Universidade Estadual de Campinas
Instituto de Computação

Alexsandro Oliveira Alexandrino

Problemas de Ordenação de Permutações por Operações Ponderadas

CAMPINAS
2019

Alexsandro Oliveira Alexandrino

Problemas de Ordenação de Permutações por Operações Ponderadas

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Zanoni Dias
Coorientadora: Profa. Dra. Carla Negri Lintzmayer

Este exemplar corresponde à versão final da Dissertação defendida por Alexsandro Oliveira Alexandrino e orientada pelo Prof. Dr. Zanoni Dias.

CAMPINAS
2019

Agência(s) de fomento e nº(s) de processo(s): FAPESP, 2017/16871-1; CNPq, 131182/2017-0
ORCID: https://orcid.org/0000-0002-6320-9747

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

AL27p

Alexandrino, Alexsandro Oliveira, 1995-
Problemas de ordenação de permutações por operações ponderadas / Alexsandro Oliveira Alexandrino. – Campinas, SP : [s.n.], 2019.

Orientador: Zanoni Dias.
Coorientador: Carla Negri Lintzmayer.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Rearranjo de genomas. 2. Biologia computacional. 3. Algoritmos de aproximação. 4. Permutações (Matemática). I. Dias, Zanoni, 1975-. II. Lintzmayer, Carla Negri, 1990-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Sorting permutations by weighted operations
Palavras-chave em inglês:
Genome rearrangements
Computational biology
Approximation algorithms
Permutations (Mathematics)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Mário César San Felice
Rafael Crivellari Saliba Schouery
Data de defesa: 21-02-2019
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Alexsandro Oliveira Alexandrino

Problemas de Ordenação de Permutações por Operações Ponderadas

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação - Universidade Estadual de Campinas
- Prof. Dr. Mário César San Felice
  Departamento de Computação - Universidade Federal de São Carlos
- Prof. Dr. Rafael Crivellari Saliba Schouery
  Instituto de Computação - Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 21 de fevereiro de 2019

Agradecimentos

Agradeço aos meus pais, Pedro e Marileide, por sempre terem me dado o apoio necessário ao longo desses anos, e ao meu irmão, Alex, pela amizade e companheirismo.

À Ana Paula pelo carinho, incentivo, paciência e companhia durante todo esse período.

Aos professores Zanoni Dias e Carla Lintzmayer pela orientação, dedicação, confiança e colaboração. Nesses dois anos de mestrado, aprendi muito com vocês e, sem dúvidas, não poderia ter escolhido melhores orientadores.

Aos membros do LOCo, em especial ao Klairton, Andre, Guilherme, Italos e Mauro.

Aos professores Críston, Lucas Ismaily e Paulyne por terem me incentivado a entrar na Pós-Graduação.

Por fim, agradeço às agências de fomento CNPq (processo 131182/2017-0) e FAPESP (processo 2017/16871-1) pelo apoio financeiro.

Resumo

Calcular a distância evolucionária entre espécies é um problema importante da área de Biologia Computacional. Em várias abordagens, o cálculo dessa distância considera apenas rearranjos de genomas, os quais são conjuntos de mutações que alteram grandes trechos do genoma de um organismo. Assumindo que o genoma não possui genes duplicados, podemos representá-lo como permutações de números inteiros, em que cada elemento corresponde a um bloco conservado (região de alta similaridade entre os genomas comparados) e o sinal de cada elemento corresponde à orientação desse bloco. Ao usar permutações, o problema de transformar um genoma em outro é equivalente ao da Ordenação de Permutações por Operações de Rearranjo. A *abordagem tradicional* desse problema considera que todas as operações tem o mesmo custo e, assim, o objetivo é encontrar uma sequência mínima de operações que ordene a permutação. Entretanto, estudos indicam que algumas operações de rearranjo tem maior probabilidade de acontecer do que outras, fazendo com que abordagens em que operações possuem custos diferentes sejam mais realistas. Nas *abordagens ponderadas*, o objetivo é encontrar a sequência que ordena a permutação, de modo que a soma dos custos dos rearranjos dessa sequência seja mínimo.

Neste trabalho, apresentamos algoritmos de aproximação para duas novas variações dos problemas da Ordenação de Permutações por Operações Ponderadas. A primeira variação utiliza uma função de custo correspondente à *quantidade de fragmentações* que a operação causa na permutação. A segunda variação utiliza uma função de custo proporcional ao *tamanho da operação*, além de adicionar a restrição de que as operações sejam curtas. Para cada uma das variações, foram considerados cinco problemas com os seguintes modelos de rearranjo: reversões sem sinais, reversões com sinais, transposições, reversões sem sinais e transposições, e reversões com sinais e transposições.

Considerando os problemas da Ordenação de Permutações por Operações Ponderadas pelo Número de Fragmentações, apresentamos uma análise da relação entre os problemas não ponderados e essa variação, dois algoritmos de 2-aproximação para cada um dos cinco modelos de rearranjo, resultados experimentais desses algoritmos e limitantes inferiores e superiores para o diâmetro desses problemas. Além disso, apresentamos propriedades sobre permutações simples e um algoritmo de 1.5-aproximação assintótica para essa classe de permutações considerando reversões com sinais e/ou transposições.

Para os problemas da Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho, apresentamos uma análise da relação entre os problemas não ponderados e essa variação, algoritmos de aproximação com fator constante para cada um dos cinco modelos de rearranjo e resultados experimentais desses algoritmos. Além disso, fizemos uma análise do fator de aproximação dos algoritmos quando a função de custo é igual a ℓ^α, onde ℓ é o tamanho da operação e $\alpha > 1$ é uma constante.

Abstract

One of the main problems in Computational Biology is to find the evolutionary distance among species. In most approaches, such distance only involves rearrangements, which are mutations that alter large pieces of the species' genome. Considering that the genome has no repeated genes, we can represent them as signed permutations, where each element corresponds to a synteny block (region of high similarity between the genomes compared) and the sign of each element corresponds to the orientation of such blocks. When using permutations, the problem of transforming one genome into another is equivalent to the problem of Sorting Permutations by Rearrangement Operations. The *traditional approach* is to consider that any rearrangement has the same probability to happen, and so the goal is to find a minimum sequence of operations which sorts the permutation. However, studies have shown that some rearrangements are more likely to happen than others, and so a weighted approach is more realistic. In a *weighted approach*, the goal is to find a sequence which sorts the permutations, such that the sum of the rearrangements' cost of that sequence is minimum.

In this work, we presented approximation algorithms for two new variations of the Sorting Permutations by Weighted Operations problem. The first variation uses a cost function related to the *amount of fragmentation* caused by a rearrangement. The second variation uses a cost function proportional to the *rearrangement's length*, along with the constraint that operations must be short. For each variation, we considered five problems with the following rearrangement models: unsigned reversals, signed reversals, transpositions, unsigned reversals and transpositions, and signed reversals and transpositions.

Considering the problems of Sorting Permutations by Fragmentation-Weighted Operations, we presented an analysis of the relation between the traditional approach and this variation, 2-approximation algorithms for each rearrangement model, experimental results of these algorithms, and upper and lower bounds for the diameter of these problems. Besides that, we showed properties of simple permutations and a 1.5-asymptotic approximation algorithm for this class of permutation considering signed reversals and/or transpositions.

Considering the problems of Sorting Permutations by Length-Weighted Short Operations, we presented an analysis of the relation between the traditional approach and this variation, approximation algorithms with constant factor for each rearrangement model, and experimental results for these algorithms. Besides that, we analyzed the approximation factor for the algorithms we developed when the cost function is equal to ℓ^α, where ℓ is the rearrangement's length and $\alpha > 1$ is a constant.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Um dos principais desafios da Biologia Computacional é o de estimar a *distância evolucionária* entre diferentes organismos, considerando que novos organismos surgem a partir de mutações que ocorreram no material genético de outros organismos. Uma forma de estimar a distância evolucionária entre dois organismos é considerar a *distância de rearranjo* entre seus genomas, que consiste no tamanho da sequência de operações de rearranjo que ocorreram na transformação de um genoma em outro. Um *rearranjo de genoma* ou *operação de rearranjo* é um tipo de mutação em larga escala que altera segmentos de um genoma.

Os tipos de rearranjos mais estudados na literatura são as reversões e as transposições. Uma reversão inverte um segmento qualquer do genoma, enquanto uma transposição faz a troca de dois segmentos adjacentes do genoma. Quando essas operações envolvem o primeiro elemento da permutação, elas são chamadas de operações de *prefixo*. De forma similar, ao envolver o último elemento da permutação, elas são chamadas de operações de *sufixo*. O *tamanho* de uma operação é igual à quantidade de elementos afetados por ela.

O genoma de um organismo é formado por cromossomos, representados como um conjunto ordenado de genes. Um *bloco conservado* é uma região de alta similaridade entre dois genomas distintos, podendo ser um gene ou uma sequência de genes. Ao considerar a comparação de genomas, representamos um genoma como uma sequência de blocos conservados. Assumindo que não existem genes duplicados, essa representação é modelada matematicamente como uma permutação de números inteiros. Genes e blocos possuem uma orientação e, quando essa orientação é conhecida, o genoma é representado como uma *permutação com sinais*, onde o sinal indica a orientação do gene ou bloco. Quando a orientação não é conhecida, o genoma é representado como uma *permutação sem sinais*.

