  in  $O(n^2)$ , therefore the overall complexity of Algorithm 20 is  $O(n^2)$ .

---

**Algorithm 20:** Algorithm for sorting by signed cyclic super short reversals.

---

**Data:** A permutation  $\pi \in S_n^\pm$ .

**Result:** Number of signed cyclic super short reversals applied for sorting  $\pi$ .

```

1 Let  $x$  be a  $n$  dimension vector
2 for  $k = 1$  to  $n$  do
3    $x_k \leftarrow |\pi_k| - k$ 
4 end
5 while  $\max(x) - \min(x) > n$  do
6   Let  $i, j$  be two integers such that  $x_i = \max(x)$  and  $x_j = \min(x)$ 
7    $x_i \leftarrow x_i - n$ 
8    $x_j \leftarrow x_j + n$ 
9 end
10 if  $n$  is odd then
11   for  $i = 1$  to  $n - 1$  do
12     for  $j = i + 1$  to  $n$  do
13       if  $x_i > x_j$  then
14          $min \leftarrow j$ 
15          $max \leftarrow i$ 
16       else
17          $min \leftarrow i$ 
18          $max \leftarrow j$ 
19       end
20       if  $x_{max} - x_{min} = n$  and  $min \in U(\pi, x)$  and  $max \in U(\pi, x)$  then
21          $x_{max} \leftarrow x_i - n$ 
22          $x_{min} \leftarrow x_j + n$ 
23       end
24     end
25   end
26 end
27 return  $C(x) + |U(\pi, x)|$ 

```

---

Note that, in this section, we have focused on the problem of computing the signed cyclic super short reversal distance of a signed permutation rather than finding the minimum number of signed cyclic super short reversals that sorts it. We remark that the proofs are constructive and directly imply algorithms for finding the sequence of signed cyclic super short reversals.

## 6.4 Sorting Circular Permutations

In this section, we briefly explain how we can use the solution for the problem of sorting by (signed) cyclic super short reversals to solve the problem of sorting a (signed) circular permutation by super short reversals. This explanation is based on Sect. 2.3 of the work of Egri-Nagy *et al.* [47] and on Sect. 2.5 of the book of Fertin *et al.* [54], where one can find more details.

Note that a circular permutation can be “unrolled” to produce a linear permutation, such as defined in the two previous sections. This process can produce  $n$  different linear permutations, one for each possible rotation of the circular permutation. Moreover, since a circular permutation represents a circular chromosome, which lives in three dimension, it can also be “turned over” before being unrolled. This means that, for each possible rotation of the circular permutation, we can first turn it over and then unroll it, producing a linear permutation. Again, this process can produce  $n$  different linear permutations. The  $n$  linear permutations produced in the first process are different from the  $n$  linear permutations produced in the second process, thus both processes can produce a total of  $2n$  different linear permutations. Each of these  $2n$  linear permutations represents a different viewpoint from which to observe the circular permutation, therefore they are all equivalent.

The discussion of the previous paragraph leads us to conclude that, in order to sort a (signed) circular permutation by super short reversals, we can sort each of the  $2n$  equivalent (signed) linear permutations by (signed) cyclic super short reversals, generating  $2n$  different sorting sequences. Then, we can take the sequence of minimum length as the sorting sequence for the (signed) circular permutation and the *super short reversal distance* of the (signed) circular permutation is the length of this sequence. Note that this procedure takes  $O(n^3)$  time because we have to execute Algorithm 19 or Algorithm 20  $O(n)$  times.

## 6.5 Experimental Results and Discussion

We implemented the procedure described in the previous section for computing the super short reversal distance of a signed circular permutation and we reproduced the experiment performed by Egri-Nagy *et al.* [47] for inferring distances and phylogenies for published *Yersinia* genomes. In fact, we performed the same experiment, except that we considered the orientation of the genes rather than ignoring it and we considered that each permutation has 78 elements rather than 79<sup>2</sup>. More specifically, we obtained from Darling *et al.* [33] the signed circular permutations which represent

---

<sup>2</sup>In their article, Darling *et al.* [33] state that they could identify 78 conserved segments (or blocks) using Mauve, but they provided permutations with elements ranging from 0 to 78. In a personal communication, Darling confirmed that there are actually 78 blocks, with 0 and 78 being part of the same block. Nevertheless, we performed another experiment, this time considering the permutations have 79 elements. Although the distances were greater, the topology of the tree was the same.

Table 6.1: Matrix of the super short reversal distances among the signed circular permutations which represent the *Yersinia* genomes. The names of the species were abbreviated so that YPK refers to *Y. pestis Kim*, YPA to *Y. pestis Antiqua*, YPM to *Y. pestis Microtus 91001*, YPC to *Y. pestis CO92*, YPN to *Y. pestis Nepal516*, YPP to *Y. pestis Pestoides F 15-70*, YT1 to *Y. pseudotuberculosis IP31758*, and YT2 to *Y. pseudotuberculosis IP32953*.

	YPK	YPA	YPM	YPC	YPN	YPP	YT1	YT2
YPK	0	243	752	205	338	533	764	760
YPA	243	0	772	352	279	510	724	773
YPM	752	772	0	728	747	643	361	385
YPC	205	352	728	0	381	656	776	760
YPN	338	279	747	381	0	547	617	624
YPP	533	510	643	656	547	0	434	457
YT1	764	724	361	776	617	434	0	189
YT2	760	773	385	760	624	457	189	0

eight *Yersinia* genomes. Then, we computed the super short reversal distance between every pair of signed circular permutation and this information was stored in a matrix of pairwise distances (Table 6.1). Finally, a phylogenetic tree was constructed from this matrix using *Neighbor-Joining* [108] method. The resulting phylogeny is shown in Fig. 6.1.

Considering the pair of *Y. pseudotuberculosis* as outgroup, the obtained phylogeny shows that *Y. pestis Microtus 91001* was the first to diverge. It was followed then by the divergences of *Y. pestis Pestoides F 15-70*, *Y. pestis Nepal516*, *Y. pestis Antiqua* and the final divergence of *Y. pestis Kim* and *Y. pestis CO92*. This result is different of the one obtained by Egri-Nagy *et. al.* [47] which used super short reversal distance between unsigned permutations. On their results, the divergence of *Y. pestis Nepal516* happened before the divergence of *Y. pestis CO92* which occurred previous to the divergence of *Y. pestis Kim* and *Y. pestis Antiqua*.

In our work and in the work of Egri-Nagy *et. al.* [47], the use of super short reversals resulted on topologies which are different from the one of Darling *et al.* [33], which considered inversions of any size. The first difference observed on the result of Darling *et al.* [33] is that *Y. pestis Pestoides F 15-70* diverged before *Y. pestis Microtus 91001*. The second difference shows that *Y. pestis Nepal516* is sibling of *Y. pestis Kim*, that *Y. pestis CO92* is sibling of *Y. pestis Antiqua* and that these four bacteria have a common ancestor that is descendant of *Y. pestis Microtus 91001*.

If we look to the branch lengths of the two trees obtained with super short reversal distances and we compare with the branch lengths of the topology obtained by Darling *et al.* [33], we can see that our results are more consistent than the one obtained by Egri-Nagy *et al.* [47]. For instance, on our results the distance between the two *Y.*