Ao usar permutações, os problemas de Rearranjo de Genomas consistem em encontrar uma sequência de rearranjos de custo mínimo que transformam uma permutação em outra. Devido às propriedades algébricas das permutações, esses problemas são equivalentes aos de ordenar uma permutação com uma sequência de rearranjos de custo mínimo.

A Ordenação de Permutações por Reversões e a Ordenação de Permutações por Transposições são problemas NP-Difíceis [7, 8]. O melhor resultado para ambos os problemas é uma 1.375-aproximação [5, 13]. No entanto, Hannenhalli e Pevzner [18] provaram que existe algoritmo polinomial para o problema da Ordenação de Permutações por Reversões

com Sinais.

Já a Ordenação de Permutações por Reversões e Transposições ainda possui sua complexidade em aberto. Walter *et al.* [35] apresentaram uma 2-aproximação, para permutações com sinais, e uma 3-aproximação, para permutações sem sinais. Ainda para a versão sem sinais, Rahman *et al.* [32] apresentaram uma $2k$-aproximação, onde k é o fator de aproximação do algoritmo utilizado para a decomposição máxima de ciclos do grafo de *breakpoints*. O melhor valor de k conhecido [9] é igual a $1.4167 + \epsilon$, para $\epsilon > 0$.

Todos os problemas citados anteriormente consideram a abordagem tradicional. Nessa abordagem, assumimos que qualquer operação tem a mesma probabilidade de acontecer e, portanto, a distância é igual ao tamanho da sequência mínima de operações que ordena uma permutação. No entanto, estudos mostram que algumas operações de rearranjo são mais prováveis de acontecer do que outras [2, 6] e, assim, uma abordagem ponderada é mais próxima do processo evolutivo real. Ainda não existem estudos conclusivos para a análise da frequência das operações e, consequentemente, é desconhecido quais valores de custo são os mais próximos à realidade [6].

Uma das funções de custo presentes na literatura é a ponderação pelo tipo de operação. Normalmente, nessa variação do problema, uma transposição recebe um custo maior que uma reversão, já que existem evidências de que reversões ocorrem em uma proporção maior que transposições em cenários reais [2, 6]. Eriksen [15] apresentou um estudo com o objetivo de encontrar qual ponderação para reversões e transposições é mais relacionada ao processo evolutivo. Os melhores cenários foram encontrados usando pesos 2 e 3 para reversões e transposições, respectivamente. Este estudo não considerou operações de prefixo e sufixo separadamente dos outros rearranjos.

Além disso, existem variações do problema de rearranjo motivadas pela observação de que, em alguns casos, as operações que ocorreram no processo evolutivo não agem em regiões muito grandes [6, 24]. Dessa observação surgiram os problemas da Ordenação por Operações de Tamanho Limitado [16, 20, 21, 22, 34] e Ordenação por Operações Ponderadas pelo Tamanho [26, 29, 31].

Neste trabalho, estudamos duas novas variações dos problemas da Ordenação de Permutações por Operações de Rearranjos: Ordenação de Permutações por Operações Ponderadas pelo Número de Fragmentações e Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho. A primeira variação introduz o estudo de uma função de custo associada à quantidade de fragmentações que uma operação de rearranjo causa em um genoma. A segunda variação combina a função de custo proporcional ao tamanho da operação com a restrição de que apenas operações curtas (tamanho menor ou igual a 3) são permitidas. Para as duas variações, serão consideradas permutações com e sem sinais para os modelos com reversões, transposições e combinação de reversões e transposições. A Tabela 1.1 apresenta os acrônimos (em inglês) para todos os problemas estudados neste trabalho.

Essa dissertação está organizada da seguinte forma. O Capítulo 2 apresenta notações e conceitos relacionados aos problemas estudados. O Capítulo 3 apresenta resultados para o problema da Ordenação de Permutações por Operações Ponderadas pelo Número de Fragmentações. O Capítulo 4 apresenta resultados para o problema da Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho. E, por último, o Capí-

tulo 5 apresenta algumas considerações finais e um resumo dos resultados alcançados.

Tabela 1.1: Acrônimos dos problemas estudados neste trabalho. Cada acrônimo é referente ao nome em inglês do respectivo problema.

Problema	**Acrônimo**
Sorting by Fragmentation-Weighted Unsigned Reversals Ordenação por Reversões sem Sinais Ponderadas pelo Número de Fragmentações	SbFWR
Sorting by Fragmentation-Weighted Signed Reversals Ordenação por Reversões com Sinais Ponderadas pelo Número de Fragmentações	SbFW$\bar{\text{R}}$
Sorting by Fragmentation-Weighted Transpositions Ordenação por Transposições Ponderadas pelo Número de Fragmentações	SbFWT
Sorting by Fragmentation-Weighted Unsigned Reversals and Transpositions Ordenação por Reversões sem Sinais e Transposições Ponderadas pelo Número de Fragmentações	SbFWRT
Sorting by Fragmentation-Weighted Signed Reversals and Transpositions Ordenação por Reversões com Sinais e Transposições Ponderadas pelo Número de Fragmentações	SbFW$\bar{\text{R}}$T
Sorting by Length-Weighted Short Unsigned Reversals Ordenação por Reversões sem Sinais Curtas Ponderadas pelo Tamanho	SbLWsR
Sorting by Length-Weighted Short Signed Reversals Ordenação por Reversões com Sinais Curtas Ponderadas pelo Tamanho	SbLWs$\bar{\text{R}}$
Sorting by Length-Weighted Short Transpositions Ordenação por Transposições Curtas Ponderadas pelo Tamanho	SbLWsT
Sorting by Length-Weighted Short Unsigned Reversals and Short Transpositions Ordenação por Reversões sem Sinais Curtas e Transposições Curtas Ponderadas pelo Tamanho	SbLWsRsT
Sorting by Length-Weighted Short Signed Reversals and Short Transpositions Ordenação por Reversões com Sinais Curtas e Transposições Curtas Ponderadas pelo Tamanho	SbLWs$\bar{\text{R}}$sT

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta notações e conceitos que serão utilizados no restante deste trabalho. Essas definições formam a base para o estudo dos problemas da Ordenação de Permutações por Operações de Rearranjo. No entanto, algumas definições específicas para variações do problema serão apresentadas no seu capítulo correspondente.

2.1 Representação de Genomas

Um genoma é modelado matematicamente como uma *permutação com sinais*. Cada elemento da permutação representa um gene ou uma sequência de genes (bloco conservado), e o sinal positivo ou negativo representa a orientação do bloco. Quando a orientação dos blocos conservados não é conhecida, o genoma é modelado como uma *permutação sem sinais*.

Definição 1. *Uma permutação sem sinais π de tamanho n é definida como $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, tal que $\pi_i \in \{1, 2, \ldots, n\}$ e $\pi_i \neq \pi_j$ se, e somente se, $i \neq j$, com $1 \leq i \leq n$ e $1 \leq j \leq n$.*

Definição 2. *Uma permutação com sinais π de tamanho n é definida como $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, tal que $\pi_i \in \{-n, -(n-1), \ldots, -1, 1, 2, \ldots, n\}$ e $|\pi_i| \neq |\pi_j|$ se, e somente se, $i \neq j$, com $1 \leq i \leq n$ e $1 \leq j \leq n$.*

As *adjacências* de uma permutação π são todos os pares de elementos consecutivos (π_i, π_{i+1}) de π, com $1 \leq i < n$. A *permutação identidade* ι é a permutação ordenada $(1\ 2\ \ldots\ n)$. Ao considerar permutações com sinais, todos os elementos de ι possuem sinal positivo. A *permutação reversa* é definida como $\eta = (n\ (n-1)\ \ldots\ 1)$, já a *permutação reversa com sinais* é definida como $\bar{\eta} = (-n\ -(n-1)\ \ldots\ -1)$. Essas permutações são importantes na demonstração de propriedades dos problemas de ordenação.

Definição 3. *A* composição *de duas permutações π e σ é a operação definida por $\pi \circ \sigma = (\alpha_1\ \alpha_2\ \ldots\ \alpha_n)$, tal que $\alpha_i = -\pi_{|\sigma_i|}$, se $\sigma_i < 0$, e $\alpha_i = \pi_{\sigma_i}$, se $\sigma_i > 0$.*

Exemplo 1. *A composição das permutações $\pi = (+5\ +3\ -2\ -1\ +4)$ e $\sigma = (-2\ -1\ +5\ +4\ -3)$ é igual a $\pi \circ \sigma = (-3\ -5\ +4\ -1\ +2)$.*

A *inversa* de uma permutação π é a permutação π^{-1} que satisfaz a igualdade $\pi \circ \pi^{-1} = \pi^{-1} \circ \pi = \iota$. A inversa é útil para indicar a posição e o sinal de cada elemento de π.

Exemplo 2. *A inversa da permutação* $\pi = (+5\ +3\ -2\ -1\ +4)$ *é igual a* $\pi^{-1} = (-4\ -3\ +2\ +5\ +1)$.