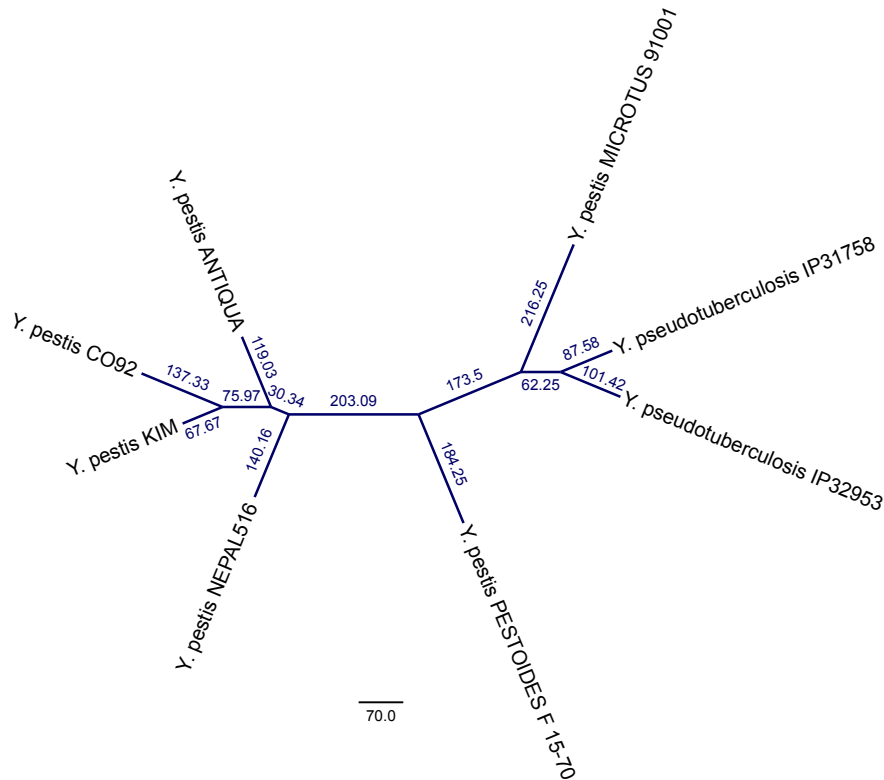


Figure 6.1: Phylogeny of the *Yersinia* genomes based on the super short reversal distance of the signed circular permutations.

*pseudotuberculosis* is smaller than the one observed between the pair *Y. pestis* *Kim* and *Y. pestis* *Antiqua*, what agrees with the configuration obtained by Darling *et al.* [33].

## 6.6 Conclusions

In this paper, we presented a polynomial-time solution for the problem of sorting a signed circular permutation by super short reversals. From a theoretical perspective, this solution is important because it closes a gap in the literature. From a biological perspective, it is important because signed permutations constitute a more adequate model for genomes. Moreover, we performed an experiment to infer distances and phylogenies for published *Yersinia* genomes and compared the results with the phylogenies presented in previous works [33, 47]. Our obtained topology is similar to the one obtained by Egri-Nagy *et al.* [47]. However, the distances calculated with our algorithm are more consistent with the topology obtained by Darling *et al.* [33]. Some

theoretical questions remain open (for instance, the diameter of the super short reversal distance for signed permutations), and we intend to address them in our future research.





# Chapter 7

## Concluding Remarks

We conclude this thesis with a summary of the previous chapters (except the first) along with a discussion of a few possible directions for further research.

In Chapter 2, we presented GRAAu, a tool for evaluating approximation algorithms and heuristics for permutation sorting problems. To build this tool, we computed the rearrangement distances of all permutations in  $S_n$ ,  $1 \leq n \leq 13$ , and in  $S_n^\pm$ ,  $1 \leq n \leq 10$ , with respect to a number of rearrangement models regarded in the literature that take into account reversals or transpositions. We did not compute the rearrangement distances for permutations with more elements due to resource constraints. Therefore, a possible direction for future research is to develop more efficient algorithms for computing the rearrangement distance of every permutation in the symmetric group. Gonçalves, Bueno, and Hausen [65] have managed to do this by restricting the rearrangement model to transpositions only. Such restriction allowed them to use toric equivalences to reduce the search space.

The rearrangement distances were stored in the database, referred to as Rearrangement Distance Database, which can be accessed via the internet. Hence, anyone can access it and extract information of interest regarding the rearrangement distances. For instance, by analyzing the distribution of the rearrangement distances, we were able to validate some conjectures about the diameter of the symmetric group. Other researchers [12, 67, 97] have also used the information available in the Rearrangement Distance Database.

Finally, to illustrate the application of GRAAu, four approximation algorithms were audited. Based on the data provided by GRAAu, specifically the Maximum Ratio and the permutations that exhibited it, we were able to show that the approximation ratios of three algorithms are tight. The idea of using the data provided by GRAAu to get insight on the approximation ratio of an algorithm was also used in other chapters of this thesis and may as well be used in future works.

In Chapter 3, we presented a general heuristic for permutation sorting problems. The heuristic works by iteratively improving an initial solution produced by other algorithm. In each step, it makes a local change within a sliding window, which

moves across the solution. The main idea employed by the heuristic is to transform the sliding window into a small instance of the permutation sorting problem in such a way that an optimal solution for that instance can be retrieved from the Rearrangement Distance Database. To evaluate the heuristic, we applied it to the solutions provided by 23 approximation algorithms. The performance of the heuristic varied considerably depending on the algorithm that produced the initial solutions: it ranged from almost 100% of improvement to below 5%. The observed variation is mainly due to the quality of the initial solutions: the closer they are to the optimal solution, the more difficult it is to improve them.

A drawback of retrieving an optimal solution from a database is that it does not scale up to larger permutations. An alternative approach is to compute an optimal solution using an exact exponential time algorithm. In this way, the size of the sliding window can be adjusted according to the efficiency of the exact algorithm at hand. Recently, Lancia, Rinaldi, and Serafini [92] proposed a general Integer Programming (IP) model that can be used to solve several variants of the permutation sorting problem. We believe that the combination of our heuristic with their IP model can lead to interesting results.