2.2 Rearranjos de Genomas

Para a representação de rearranjos também são utilizadas permutações. Assim, a aplicação de um rearranjo β em uma permutação π resulta na permutação $\pi \circ \beta$, ou seja, utilizamos a composição de permutações para indicar a ocorrência de um rearranjo β em uma permutação π. Os dois tipos de rearranjo estudados neste trabalho são Reversões e Transposições.

Definição 4. *Uma* reversão sem sinais $\rho(i,j)$, *para* $1 \leq i < j \leq n$, *é o rearranjo* $(1\ 2\ \ldots\ i{-}1\ j\ j{-}1\ \ldots\ i{+}1\ i\ j{+}1\ \ldots\ n)$ *que, quando aplicado a uma permutação* π, *inverte o segmento* $\langle \pi_i, \pi_{i+1}, \ldots, \pi_j \rangle$, *transformando* π *na permutação* $\pi \circ \rho(i,j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$. *Uma* reversão sem sinais de prefixo *é uma operação da forma* $\rho(1,j)$, *para* $1 < j \leq n$. *Já uma* reversão sem sinais de sufixo *é uma operação da forma* $\rho(i,n)$, *para* $1 \leq i < n$.

Exemplo 3. *A aplicação da reversão sem sinais* $\rho(3,6)$ *na permutação* $\pi = (3\ 2\ \underline{6\ 5\ 4\ 1})$ *resulta na permutação* $\pi \circ \rho(3,6) = (3\ 2\ \underline{1\ 4\ 5\ 6})$.

Definição 5. *Uma* reversão com sinais $\bar{\rho}(i,j)$, *para* $1 \leq i \leq j \leq n$, *é o rearranjo* $(1\ 2\ \ldots\ i{-}1\ {-}j\ {-}(j{-}1)\ \ldots\ {-}(i{+}1)\ {-}i\ j{+}1\ \ldots\ n)$ *que, quando aplicado a uma permutação* π, *inverte o segmento* $\langle \pi_i, \pi_{i+1}, \ldots, \pi_j \rangle$ *e troca os sinais dos elementos desse segmento, transformando* π *na permutação* $\pi \circ \bar{\rho}(i,j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{-\pi_j\ -\pi_{j-1}\ \ldots\ -\pi_{i+1}\ -\pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$. *Uma* reversão com sinais de prefixo *é uma operação da forma* $\bar{\rho}(1,j)$, *para* $1 \leq j \leq n$. *Já uma* reversão com sinais de sufixo *é uma operação da forma* $\bar{\rho}(i,n)$, *para* $1 \leq i \leq n$.

Exemplo 4. *A aplicação da reversão com sinais* $\bar{\rho}(1,3)$ *na permutação* $\pi = (\underline{-3\ -2\ -1}\ +4\ +5\ +6)$ *resulta na permutação* $\pi \circ \bar{\rho}(1,3) = (\underline{+1\ +2\ +3}\ +4\ +5\ +6)$.

Ao longo do texto, o termo reversões e a notação ρ são utilizados para denotar reversões de uma forma genérica sem considerar o efeito no sinal dos elementos. Essa terminologia é usada para simplificar conceitos análogos para ambas as operações e em alguns algoritmos apresentados.

Definição 6. *Uma* transposição $\tau(i,j,k)$, *para* $1 \leq i < j < k \leq n+1$, *é o rearranjo* $(1\ 2\ \ldots\ i-1\ j\ j{+}1\ \ldots\ k{-}1\ i\ i{+}1\ \ldots\ j{-}1\ k\ \ldots\ n)$ *que, quando aplicado a uma permutação* π, *troca de posição o segmento* $\langle \pi_i, \pi_{i+1}, \ldots, \pi_{j-1} \rangle$ *com o segmento adjacente* $\langle \pi_j, \pi_{j+1}, \ldots, \pi_{k-1} \rangle$, *transformando* π *na permutação* $\pi \circ \tau(i,j,k) = (\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j+1}\ \ldots\ \pi_{k-1}}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}}\ \pi_k\ \ldots\ \pi_n)$. *Uma* transposição de prefixo *é uma operação da forma* $\tau(1,j,k)$, *para* $1 < j < k \leq n+1$. *Já uma* transposição de sufixo *é uma operação da forma* $\tau(i,j,n+1)$, *para* $1 \leq i < j < n+1$.

Exemplo 5. *A aplicação da transposição* $\tau(1,3,7)$ *na permutação* $\pi = (\underline{5\ 6}\ \underline{4\ 3\ 2\ 1})$ *resulta na permutação* $\pi \circ \tau(1,3,7) = (\underline{4\ 3\ 2\ 1}\ \underline{5\ 6})$.

Além das operações de prefixo e sufixo, que envolvem o primeiro e último elemento, respectivamente, existem as operações completas. Uma operação é *completa* se envolve todos os elementos da permutação.

Definição 7. *O* tamanho $|\beta|$ *de uma operação* β *é igual a quantidade de elementos afetados pela aplicação de* β *em uma permutação qualquer. O tamanho de uma reversão* $\rho(i,j)$ *é igual a* $j-i+1$. *Já o tamanho de uma transposição* $\tau(i,j,k)$ *é igual a* $k-i$.

Definição 8. *Uma operação* β *é* super curta *se* $|\beta| \leq 2$ *e* curta *se* $|\beta| \leq 3$.

Definição 9. *Uma reversão* ρ *é uma* k-reversão *se* $k = |\rho|$.

2.3 Distância e Diâmetro

Um *modelo de rearranjo* M é o conjunto de operações de rearranjo consideradas em um problema de ordenação. Um modelo de rearranjo pode ser formado por mais de um tipo de rearranjo.

Considerando as operações de reversão e transposição, denotamos por r, $\bar{r}$, t, rt e $\bar{r}t$ os modelos de rearranjo que permitem reversões sem sinais, reversões com sinais, transposições, reversões sem sinais e transposições, e reversões com sinais e transposições, respectivamente. Uma transposição não afeta os sinais dos segmentos afetados e, assim, problemas que permitem apenas transposições sempre consideram permutações sem sinais.

Ao considerar modelos de rearranjo permitindo apenas operações super curtas ou curtas, adicionamos à notação do modelo o prefixo *ss* (super curtas, do inglês *super short*) ou *s* (curtas, do inglês *short*). Como exemplo, *ssr* é o modelo de rearranjo que permite apenas reversões sem sinais super curtas.

Dada uma sequência de operações $S = \langle \beta_1, \beta_2, \ldots, \beta_\ell \rangle$, denotamos por $\pi \circ S$ a aplicação das operações de S em π, tal que $\pi \circ S = \pi \circ \beta_1 \circ \beta_2 \circ \ldots \circ \beta_\ell$. O tamanho da sequência S é denotado por $|S|$.

Para toda operação β, existe um custo associado à aplicação de β em π, e esse custo é denotado pela função $f : M \to \mathbb{R}$, onde M é o modelo de rearranjo considerado. O custo de uma sequência de operações $S = \langle \beta_1, \beta_2, \ldots, \beta_\ell \rangle$ é igual a $f(S) = \sum_{i=1}^{\ell} f(\beta_i)$.

O principal objetivo dos problemas de rearranjo de genomas é encontrar a distância entre duas espécies. A seguir apresentamos a definição de distância entre duas permutações.

Definição 10. *Dados um modelo de rearranjo* M, *uma função de custo* f *e as permutações* π *e* σ, *a* distância $d_M^f(\pi,\sigma)$ *é o custo de uma sequência de operações* $S = \langle \beta_1, \beta_2, \ldots, \beta_\ell \rangle$ *pertencentes a* M, *tal que* $\pi \circ S = \sigma$ *e* $\sum_{i=1}^{\ell} f(\beta_i)$ *é mínimo.*

Quando $f(\beta) = 1$ para todo $\beta \in M$, a distância $d_M(\pi,\sigma)$ entre π e σ é equivalente a encontrar o número mínimo de operações que transformam π em σ.

Seja γ uma permutação qualquer. A *distância de ordenação* de γ é denotada por $d_M^f(\gamma)$, de modo que $d_M^f(\gamma) = d_M^f(\gamma, \iota)$. O problema de transformar π em σ é equivalente ao de ordenar $\gamma = \sigma^{-1} \circ \pi$, já que $d_M^f(\pi, \sigma) = d_M^f(\sigma^{-1} \circ \pi, \sigma^{-1} \circ \sigma) = d_M^f(\gamma, \iota) = d_M^f(\gamma)$.

Como um objetivo secundário dos problemas de ordenação, também podemos estudar o diâmetro dos problemas de ordenação por operações de rearranjo.

Definição 11. *Dados um modelo de rearranjo M e uma função de custo f, o* diâmetro *de tamanho n é denotado por $D_M^f(n) = \max\{d_M^f(\pi) : \pi$ tem tamanho $n\}$, com $n \in \mathbb{Z}^+$. Ou seja, $D_M^f(n)$ é a maior distância $d_M^f(\pi)$ entre todas as permutações de tamanho n.*

Para a abordagem tradicional (não ponderada) dos problemas de ordenação por rearranjos, utilizamos $d_M(\pi)$ e $D_M(n)$ para indicar a distância de ordenação e o diâmetro, respectivamente.

2.4 Algoritmos de Aproximação

Muitos dos problemas de ordenação por rearranjo de genomas são NP-Difíceis. Portanto, a não ser que P = NP, não existem algoritmos polinomiais para encontrar soluções ótimas de tais problemas. Uma das abordagens utilizadas para a resolução de problemas NP-Difíceis é o desenvolvimento de algoritmos polinomiais que encontram soluções viáveis próximas da solução ótima, garantindo também o quão próximo a solução encontrada está da solução ótima. Esses algoritmos são chamados de algoritmos de aproximação.

Definição 12. *Seja ALG um algoritmo polinomial para um problema de minimização. Seja $ALG(I)$ o custo da solução encontrada por ALG e $OPT(I)$ o custo de uma solução ótima para a instância I. O algoritmo ALG é uma α-aproximação se $ALG(I) \leq \alpha \times OPT(I)$, para toda instância I.*

Definição 13. *Seja ALG um algoritmo polinomial para um problema de minimização. Seja $ALG(I)$ o custo da solução encontrada por ALG e $OPT(I)$ o custo de uma solução ótima para a instância I. O algoritmo ALG é uma α-aproximação assintótica se $ALG(I) \leq \alpha \times OPT(I) + c$, onde c é uma constante, para toda instância I.*

Definição 14. *Seja ALG um algoritmo de aproximação para um problema de minimização. Dados uma instância I e um limitante inferior $LB(I)$ para o valor da solução ótima para a instância I, a* qualidade *de ALG em relação a instância I é igual a $ALG(I)/LB(I)$, onde $ALG(I)$ é o custo da solução encontrada por ALG para a instância I. Ou seja, a qualidade indica o quão próximo o custo da solução encontrada por ALG está do valor $LB(I)$.*

Ao longo deste trabalho, apresentaremos algoritmos de aproximação e algoritmos de aproximação assintótica para os problemas estudados.

Capítulo 3

Operações Ponderadas pelo Número de Fragmentações

Uma das funções de custo mais estudadas na literatura é a ponderação pelo tipo de rearranjo. Normalmente, nessa variação do problema, uma transposição recebe um custo maior que uma reversão, já que existem evidências de que reversões ocorrem em uma proporção maior que transposições em cenários reais [2, 6].

Gu *et al.* [17] introduziram o estudo de uma nova operação chamada de transposição reversa. Uma transposição reversa troca dois segmentos adjacentes de lugar e inverte os elementos de um desses segmentos. Para uma proporção de custos de 2:1 (custo transposição : custo reversão) e considerando que o custo de uma transposição reversa é igual ao de uma transposição, Eriksen [15] apresentou uma 7/6-aproximação e um esquema de aproximação de tempo polinomial (PTAS, do inglês *Polynomial-Time Approximation Scheme*). Após esse estudo, Bader e Ohlebusch [2] apresentaram uma 1.5-aproximação para qualquer proporção de custos entre 1:1 e 2:1 (custo transposição : custo reversão), também considerando que o custo de uma transposição é igual ao de uma transposição reversa.

Eriksen [14] apresentou um estudo com o objetivo de encontrar quais custos para reversões e transposições mais aproximam a distância de rearranjo do processo evolucionário. Como métrica de proximidade para o processo evolucionário, Eriksen [14] considerou a ponderação que produz as distâncias de menor custo. Os melhores cenários foram encontrados quando os custos de reversões e transposições são 2 e 3, respectivamente. Esse estudo não considerou operações de prefixo e sufixo separadamente das outras operações.

Neste capítulo, consideramos uma nova ponderação que é proporcional à quantidade de fragmentações que uma operação causa na permutação. Esses custos são similares aos apresentados por Eriksen [14], diferindo apenas nas operações de prefixo e sufixo, que causam menos fragmentações na permutação. A seguir, definimos como uma operação causa fragmentações e detalhamos a ponderação pelo número de fragmentações.

Definição 15. *Para qualquer par de posições* $(i, i+1)$ *de uma permutação* π*, uma operação* β *causa* fragmentação *entre* $(i, i+1)$ *se* π_i *e* π_{i+1} *não são adjacentes em* $\pi \circ \beta$*.*

Definição 16. *A função* $f : M \to \mathbb{R}$*, onde* M *é o modelo de rearranjo considerado, para*

reversões ponderadas é definida como

$$f(\rho(i,j)) = \begin{cases} 0, & \textit{se } i = 1 \textit{ e } j = n \\ 1, & \textit{se } i = 1 \textit{ e } j < n \\ 1, & \textit{se } i > 1 \textit{ e } j = n \\ 2, & \textit{caso contrário.} \end{cases} \tag{3.1}$$

Podemos observar a seguir como cada tipo de reversão altera uma permutação π.

$$\pi = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1} \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_n) \tag{3.2}$$
$$\pi \circ \rho(i,j) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \pi_{j-1} \ \ldots \ \pi_{i+1} \ \pi_i} \ \pi_{j+1} \ \ldots \ \pi_n) \tag{3.3}$$
$$\pi \circ \rho(1,j) = (\underline{\pi_j \ \pi_{j-1} \ \ldots \ \pi_{i+1} \ \pi_i \ \pi_{i-1} \ \ldots \ \pi_1} \ \pi_{j+1} \ \ldots \ \pi_n) \tag{3.4}$$
$$\pi \circ \rho(i,n) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_n \ \ldots \ \pi_{j+1} \ \pi_j \ \pi_{j-1} \ \ldots \ \pi_{i+1} \ \pi_i}) \tag{3.5}$$
$$\pi \circ \rho(1,n) = (\underline{\pi_n \ \ldots \ \pi_{j+1} \ \pi_j \ \pi_{j-1} \ \ldots \ \pi_{i+1} \ \pi_i \ \pi_{i-1} \ \ldots \ \pi_1}) \tag{3.6}$$

Definição 17. *A função $f : M \to \mathbb{R}$, onde M é o modelo de rearranjo considerado, para transposições ponderadas é definida como*

$$f(\tau(i,j,k)) = \begin{cases} 1, & \textit{se } i = 1 \textit{ e } k = n+1 \\ 2, & \textit{se } i = 1 \textit{ e } k < n+1 \\ 2, & \textit{se } i > 1 \textit{ e } k = n+1 \\ 3, & \textit{caso contrário.} \end{cases} \tag{3.7}$$

Podemos observar a seguir como cada tipo de transposição altera uma permutação π.

$$\pi = (\pi_1 \ \ldots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1} \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1} \ \pi_k \ \ldots \ \pi_n) \tag{3.8}$$
$$\pi \circ \tau(i,j,k) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1}} \ \underline{\pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1}} \ \pi_k \ \ldots \ \pi_n) \tag{3.9}$$
$$\pi \circ \tau(1,j,k) = (\underline{\pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1}} \ \underline{\pi_1 \ \ldots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1}} \ \pi_k \ \ldots \ \pi_n) \tag{3.10}$$
$$\pi \circ \tau(i,j,n+1) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1} \ \pi_k \ \ldots \ \pi_n} \ \underline{\pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1}}) \tag{3.11}$$
$$\pi \circ \tau(1,j,n+1) = (\underline{\pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1} \ \pi_k \ \ldots \ \pi_n} \ \underline{\pi_1 \ \ldots \ \pi_{i-1} \ \pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1}}) \tag{3.12}$$

Exemplo 6. *Seja $\pi = (1\ 3\ 8\ 5\ 4\ 7\ 6\ 2)$. Para o problema da Ordenação de Permutações por Reversões sem Sinais não Ponderadas, uma sequência ótima S para ordenar π é igual a $\langle \rho(3,5), \rho(5,7), \rho(2,7), \rho(2,8) \rangle$. Ao aplicar a ponderação por fragmentações em S, temos que os custos das operações $\rho(3,5)$, $\rho(5,7)$, $\rho(2,7)$ e $\rho(2,8)$ são 2, 2, 2 e 1, respectivamente. Assim, o custo total da sequência é igual a 7. Para o problema da Ordenação de Permutações por Reversões sem Sinais Ponderadas por Fragmentações, a sequência ótima $S' = \langle \rho(3,5), \rho(1,7), \rho(1,3), \rho(7,8), \rho(1,8) \rangle$ ordena π com custo 5.*

Para essa abordagem ponderada por fragmentações são considerados os problemas com as operações de reversões sem sinais (SbFWR), reversões com sinais (SbFW$\bar{\text{R}}$), transpo-

sições (SbFWT), reversões sem sinais e transposições (SbFWRT), reversões com sinais e transposições (SbFW$\bar{\text{R}}$T). Neste capítulo, consideramos que o custo $f(\beta)$ de uma operação β segue a ponderação pelo número de fragmentações, exceto quando especificado o contrário.

O restante deste capítulo é dividido da seguinte forma. A Seção 3.1 relaciona a abordagem tradicional com a abordagem ponderada. A Seção 3.2 apresenta limitantes inferiores e algoritmos de 2-aproximação para os problemas estudados neste capítulo. A Seção 3.3 introduz um limitante inferior no número de permutações simples, além de apresentar um algoritmo de 1.5-aproximação assintótica para os problemas SbFWT e SbFWRT considerando permutações simples. Por último, a Seção 3.4 apresenta limitantes para o diâmetro dos problemas estudados neste capítulo.