In Chapter 4, we presented experimental and theoretical results regarding three algorithms for the problem of sorting by transpositions, namely Walter, Dias, and Meidanis' 2.25-approximation algorithm [124], Benoît-Gagné and Hamel's 3-approximation algorithm [14], and Guyer, Heath, and Vergara's heuristic [69]. These algorithms are based on alternative approaches to the cycle graph, which is the tool used by the best known approximation algorithms for the problem of sorting by transpositions [38, 48]. Chapter 4 delivers a good picture of this group of algorithms and lays the groundwork for future research. In particular, it is still not clear whether these alternative approaches can or cannot yield algorithms with low approximation ratios. Therefore, searching for results that could help make progress on this question either way is an interesting direction to follow.

In chapters 5 and 6, we addressed the problem of sorting signed permutations by bounded operations. Specifically, we studied (i) the problem of sorting a signed linear permutation by super short reversals, (ii) the problem of sorting a signed circular permutation by super short reversals, (iii) the problem of sorting a signed linear permutation by short reversals, (iv) the problem of sorting a signed linear permutation by super short operations, and (v) the problem of sorting a signed linear permutation by short operations. We presented polynomial-time algorithms for problems (i), (ii) and (iv), and we presented polynomial-time approximation algorithms for problems (iii) and (v). Possible directions for future research include developing better algorithms for problems (iii) and (v) as well as considering other rearrangement operations, such as short transpositions and short reversals, for sorting a signed circular permutation.

Finally, we used the algorithm developed for problem (ii) to perform an experiment for inferring distances and phylogenies for published *Yersinia* genomes. In fact,

we reproduced the experiment performed by Egri-Nagy *et al.* [47], but this time we considered the orientation of the genes (they have ignored it in order to treat the permutations as unsigned). Our obtained topology is similar to the one obtained by them, but the distances calculated with our algorithm are more consistent with the phylogeny obtained by Darling *et al.* [33]. Future work should include more experiments on real data, possibly with genomes of not so closely related species.



# Bibliography

- [1] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, 1989.
- [2] M. Alekseyev. *Duplications and Genome Rearrangements*. PhD thesis, University of California, San Diego, 2007.
- [3] M. A. Alekseyev and P. A. Pevzner. Multi-break rearrangements and chromosomal evolution. *Theoretical Computer Science*, 395(2-3):193–202, 2008.
- [4] J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, New York, USA, 2010.
- [5] F. J. Ayala and W. M. Fitch. Genetics and the origin of species: An introduction. *Proceedings of the National Academy of Sciences of the United States of America*, 94(15):7691–7697, 1997.
- [6] L. Babai and Á. Seress. On the diameter of Cayley graphs of the symmetric group. *Journal of Combinatorial Theory, Series A*, 49(1):175 – 179, 1988.
- [7] L. Babai and Á. Seress. On the diameter of permutation groups. *European Journal of Combinatorics*, 13(4):231–243, 1992.
- [8] D. Bader, B. 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):483–491, 2001.
- [9] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.
- [10] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.
- [11] C. Baudet. *Enumeração de Traças e Identificação de Breakpoints: Estudos de Aspectos da Evolução*. PhD thesis, University of Campinas, 2010. In Portuguese.

- [12] T. R. Benavidez. *Polynomial Formulae for the  $k$ -Slice of the Symmetric Group under Various Genome Rearrangement Models*. Honors math thesis, University of Oklahoma, 2014.
- [13] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting by length-weighted reversals. *Journal of Computer and System Sciences*, 74(5):744–774, 2008.
- [14] M. Benoît-Gagné and S. Hamel. A new and faster method of sorting by transpositions. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM'2007)*, volume 4580 of *Lecture Notes in Computer Science*, pages 131–141, London, Ontario, Canada, 2007. Springer-Verlag.
- [15] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'2002)*, volume 2461 of *Lecture Notes in Computer Science*, pages 200–210, Rome, Italy, 2002. Springer-Verlag.
- [16] M. Bernt. *Gene order rearrangement methods for the reconstruction of phylogeny*. PhD thesis, Universität Leipzig, 2010.
- [17] K. I. Bos, V. J. Schuenemann, G. B. Golding, H. A. Burbano, N. Waglechner, B. K. Coombes, J. B. McPhee, S. N. DeWitte, M. Meyer, S. Schmedes, J. Wood, D. J. Earn, D. A. Herring, P. Bauer, H. N. Poinar, and J. Krause. A draft genome of *Yersinia pestis* from victims of the black death. *Nature*, 478(7370):506–510, 2011.
- [18] G. Bourque. *Algorithms for phylogenetic tree reconstruction based on genome rearrangements*. PhD thesis, University of Southern California, 2002.
- [19] M. D. V. Braga. *Exploring the solution space of sorting by reversals when analyzing genome rearrangements*. PhD thesis, Université Lyon, 2009.
- [20] L. Bulteau. *Algorithmic Aspects of Genome Rearrangements*. PhD thesis, Université de Nantes, 2013.
- [21] L. Bulteau, G. Fertin, and I. Rusu. Pancake flipping is hard. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS'2012)*, volume 7464 of *Lecture Notes in Computer Science*, pages 247–258, Bratislava, Slovakia, 2012. Springer-Verlag.
- [22] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012.
- [23] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