3.1 Relação com a Abordagem Não Ponderada

Esta seção apresenta resultados sobre a relação dos problemas não ponderados com os problemas ponderados por fragmentações. Os lemas 1, 2 e 3 demonstram propriedades sobre sequências de ordenação para problemas envolvendo reversões ponderadas.

Lema 1. *Para toda permutação π, qualquer sequência de operações $S = \langle \rho(1,n), \beta \rangle$, onde β é uma reversão ou transposição, pode ser substituída por uma sequência $S' = \langle \beta', \rho(1,n) \rangle$, tal que $f(S') = f(S)$ e $\pi \circ S' = \pi \circ S$.*

Demonstração. Podemos dividir a prova nos seguintes casos:

1. Se $\beta = \rho(i,j)$, então $S' = \langle \rho(i',j'), \rho(1,n) \rangle$, com $i' = n+1-j$ e $j' = n+1-i$. Ilustramos abaixo como ambas sequências afetam π, sendo que π^2 é o resultado da aplicação das operações de S em π e σ^2 é o resultado da aplicação das operações de S' em π.

$$\begin{aligned}
\pi^0 = \pi \qquad\qquad &= (\pi_1\ \pi_2\ \ldots\ \pi_{i'}\ \ldots\ \pi_{j'}\ \ldots\ \pi_{n-1}\ \pi_n) \\
\pi^1 = \pi^0 \circ \rho(1,n) &= (\underline{\pi_n\ \pi_{n-1}\ \ldots\ \pi_{j'}\ \ldots\ \pi_{i'}\ \ldots\ \pi_2\ \pi_1}) \\
\pi^2 = \pi^1 \circ \rho(i,j) &= (\pi_n\ \pi_{n-1}\ \ldots\ \underline{\pi_{i'}\ \ldots\ \pi_{j'}}\ \ldots\ \pi_2\ \pi_1) \\
\sigma^1 = \pi^0 \circ \rho(i',j') &= (\pi_1\ \pi_2\ \ldots\ \underline{\pi_{j'}\ \ldots\ \pi_{i'}}\ \ldots\ \pi_{n-1}\ \pi_n) \\
\sigma^2 = \sigma^1 \circ \rho(1,n) &= (\underline{\pi_n\ \pi_{n-1}\ \ldots\ \pi_{i'}\ \ldots\ \pi_{j'}\ \ldots\ \pi_2\ \pi_1})
\end{aligned}$$

2. Se $\beta = \tau(i,j,k)$, então $S' = \langle \tau(i',j',k'), \rho(1,n) \rangle$, com $i' = n+1-(k-1)$, $j' = n+1-(j-1)$ e $k' = n+1-(i-1)$. Ilustramos abaixo como ambas sequências afetam π, sendo que π^2 é o resultado da aplicação das operações de S em π e σ^2 é

o resultado da aplicação das operações de S' em π.

$$\begin{aligned}
\pi^0 = \pi &= (\pi_1\ \pi_2\ \ldots\ \pi_{i'-1}\ \pi_{i'}\ \ldots\ \pi_{j'-1}\ \pi_{j'}\ \ldots\ \pi_{k'-1}\ \pi_{k'}\ \ldots\ \pi_{n-1}\ \pi_n)\\
\pi^1 = \pi^0 \circ \rho(1,n) &= (\underline{\pi_n\ \pi_{n-1}\ \ldots\ \pi_{k'}\ \pi_{k'-1}\ \ldots\ \pi_{j'}\ \pi_{j'-1}\ \ldots\ \pi_{i'}\ \pi_{i'-1}\ \ldots\ \pi_2\ \pi_1})\\
\pi^2 = \pi^1 \circ \tau(i,j,k) &= (\pi_n\ \pi_{n-1}\ \ldots\ \pi_{k'}\ \underline{\pi_{j'-1}\ \ldots\ \pi_{i'}}\ \underline{\pi_{k'-1}\ \ldots\ \pi_{j'}}\ \pi_{i'-1}\ \ldots\ \pi_2\ \pi_1)\\
\sigma^1 = \pi^0 \circ \tau(i',j',k') &= (\pi_1\ \pi_2\ \ldots\ \pi_{i'-1}\ \underline{\pi_{j'}\ \ldots\ \pi_{k'-1}}\ \underline{\pi_{i'}\ \ldots\ \pi_{j'-1}}\ \pi_{k'}\ \ldots\ \pi_{n-1}\ \pi_n)\\
\sigma^2 = \sigma^1 \circ \rho(1,n) &= (\underline{\pi_n\ \pi_{n-1}\ \ldots\ \pi_{k'}\ \pi_{j'-1}\ \ldots\ \pi_{i'}\ \pi_{k'-1}\ \ldots\ \pi_{j'}\ \pi_{i'-1}\ \ldots\ \pi_2\ \pi_1})
\end{aligned}$$

Note que $f(S) = f(S')$, já que essas operações causam o mesmo número de fragmentações em ambos os casos. □

Lema 2. *Para toda permutação π, qualquer sequência de operações $S = \langle \rho(1,n), \beta_1, \ldots, \beta_\ell, \rho(1,n)\rangle$, onde β_i é uma reversão ou transposição, pode ser substituída por uma sequência $S' = \langle \beta'_1, \ldots, \beta'_\ell\rangle$, tal que $f(S') = f(S)$ e $\pi \circ S' = \pi \circ S$.*

Demonstração. Provaremos por indução em ℓ que a sequência S' existe.

Para $\ell = 0$, temos que $\pi \circ S = \pi$ e S' é uma sequência vazia. Note que a aplicação de duas reversões completas não altera π.

Suponha que $\ell > 0$ e que qualquer sequência de operações $S = \langle \rho(1,n), \beta_1, \ldots, \beta_{\ell-1}, \rho(1,n)\rangle$, tal que β_i é uma reversão ou transposição, pode ser substituída por uma sequência $S' = \langle \beta'_1, \ldots, \beta'_{\ell-1}\rangle$, tal que $f(S') = f(S)$ e $\pi \circ S' = \pi \circ S$.

Seja $S = \langle \rho(1,n), \beta_1, \ldots, \beta_\ell, \rho(1,n)\rangle$, tal que β_i é uma reversão ou transposição. Pelo Lema 1, existe subsequência $\langle \beta'_1, \rho(1,n)\rangle$ com mesmo efeito de $\langle \rho(1,n), \beta_1\rangle$, tal que $f(\beta'_1) = f(\beta_1)$. Seja $O = \langle \beta'_1, \rho(1,n), \beta_2, \ldots, \beta_\ell, \rho(1,n)\rangle$ e $O' = \langle \rho(1,n), \beta_2, \ldots, \beta_\ell, \rho(1,n)\rangle$. Note que $f(O) = f(S)$ e $\pi \circ O = \pi \circ S$. Pela hipótese de indução, existe subsequência $O'' = \langle \beta''_1, \beta''_2, \ldots, \beta''_{\ell-1}\rangle$, tal que $f(O'') = f(O')$ e $\pi \circ O'' = \pi \circ O'$. Assim, construímos $S' = \langle \beta'_1, \beta''_1, \beta''_2, \ldots, \beta''_{\ell-1}\rangle$ para garantir que $|S'| = \ell$, $f(S) = f(S')$ e $\pi \circ S = \pi \circ S'$. □

Lema 3. *Considerando os problemas SbFWR, SbFW$\bar{R}$, SbFWRT e SbFW$\bar{R}$T, para toda permutação π existe uma sequência de ordenação ótima contendo no máximo uma reversão completa $\rho(1,n)$ de custo 0.*

Demonstração. Considere o problema SbFWR. Seja S uma sequência de ordenação ótima que possui k reversões completas. Se $k \leq 1$, então o resultado segue. Se $k > 1$, então existe uma sequência de ordenação S' de tamanho $|S| - k + (k \mod 2)$, tal que $f(S) = f(S')$ e S' possui no máximo uma reversão completa. A sequência S' pode ser construída pela substituição de toda subsequência $\langle \rho(1,n), \beta_1, \ldots, \beta_\ell, \rho(1,n)\rangle$ em S por uma subsequência de mesmo custo $\langle \beta'_1, \ldots, \beta'_\ell\rangle$ (Lema 2).

A prova é análoga para os outros problemas. □

Os lemas 4, 5 e 6 definem limitantes para a distância de ordenação $d^f_M(\pi)$ a partir de $d_M(\pi)$.

Lema 4. *Para qualquer permutação π, $d_r(\pi) \leq d^f_r(\pi) + 1$, $d_{rt}(\pi) \leq d^f_{rt}(\pi) + 1$, $d_{\bar{r}}(\pi) \leq d^f_{\bar{r}}(\pi) + 1$ e $d_{\bar{r}t}(\pi) \leq d^f_{\bar{r}t}(\pi) + 1$.*

Demonstração. Considere o problema SbFWR e uma permutação π qualquer. Observe que uma sequência de ordenação ótima para um problema ponderado também é uma sequência de ordenação válida para o problema não ponderado correspondente. Pelo Lema 3, existe uma sequência de ordenação ótima S que contém no máximo uma reversão completa. Portanto, todos os rearranjos dessa sequência tem custo maior ou igual a 1, exceto por no máximo uma reversão, e $|S| \leq d_r^f(\pi) + 1$. Pela definição do problema não ponderado, sabemos que o tamanho de qualquer sequência de ordenação de π é maior ou igual a $d_r(\pi)$ e, assim, $d_r(\pi) \leq |S| \leq d_r^f(\pi) + 1$.