- [24] A. Cayley. Note on the theory of permutations. *Philosophical Magazine*, 34:527–529, 1849.
- [25] A. Cayley. Desiderata and suggestions: No. 2. The theory of groups: Graphical representation. *American Journal of Mathematics*, 1(2):174–176, 1878.
- [26] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21th ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 161–173, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.
- [27] X. Chen. On sorting unsigned permutations by double-cut-and-joins. *Journal of Combinatorial Optimization*, 25:339–351, 2013.
- [28] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. Shields, I. H. Sudborough, and W. Voit. An  $(18/11)n$  upper bound for sorting by prefix reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009.
- [29] B. Chitturi and I. H. Sudborough. Bounding prefix transposition distance for strings and permutations. *Theoretical Computer Science*, 421:15–24, 2012.
- [30] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1998.
- [31] D. S. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.
- [32] D. A. Dalevi, N. Eriksen, K. Eriksson, and S. G. E. Andersson. Measuring genome divergence in bacteria: A case study using chlamydian data. *Journal of Molecular Evolution*, 55(1):24–36, 2002.
- [33] A. E. Darling, I. Miklós, and M. A. Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS Genetics*, 4(7):e1000128, 2008.
- [34] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, England, 1859.
- [35] U. Dias. *Problemas de Comparação de Genomas*. PhD thesis, University of Campina, 2012. In Portuguese.
- [36] U. Dias and Z. Dias. Extending Bafna-Pevzner algorithm. In *Proceedings of the ACM International Symposium on Biocomputing (ISB'2010)*, pages 23:1–23:8, Calicut, Kerala, India, 2010. ACM Press.

- [37] U. Dias and Z. Dias. An improved 1.375-approximation algorithm for the transposition distance problem. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology (BCB'2010)*, pages 334–337, Niagara Falls, New York, 2010. ACM Press.
- [38] U. Dias and Z. Dias. Heuristics for the transposition distance problem. *Journal of Bioinformatics and Computational Biology*, 11:1350013, 2013.
- [39] U. Dias, G. R. Galvão, C. N. Lintzmayer, and Z. Dias. A general heuristic for genome rearrangement problems. *Journal of Bioinformatics and Computational Biology*, 12(3):1450012, 2014.
- [40] Z. Dias. *Rearranjo de Genomas: uma Coletânea de Artigos*. PhD thesis, University of Campinas, 2002.
- [41] Z. Dias and U. Dias. Sorting by prefix reversals and prefix transpositions. *Discrete Applied Mathematics*, 181:78–89, 2015.
- [42] Z. Dias and J. Meidanis. Sorting by prefix transpositions. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, volume 2476 of *Lecture Notes in Computer Science*, pages 65–76, Lisbon, Portugal, 2002. Springer-Verlag.
- [43] T. Dobzhansky. *Genetics and the Origin of Species*. Columbia University Press, New York, USA, 1937.
- [44] T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *Drosophila pseudoobscura*. *Genetics*, 23(1):28–64, 1938.
- [45] J. R. Driscoll and M. L. Furst. Computing short generator sequences. *Information and Computation*, 72(2):117–132, 1987.
- [46] H. Dweighter. Elementary problems and solutions, problem e2569. *American Mathematical Monthly*, 82:1010, 1975.
- [47] A. Egri-Nagy, V. Gebhardt, M. M. Tanaka, and A. R. Francis. Group-theoretic models of the inversion process in bacterial genomes. *Journal of Mathematical Biology*, 69(1):243–265, 2014.
- [48] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.
- [49] N. Eriksen. *Combinatorial methods in comparative genomics*. PhD thesis, Royal Institute of Technology, 2003.



- [50] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1-3):289–300, 2001.
- [51] S. Even and O. Goldreich. The minimum-length generator sequence problem is NP-hard. *Journal of Algorithms*, 2(3):311–313, 1981.
- [52] P. C. Feijão. *On genome rearrangement models*. PhD thesis, University of Campinas, 2012.
- [53] P. C. Feijão and J. Meidanis. SCJ: A breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1318–1329, 2011.
- [54] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. The MIT Press, Cambridge, MA, USA, 2009.
- [55] J. Fischer and S. W. Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 415–425, Mallorca, Spain, 2005. Springer-Verlag.
- [56] G. R. Galvão. *Uma Ferramenta de Auditoria para Algoritmos de Rearranjo de Genomas*. Master's thesis, University of Campinas, 2012. In Portuguese.
- [57] G. R. Galvão, C. Baudet, and Z. Dias. Sorting signed circular permutations by super short reversals. In *Proceedings of the 11th International Symposium on Bioinformatics Research and Applications (ISBRA'2015)*, volume 9096 of *Lecture Notes in Computer Science*, pages 272–283. Springer International Publishing, 2015.
- [58] G. R. Galvão and Z. Dias. On the performance of sorting permutations by prefix operations. In *Proceedings of the 4th International Conference on Bioinformatics and Computational Biology (BICoB'2012)*, pages 102–107, Las Vegas, Nevada, USA, 2012. Curran Associates, Inc.
- [59] G. R. Galvão and Z. Dias. Approximation algorithms for sorting by signed short reversals. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB'2014)*, pages 360–369, Newport Beach, California, USA, 2014. ACM Press.
- [60] G. R. Galvão and Z. Dias. An audit tool for genome rearrangement algorithms. *ACM Journal of Experimental Algorithmics*, 19:1.1–1.34, 2014.
- [61] G. R. Galvão, O. Lee, and Z. Dias. Sorting signed permutations by short operations. *Algorithms for Molecular Biology*, 10(12), 2015.

- [62] G. R. Galvão and Z. Dias. On alternative approaches for approximating the transposition distance. *Journal of Universal Computer Science*, 20(9):1259–1283, 2014.
- [63] O. Gascuel. *Mathematics of Evolution and Phylogeny*. Oxford University Press, Inc., New York, NY, USA, 2005.
- [64] W. Gates and C. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27:47–57, 1979.
- [65] J. Gonçalves, L. R. Bueno, and R. A. Hausen. Assembling a new and improved transposition distance database. In *Anais do XLV Simpósio Brasileiro de Pesquisa Operacional (SBPO'2013)*, pages 2355–2365, Natal, Brazil, 2013.
- [66] A. J. F. Griffiths, W. M. Gelbart, J. H. Miller, and R. C. Lewontin. *Modern Genetic Analysis*. W. H. Freeman, New York, USA, 1999. Chromosomal Rearrangements. Available from: <http://www.ncbi.nlm.nih.gov/books/NBK21367/>.
- [67] S. Grusea and A. Labarre. The distribution of cycles in breakpoint graphs of signed permutations. *Discrete Applied Mathematics*, 161(10-11):1448–1466, 2013.
- [68] Q. Gu, S. Peng, and I. H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.
- [69] S. A. Guyer, L. S. Heath, and J. P. C. Vergara. Subsequence and run heuristics for sorting by transpositions. Technical Report TR-97-20, Virginia Polytechnic Institute & State University, 1997.
- [70] S. Hannenhalli. *Computational theory of genome evolution via rearrangements*. PhD thesis, Pennsylvania State University, 1996.
- [71] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'1995)*, pages 581–592, Washington, DC, USA, 1995. IEEE Computer Society.
- [72] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.
- [73] T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204(2):275–290, 2006.