A prova é similar para os outros problemas. □

Lema 5. *Para qualquer permutação π, $d_t(\pi) \leq d_t^f(\pi)$.*

Demonstração. Este resultado segue do fato de que qualquer transposição tem custo maior ou igual a 1. □

Lema 6. *Para qualquer permutação π, $d_r^f(\pi) \leq 2d_r(\pi)$, $d_{\bar{r}}^f(\pi) \leq 2d_{\bar{r}}(\pi)$, $d_t^f(\pi) \leq 3d_t(\pi)$, $d_{rt}^f(\pi) \leq 3d_{rt}(\pi)$ e $d_{\bar{r}t}^f(\pi) \leq 3d_{\bar{r}t}(\pi)$.*

Demonstração. Considere o problema SbFWR e uma permutação π qualquer. Observe que uma sequência de ordenação ótima para um problema não ponderado também é uma sequência de ordenação válida para o problema ponderado correspondente. Seja S uma sequência de ordenação ótima para o problema não ponderado. Como o custo máximo de uma reversão é igual a 2, temos que $f(S) \leq 2|S| = 2d_r(\pi)$. Assim, concluímos que $d_r^f(\pi) \leq 2d_r(\pi)$ para qualquer π.

A prova é similar para os outros problemas. Note que o custo máximo de uma transposição é igual a 3. □

Os próximos lemas apresentam fatores de aproximação para os problemas estudados utilizando os limitantes apresentados.

Lema 7. *Considerando os problemas SbFWR e SbFW$\bar{R}$, temos que uma α-aproximação, com $\alpha > 1$ constante, para a versão não ponderada do problema é uma 2α-aproximação assintótica para a versão ponderada por fragmentações.*

Demonstração. Considere o problema SbFWR e uma permutação π qualquer. Um algoritmo de α-aproximação para a versão não ponderada retorna uma sequência de ordenação S com tamanho $|S| \leq \alpha d_r(\pi)$ e custo $f(S) \leq 2\alpha d_r(\pi)$. Pelo Lema 4, temos

$$f(S) \leq 2\alpha d_r(\pi) \leq 2\alpha(d_r^f(\pi) + 1) \leq 2\alpha d_r^f(\pi) + 2\alpha.$$

Portanto, este algoritmo é uma 2α-aproximação assintótica para SbFWR. A prova é similar para o problema SbFW$\bar{\text{R}}$. □

Lema 8. *Considerando o problema SbFWT, temos que uma α-aproximação, com $\alpha > 1$ constante, para a versão não ponderada do problema é uma 3α-aproximação para a versão ponderada por fragmentações.*

Demonstração. Similar à prova do Lema 7. □

Tabela 3.1: Fatores de aproximação dos problemas ponderados pelo número de fragmentações usando os melhores algoritmos conhecidos para a versão não ponderada.

	Melhor Algoritmo	
	Problema Não Ponderado	**Problema Ponderado**
SbFWR	1.375-aproximação [5]	2.75-aproximação assintótica (Lema 7)
SbFW$\bar{\text{R}}$	Algoritmo exato [18]	2-aproximação assintótica (Lema 7)
SbFWT	1.375-aproximação [13]	4.125-aproximação (Lema 8)
SbFWRT	$(2.8334 + \epsilon)$-aproximação, para $\epsilon > 0$ [9, 32]	$(8.5002 + \epsilon)$-aproximação assintótica, para $\epsilon \geq 0$ (Lema 9)
SbFW$\bar{\text{R}}$T	2-aproximação [35]	6-aproximação assintótica (Lema 9)

Lema 9. *Considerando os problemas SbFWRT e SbFW$\bar{R}$T, temos que uma α-aproximação, com $\alpha > 1$ constante, para a versão não ponderada do problema é uma 3α-aproximação assintótica para a versão ponderada por fragmentações.*

Demonstração. Similar à prova do Lema 7. □

A partir dos resultados dos lemas 7, 8 e 9, a Tabela 3.1 apresenta os melhores fatores de aproximação obtidos ao adaptar algoritmos dos problemas não ponderados para a versão ponderada por fragmentações.

3.2 Algoritmos de Aproximação

Nesta seção, apresentamos limitantes e algoritmos de aproximação para os cinco problemas estudados neste capítulo. Esses algoritmos são separados em *básicos* e *gulosos*. Ambos os grupos de algoritmos utilizam o conceito de *breakpoints*, o qual é introduzido a seguir, e todos possuem fator de aproximação de 2.

3.2.1 Breakpoints

O conceito de *breakpoint* é muito comum em problemas de rearranjo de genomas, sendo utilizado para definição de limitantes e criação de algoritmos. De forma geral, existe um *breakpoint* entre um par de elementos consecutivos se esses elementos não devem ser consecutivos no final da ordenação. Essa definição sofre variações de acordo com o modelo de rearranjo considerado, como apresentado a seguir.

Definição 18. *Um* breakpoint de reversão sem sinais *existe entre um par de elementos consecutivos π_i e π_{i+1} se $|\pi_{i+1} - \pi_i| \neq 1$, para $1 \leq i < n$. O número de breakpoints de reversão sem sinais em uma permutação π é denotado por $b_r(\pi)$.*

Exemplo 7. *A permutação $\pi = (3\ 2\ 1\ 6\ 4\ 5)$ possui breakpoints de reversão sem sinais entre os pares $(\pi_3 = 1, \pi_4 = 6)$ e $(\pi_4 = 6, \pi_5 = 4)$. Neste exemplo, temos $b_r(\pi) = 2$.*

Definição 19. *Dada uma permutação π, a variação no número de breakpoints de reversão sem sinais causada por uma operação β é denotada por $\Delta b_r(\pi, \beta) = b_r(\pi) - b_r(\pi \circ \beta)$.*

Para permutações sem sinais, apenas a permutação identidade ι e a permutação reversa η não possuem *breakpoints* de reversão sem sinais. Esse tipo de *breakpoint* é utilizado em modelos de rearranjo com reversões sem sinais, tais como SbFWR e SbFWRT.

Definição 20. *Um* breakpoint de transposição *ou* breakpoint de reversão com sinais *existe entre um par de elementos consecutivos π_i e π_{i+1} se $\pi_{i+1} - \pi_i \neq 1$, para $1 \leq i < n$. Denotamos o número de breakpoints de transposição e breakpoints de reversão com sinais em uma permutação π como $b_t(\pi)$ e $b_{\bar{r}}(\pi)$, respectivamente.*

Exemplo 8. *A permutação $\pi = (+1\ +2\ +6\ +5\ -4\ -3)$ possui breakpoints de reversão com sinais entre os pares $(\pi_2 = +2, \pi_3 = +6)$, $(\pi_3 = +6, \pi_4 = +5)$ e $(\pi_4 = +5, \pi_5 = -4)$. Neste exemplo, temos $b_{\bar{r}}(\pi) = 3$.*

Definição 21. *Dada uma permutação π, a variação no número de breakpoints de transposição causada por uma operação β é denotada por $\Delta b_t(\pi, \beta) = b_t(\pi) - b_t(\pi \circ \beta)$. De forma similar, a notação $\Delta b_{\bar{r}}(\pi, \beta)$ é utilizada para denotar a variação no número de breakpoints de reversão com sinais causada por β.*

Para *breakpoints* de transposição ou reversão com sinais, apenas a permutação identidade ι e a permutação reversa com sinais $\bar{\eta}$ não possuem *breakpoints*. Esse tipo de *breakpoint* é utilizado em modelos de rearranjo com reversões com sinais ou com apenas transposições, tais como SbFWT, SbFW$\bar{\text{R}}$ e SbFW$\bar{\text{R}}$T.

Definição 22. *Considerando algum tipo de breakpoint, uma* strip *$\langle \pi_i, \pi_{i+1}, \ldots, \pi_j \rangle$, com $1 \leq i \leq j \leq n$, é uma subsequência maximal de π tal que não existe breakpoint entre quaisquer dois elementos consecutivos dessa subsequência.*

As *strips* de uma permutação π dependem do tipo de *breakpoint* utilizado, como apresentado nos exemplos a seguir.

Exemplo 9. *Considerando breakpoints de reversão sem sinais, a permutação $\pi =$* (1 2 3 6 5 4) *possui as strips $\langle 1, 2, 3 \rangle$ e $\langle 6, 5, 4 \rangle$.*

Exemplo 10. *Considerando breakpoints de transposição, a permutação $\pi =$* (1 2 3 6 5 4) *possui as strips $\langle 1, 2, 3 \rangle$, $\langle 6 \rangle$, $\langle 5 \rangle$ e $\langle 4 \rangle$.*

Um *singleton* é uma *strip* com apenas um elemento. Para permutações sem sinais, uma *strip* $\langle \pi_i, \pi_{i+1}, \ldots, \pi_j \rangle$, com $1 \leq i \leq j \leq n$, é dita *crescente* se $\pi_{k+1} > \pi_k$, para todo $i \leq k < j$, e é dita *decrescente* caso contrário. Por definição, um *singleton* é uma *strip* crescente. Ao considerar *breakpoints* de transposição, só existem *strips* crescentes em uma permutação sem sinais. Para permutações com sinais, uma *strip* é *positiva* se todos os seus elementos são positivos ou *negativa* se todos os seus elementos são negativos.