- [74] T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005.
- [75] R. A. Hausen. *Rearranjos de Genomas: Teoria e Aplicações*. PhD thesis, Federal University of Rio de Janeiro, 2007. In Portuguese.
- [76] L. S. Heath and J. P. C. Vergara. Sorting by bounded block-moves. *Discrete Applied Mathematics*, 88:181–206, 1998.
- [77] L. S. Heath and J. P. C. Vergara. Sorting by short blockmoves. *Algorithmica*, 28(3):323–354, 2000.
- [78] L. S. Heath and J. P. C. Vergara. Sorting by short swaps. *Journal of Computational Biology*, 10(5):775–789, 2003.
- [79] M. H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25(1):67–94, 1997.
- [80] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.
- [81] H. Jiang, H. Feng, and D. Zhu. An  $5/4$ -approximation algorithm for sorting permutations by short block moves. In *Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC'2014)*, pages 491–503, Jeonju, Korea, 2014. Springer International Publishing.
- [82] H. Jiang, D. Zhu, and B. Zhu. A  $(1+\epsilon)$ -approximation algorithm for sorting by short block-moves. *Theoretical Computer Science*, 439:1–8, 2012.
- [83] J. Kanai, S. V. Rice, T. A. Nartker, and G. Nagy. Automated evaluation of OCR zoning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):86–90, 1995.
- [84] J. D. Kececioğlu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1-2):80–110, 1995.
- [85] B. M. Kiernan. The development of Galois theory from Lagrange to Artin. *Archive for History of Exact Sciences*, 8(1-2):40–154, 1971.
- [86] Y. Koh and S. Ree. Connected permutation graphs. *Discrete Mathematics*, 307(21):2628–2635, 2007.
- [87] E. Konstantinova. Some problems on Cayley graphs. *Linear Algebra and its Applications*, 429(11-12):2754–2769, 2008.

- [88] J. Kováč. *Algorithms for Genome Rearrangements*. PhD thesis, Comenius University, 2013.
- [89] A. Labarre. *Combinatorial aspects of genome rearrangements and haplotype networks*. PhD thesis, Université Libre de Bruxelles, 2008.
- [90] A. Labarre. Lower bounding edit distances between permutations. *SIAM Journal on Discrete Mathematics*, 27:1410–1428, 2013.
- [91] S. Lakshmivarahan, J.-S. Jwo, and S. K. Dhall. Symmetry in interconnection networks based on Cayley graphs of permutation groups: A survey. *Parallel Computing*, 19(4):361–407, 1993.
- [92] G. Lancia, F. Rinaldi, and P. Serafini. A unified integer programming model for genome rearrangement problems. In *Proceedings of the 3rd International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO'2015)*, volume 9043 of *Lecture Notes in Computer Science*, pages 491–502, Granada, Spain, 2015. Springer International Publishing.
- [93] S. Latifi. How can permutations be used in the evaluation of zoning algorithms? *International Journal of Pattern Recognition and Artificial Intelligence*, 10(3):223–237, 1996.
- [94] J. F. Lefebvre, N. El-Mabrouk, E. Tillier, and D. Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, 19(suppl 1):i190–i196, 2003.
- [95] D. H. Lehmer. Teaching combinatorial tricks to a computer. *Proceedings of Symposia in Applied Mathematics*, 10:179–193, 1960.
- [96] P. Lemey, M. Salemi, and A. Vandamme. *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge University Press, Cambridge, UK, 2009.
- [97] C. N. Lintzmayer and Z. Dias. On the diameter of rearrangement problems. In *Proceedings of the First International Conference on Algorithms for Computational Biology (AlCoB'2014)*, volume 8542 of *Lecture Notes in Computer Science*, pages 158–170, Tarragona, Spain, 2014. Springer International Publishing.
- [98] C. N. Lintzmayer and Z. Dias. Sorting permutations by prefix and suffix versions of reversals and transpositions. In *Proceedings of the 11th Latin American Theoretical Informatics Symposium (LATIN'2014)*, volume 8392 of *Lecture Notes in Computer Science*, pages 1–12, Montevideo, Uruguay, 2014. Springer-Verlag.
- [99] L. Lu and Y. Yang. A lower bound on the transposition diameter. *SIAM Journal on Discrete Mathematics*, 24(4):1242–1249, 2010.