3.2.2 Limitantes Inferiores

Usando o conceito de *breakpoints*, apresentamos novos limitantes inferiores para os problemas estudados. Para isso, os próximos lemas dão limitantes na variação do número de *breakpoints* causada pela aplicação de uma operação em uma permutação π.

Lema 10. *Para qualquer permutação π e reversão sem sinais ρ, $-f(\rho) \leq \Delta b_r(\pi, \rho) \leq f(\rho)$.*

Demonstração. Seja $\rho = \rho(i, j)$ e $\pi' = \pi \circ \rho(i, j)$. Para todo $i \leq k < j$, $|\pi_{k+1} - \pi_k| = |\pi'_{j-((k+1)-i)} - \pi'_{j-(k-i)}|$ pois $\pi'_{j-(k-i)} = \pi_k$ e $\pi'_{j-((k+1)-i)} = \pi_{k+1}$. Portanto, uma reversão sem sinais $\rho(i, j)$ não altera o número de *breakpoints* no segmento de i até j.

Como os segmentos de 1 até $i-1$ e de $j+1$ até n não são alterados por $\rho(i, j)$, a operação $\rho(i, j)$ só pode adicionar ou remover *breakpoints* entre os pares de elementos de π que não são adjacentes em π', ou seja, os pares de elementos de π' que sofreram fragmentação. Portanto, temos que $-f(\rho) \leq \Delta b_r(\pi, \rho) \leq f(\rho)$. □

Lema 11. *Para qualquer permutação π e reversão com sinais $\bar{\rho}$, $-f(\bar{\rho}) \leq \Delta b_{\bar{r}}(\pi, \bar{\rho}) \leq f(\bar{\rho})$.*

Demonstração. Seja $\bar{\rho} = \bar{\rho}(i, j)$ e $\pi' = \pi \circ \bar{\rho}(i.j)$. Para todo $i \leq k < j$, $\pi_{k+1} - \pi_k = -\pi_k - (-\pi_{k+1}) = \pi'_{j-(k-i)} - \pi'_{j-((k+1)-i)}$ pois $\pi'_{j-(k-i)} = -\pi_k$ e $\pi'_{j-((k+1)-i)} = -\pi_{k+1}$. Portanto, uma reversão com sinais $\rho(i, j)$ não altera o número de *breakpoints* no intervalo das posições de i até j.

Por um argumento similar ao utilizado na prova do Lema 10, a operação $\bar{\rho}(i, j)$ só pode adicionar ou remover *breakpoints* entre os pares de elementos de π' que sofreram fragmentação. Portanto, temos que $-f(\bar{\rho}) \leq \Delta b_r(\pi, \bar{\rho}) \leq f(\bar{\rho})$. □

Lema 12. *Para qualquer permutação π e transposição τ, $-f(\tau) \leq \Delta b_t(\pi, \tau) \leq f(\tau)$.*

Demonstração. Similar à prova dos lemas 10 e 11. □

Com esses resultados, o Lema 13 apresenta um limitante inferior para as distâncias $d_r^f(\pi)$ e $d_{\bar{r}}^f(\pi)$.

Lema 13. *Para qualquer permutação π, $b_r(\pi) \leq d_r^f(\pi)$ e $b_{\bar{r}}(\pi) \leq d_{\bar{r}}^f(\pi)$.*

Demonstração. Considere o problema SbFWR, as únicas permutações que não possuem *breakpoints* são a identidade ι e a reversa η. Note que a distância entre ι e η é igual a 0, já que $\eta \circ \rho(1, n) = \iota$. Portanto, uma sequência de ordenação deve remover todos os *breakpoints* de π. Pelo Lema 10, o custo médio para remoção de um *breakpoint* é maior ou igual a 1 e, portanto, qualquer sequência de ordenação S tem custo $f(S) \geq b_r(\pi)$.

A prova é similar para o problema SbFW$\bar{\text{R}}$. □

Usando argumentos similares ao do Lema 13, podemos obter os seguintes limitantes inferiores para os outros três problemas.

Lema 14. *Para qualquer permutação π, $b_t(\pi) \leq d_t^f(\pi)$.*

Lema 15. *Para qualquer permutação π, $b_r(\pi) \leq d_{rt}^f(\pi)$ e $b_{\bar{r}}(\pi) \leq d_{\bar{r}t}^f(\pi)$.*

Algoritmo 1: Algoritmo de 2-aproximação para os problemas SbFWR e SbFWRT

Entrada: permutação sem sinais π e o seu tamanho n
Saída: custo para ordenar π

```
2-R(π, n) :
    d ← 0
    enquanto π ≠ ι faça
        Seja π_ℓ = k o elemento com maior valor absoluto fora de posição
        Seja s = ⟨π_i, ..., π_j⟩ a strip contendo π_ℓ
        se ℓ = j então
            π ← π ∘ ρ(1, j) ∘ ρ(1, k)
            d ← d + f(ρ(1, j)) + f(ρ(1, k))
        senão
            π ← π ∘ ρ(i, k)
            d ← d + f(ρ(i, k))
    retorne d
```

3.2.3 Algoritmos Básicos de 2-Aproximação

Os primeiros algoritmos de 2-aproximação utilizam o seguinte procedimento para ordenar uma permutação. Enquanto a permutação não está ordenada, a estratégia desses algoritmos é encontrar a *strip* contendo o elemento com maior valor absoluto fora de posição e colocá-la no seu lugar correto, utilizando uma sequência de operações de custo menor ou igual a 2.

Os algoritmos 1, 2 e 3 mostram o pseudocódigo dos algoritmos 2-R, 2-$\bar{\text{R}}$ e 2-T, respectivamente. Como uma permutação tem no máximo n *strips*, os três algoritmos executam no máximo $n-1$ iterações. Em cada iteração, eles gastam tempo linear para encontrar e mover a *strip* do elemento com maior valor absoluto fora de posição. Portanto, 2-R, 2-$\bar{\text{R}}$ e 2-T possuem complexidade de tempo $O(n^2)$.

Os próximos lemas apresentam limitantes superiores para as sequências encontradas por esses algoritmos.

Lema 16. *Para qualquer permutação π, o algoritmo 2-R encontra uma sequência de ordenação S tal que $f(S) \leq 2b_r(\pi)$.*

Demonstração. A cada iteração, enquanto a permutação não está ordenada, o algoritmo executa o seguinte procedimento. Seja $|\pi_\ell| = k$ o elemento com maior valor absoluto fora de posição. Seja $s = \langle \pi_i, \ldots, \pi_j \rangle$ a *strip* que contém o elemento π_ℓ (note que $\ell = i$ ou $\ell = j$). O algoritmo move s para a sua posição correta de acordo com os seguintes casos.

1. Se $\ell = i$, então a reversão $\rho(i, k)$ de custo 2 move a *strip* s para a sua posição correta, como mostrado a seguir. Se $k < n$, então um *breakpoint* entre as posições k e $k+1$ é removido, já que $\pi_i = k$.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \ldots \ \pi_k \ k+1 \ k+2 \ \ldots \ n) \\ \pi \circ \rho(i,k) &= (\pi_1 \ \ldots \ \underline{\pi_k \ \ldots \ \pi_j \ \ldots \ \pi_i} \ k+1 \ k+2 \ \ldots \ n) \end{aligned} \tag{3.13}$$

Algoritmo 2: Algoritmo de 2-aproximação para os problemas SbFW$\bar{\text{R}}$ e SbFW$\bar{\text{R}}$T

Entrada: permutação com sinais π e o seu tamanho n
Saída: custo para ordenar π

```
1  2-R̄(π, n) :
2      d ← 0
3      enquanto π ≠ ι faça
4          Seja |π_ℓ| = k o elemento com maior valor absoluto, tal que ℓ ≠ k ou π_ℓ < 0
5          Seja s = ⟨π_i, ..., π_j⟩ a strip contendo π_ℓ
6          se ℓ = j então
7              π ← π ∘ ρ̄(1, j) ∘ ρ̄(1, k)
8              d ← d + f(ρ̄(1, j)) + f(ρ̄(1, k))
9          senão
10             π ← π ∘ ρ̄(i, k)
11             d ← d + f(ρ̄(i, k))
12     retorne d
```

Algoritmo 3: Algoritmo de 2-aproximação para o problema SbFWT

Entrada: permutação sem sinais π e o seu tamanho n
Saída: custo para ordenar π

```
1  2-T(π, n) :
2      d ← 0
3      enquanto π ≠ ι faça
4          Seja π_ℓ = k o elemento com maior valor absoluto fora de posição
5          Seja s = ⟨π_i, ..., π_j⟩ a strip contendo π_ℓ
6          π ← π ∘ τ(1, j + 1, k + 1)
7          d ← d + f(τ(1, j + 1, k + 1))
8      retorne d
```