- [100] A. McLysaght, C. Seoighe, and K. H. Wolfe. High frequency of inversions during eukaryote gene order evolution. In D. Sankoff and J. H. Nadeau, editors, *Comparative Genomics*, volume 1 of *Computational Biology*, pages 47–58. Springer Netherlands, 2000.
- [101] J. Meidanis, M. E. M. T. Walter, and Z. Dias. A lower bound on the reversal and transposition diameter. *Journal of Computational Biology*, 9(5):743–745, 2002.
- [102] C. Mira. *Análise Algébrica de Problemas de Rearranjo em Genomas: Algoritmos e Complexidade*. PhD thesis, University of Campinas, 2008. In Portuguese.
- [103] J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations and functions. *Theoretical Computer Science*, 22:74–88, 2012.
- [104] W. Myrvold and F. Ruskey. Ranking and unranking permutations in linear time. *Information Processing Letters*, 79(6):281–284, 2001.
- [105] M. Ozery-Flato. *Computational Problems in Genome Rearrangements: from Evolution to Cancer*. PhD thesis, Tel-Aviv University, 2009.
- [106] R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:103–111, 2002.
- [107] A. Rahman, S. Shatabda, and M. Hasan. An approximation algorithm for sorting by reversals and transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.
- [108] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(1):406–425, 1987.
- [109] T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153, 2007.
- [110] C. Seoighe, N. Federspiel, T. Jones, N. Hansen, V. Bivolarovic, R. Surzycki, R. Tamse, C. Komp, L. Huizar, R. W. Davis, S. Scherer, E. Tait, D. J. Shaw, D. Harris, L. Murphy, K. Oliver, K. Taylor, M. A. Rajandream, B. G. Barrell, and K. H. Wolfe. Prevalence of small inversions in yeast gene order evolution. *Proceedings of the National Academy of Sciences USA*, 97(26):14433–14437, 2000.
- [111] M. Sharmin, R. Yeasmin, M. Hasan, A. Rahman, and M. S. Rahman. Pancake flipping with two spatulas. *Electronic Notes in Discrete Mathematics*, 36:231–238, 2010.

- [112] D. E. Smith. *History of Mathematics, Vol. II*. Ginn and Company, Boston, USA, 1925.
- [113] A. H. Sturtevant. The linear arrangement of six sex-linked factors in *Drosophila*. *Journal of Experimental Zoology*, 14:43–59, 1913.
- [114] A. H. Sturtevant. A case of rearrangement of genes in *Drosophila*. *Proceedings of the National Academy of Sciences of the United States of America*, 7(8):235–237, 1921.
- [115] A. H. Sturtevant and T. Dobzhansky. Inversions in the third chromosome of wild races of *Drosophila pseudoobscura*, and their use in the study of the history of the species. *Proceedings of the National Academy of Sciences of the United States of America*, 22(7):448–450, 1936.
- [116] A. H. Sturtevant and E. Novitski. The homologies of the chromosome elements in the genus *Drosophila*. *Genetics*, 26(5):517–541, 1941.
- [117] K. Swenson. *Evolution of whole genomes through inversions: models and algorithms for duplicates, ancestors, and edit scenarios*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2009.
- [118] F. Swidan, M. A. Bender, D. Ge, S. He, H. Hu, and R. Y. Pinter. Sorting by length-weighted reversals: Dealing with signs and circularity. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM'2004)*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46, Istanbul, Turkey, 2004. Springer Berlin Heidelberg.
- [119] E. Tannier, A. Bergeron, and M. F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.
- [120] G. Tesler. GRIMM: genome rearrangements web server. *Bioinformatics*, 18:492–493, 2002.
- [121] J. P. C. Vergara. *Sorting by Bounded Permutations*. PhD thesis, Virginia Polytechnic Institute & State University, 1998.
- [122] M. E. M. T. Walter. *Algoritmos para Problemas em Rearranjo de Genomas*. PhD thesis, University of Campinas, 1999. In Portuguese.
- [123] M. E. M. T. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *Proceedings of the 5th International Symposium on String Processing and Information Retrieval (SPIRE'1998)*, pages 96–102, Santa Cruz, Bolivia, 1998. IEEE Computer Society.

- [124] M. E. M. T. Walter, Z. Dias, and J. Meidanis. A new approach for approximating the transposition distance. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'2000)*, pages 199–208, Washington, DC, USA, 2000. IEEE Computer Society.
- [125] M. E. M. T. Walter, M. C. Sobrinho, E. T. G. Oliveira, L. S. Soares, A. G. Oliveira, T. E. S. Martins, and T. M. Fonseca. Improving the algorithm of Bafna and Pevzner for the problem of sorting by transpositions: a practical approach. *Journal of Discrete Algorithms*, 3(2-4):342–361, 2005.
- [126] G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1–7, 1982.
- [127] A. Williams.  $O(1)$ -time unsorting by prefix-reversals in a boustrophedon linked list. In *Proceedings of the Fifth International Conference on Fun With Algorithms (FUN'2010)*, volume 6099 of *Lecture Notes in Computer Science*, pages 368–379, Ischia Island, Italy, 2010. Springer Berlin Heidelberg.
- [128] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.