2. Se $\ell = j$ e $j \neq i$, então a sequência $S = \langle \rho(1, j), \rho(1, k) \rangle$ de custo 2 move a *strip* s para a sua posição correta, como mostrado a seguir. Se $k < n$, então um *breakpoint* entre as posições k e $k + 1$ é removido, já que $\pi_j = k$.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \ldots \ \pi_k \ k+1 \ k+2 \ \ldots \ n) \\ \pi \circ \rho(1, j) &= (\underline{\pi_j \ \ldots \ \pi_i \ \ldots \ \pi_1} \ \ldots \ \pi_k \ k+1 \ k+2 \ \ldots \ n) \\ \pi \circ \rho(1, k) &= (\underline{\pi_k \ \ldots \ \pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j} \ k+1 \ k+2 \ \ldots \ n) \end{aligned} \tag{3.14}$$

Se $\pi_n \neq n$, então a primeira iteração do algoritmo colocará a *strip* contendo n na sua posição correta com custo 1, mas não necessariamente irá remover *breakpoints* da permutação. Note que isso será feito com uma reversão de prefixo e uma reversão completa, se n está contido em uma *strip* crescente, ou com uma reversão de sufixo, caso contrário. Todas as outras iterações do algoritmo removem pelo menos um *breakpoint* com um custo de no máximo 2. A última iteração remove dois *breakpoints* com custo 2, se a última *strip*

fora de posição não contém o elemento 1, ou remove um *breakpoint* com custo 1, caso contrário. Dessa forma, o custo médio para remoção de um *breakpoint* é menor ou igual a 2 e, consequentemente, $f(S) \leq 2b_r(\pi)$. □

Teorema 1. *O algoritmo 2-R é uma 2-aproximação para os problemas SbFWR e SbFWRT.*

Demonstração. Segue diretamente dos lemas 13, 15 e 16. □

Lema 17. *Para qualquer permutação π, o algoritmo 2-$\bar{R}$ encontra uma sequência de ordenação S tal que $f(S) \leq 2b_{\bar{r}}(\pi)$.*

Demonstração. Similar à prova do Lema 16. Notamos que um elemento $\pi_k = -k$ é considerado como fora da sua posição correta. □

Teorema 2. *O algoritmo 2-$\bar{R}$ é uma 2-aproximação para os problemas SbFW$\bar{R}$ e SbFW$\bar{R}$T.*

Demonstração. Segue diretamente dos lemas 13, 15 e 17. □

Lema 18. *Para qualquer permutação π, o algoritmo 2-T encontra uma sequência de ordenação S tal que $f(S) \leq 2b_t(\pi)$.*

Demonstração. A cada iteração, enquanto a permutação não está ordenada, o algoritmo executa o seguinte procedimento. Seja $|\pi_\ell| = k$ o elemento com maior valor absoluto fora de posição. Seja $s = \langle \pi_i, \ldots, \pi_j \rangle$ a *strip* que contém o elemento π_ℓ. Notamos que $\pi_\ell = \pi_j$, já que só existem *strips* crescentes ao considerar *breakpoints* de transposição. O algoritmo move s para a sua posição correta com uma transposição $\tau(1, j+1, k+1)$ de custo no máximo 2, como mostrado a seguir.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_k \ k+1 \ k+2 \ \ldots \ n) \\ \pi \circ \tau(1, j+1, k+1) &= (\underline{\pi_{j+1} \ \ldots \ \pi_k} \ \underline{\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j} \ k+1 \ k+2 \ \ldots \ n) \end{aligned} \tag{3.15}$$

Se $\pi_n \neq n$, então a primeira iteração do algoritmo colocará a *strip* contendo n na sua posição correta com uma transposição completa de custo 1, mas não necessariamente irá remover *breakpoints* da permutação. Todas as outras iterações do algoritmo removem pelo menos um *breakpoint* com custo 2. Na última iteração do algoritmo, π atende a configuração $(\pi_1 \ \ldots \ \pi_k \ \pi_i \ \ldots \ \pi_j \ j{+}1 \ \ldots \ n)$, tal que $\pi_k = j$ e $s = \langle \pi_1 \ \ldots \ \pi_k \rangle$ é uma *strip*. Portanto, a transposição $\tau(1, k+1, j+1)$ remove os dois últimos *breakpoints* da permutação com custo 2. Dessa forma, o custo médio para remoção de um *breakpoint* é menor ou igual a 2 e, consequentemente, $f(S) \leq 2b_t(\pi)$. □

Teorema 3. *O algoritmo 2-T é uma 2-aproximação para o problema SbFWT.*

Demonstração. Segue diretamente dos lemas 14 e 18. □

3.2.4 Algoritmos Gulosos de 2-Aproximação

O segundo grupo de aproximações é formado por algoritmos gulosos, os quais funcionam da seguinte forma. Enquanto existem *breakpoints* na permutação π, a estratégia desses algoritmos é aplicar a operação com melhor razão "*breakpoints* removidos : custo da operação". Para os problemas SbFWR e SbFW$\bar{\text{R}}$, os algoritmos podem chegar a um estado em que não existem reversões que possam remover *breakpoints*. Nesse caso, o algoritmo aplica uma reversão de custo 1 a fim de garantir que exista operação com razão 1 na próxima iteração. De forma geral, a permutação π está ordenada no fim desse procedimento. No entanto, para problemas envolvendo reversões, a permutação π pode ter sido transformada na permutação reversa, a qual pode ser ordenada com uma reversão completa de custo 0.

Os algoritmos 4 e 5 apresentam pseudocódigos genéricos para esses algoritmos. Eles são nomeados 2-Rg, 2-$\bar{\text{R}}$g, 2-Tg, 2-RTg e 2-$\bar{\text{R}}$Tg para os problemas SbFWR, SbFW$\bar{\text{R}}$, SbFWT, SbFWRT e SbFW$\bar{\text{R}}$T, respectivamente. Como o número de *breakpoints* em uma permutação π é menor ou igual a n, esses algoritmos executam $O(n)$ iterações. Considerando 2-Rg e 2-$\bar{\text{R}}$g, cada iteração gasta tempo $O(n^2)$ para escolher uma reversão dentre todas as possíveis e, portanto, possuem complexidade de tempo de $O(n^3)$. Considerando 2-Tg, 2-RTg e 2-$\bar{\text{R}}$Tg, cada iteração gasta tempo $O(n^3)$ para escolher uma operação dentre todas as possíveis e, portanto, possuem complexidade de tempo de $O(n^4)$. Observe que podemos calcular a variação no número de *breakpoints* causado por uma operação β em tempo constante, já que os únicos pares de posições onde um *breakpoint* pode ser adicionado ou removido são aqueles onde β causa fragmentação.

Os próximos teoremas demonstram que o fator de aproximação desses algoritmos também é 2.

Teorema 4. *2-Rg e 2-$\bar{\text{R}}$g são algoritmos de 2-aproximação para SbFWR e SbFW$\bar{\text{R}}$, respectivamente.*

Demonstração. Considere o problema SbFWR. Precisamos demonstrar que sempre existe uma sequência que remove pelo menos um *breakpoint* com custo de no máximo 2. Assim, o algoritmo 2-Rg encontra uma sequência de ordenação com custo menor ou igual a duas vezes o número de *breakpoints* da permutação e, portanto, o algoritmo é uma 2-aproximação (Lema 13).

Para qualquer permutação π, podemos dividir a prova nos seguintes casos:

1. π possui pelo menos uma *strip* de tipo diferente das outras ou π possui um *singleton*. Seja $\langle \pi_i, \ldots, \pi_j \rangle$ a primeira *strip* de π tal que existe outra *strip* $\langle \pi_{i'}, \ldots, \pi_{j'} \rangle$ de tipo diferente, com $|\pi_i - \pi_{i'}| = 1$ ou $|\pi_j - \pi_{j'}| = 1$. Note que sempre poderemos achar tais *strips* nesse caso. Dividimos a prova nos seguintes subcasos:

   (a) Se $|\pi_j - \pi_{j'}| = 1$, então $\rho(j+1, j')$ remove um *breakpoint*.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_{i'} \ \ldots \ \pi_{j'} \ \ldots \ \pi_n) \\ \pi \circ \rho(j+1, j') &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \underline{\pi_{j'} \ \ldots \ \pi_{i'} \ \ldots \ \pi_{j+1}} \ \ldots \ \pi_n) \end{aligned} \tag{3.16}$$

Algoritmo 4: Algoritmo genérico para 2-Rg e 2-$\bar{\text{R}}$g

Entrada: modelo M, tipo de *breakpoint* b, permutação π e o seu tamanho n
Saída: custo para ordenar π

```
1  sort(M, b, π, n) :
2      d ← 0
3      enquanto b(π) > 0 faça
4          Seja β uma operação de M, tal que Δb(π,β)/f(β) ≥ Δb(π,β')/f(β') para todo β' ∈ M
5          se Δb(π, β) > 0 então
6              π ← π ∘ β
7              d ← d + f(β)
8          senão
9              Seja ⟨π_1, π_2, ..., π_j⟩ a primeira strip de π
10             se |π_j| = 1 então
11                 Seja k o elemento com valor absoluto |π_1| + 1
12                 Seja i o início da strip contendo k
13                 π ← ρ(i, n)
14                 d ← d + f(ρ(i, n))
15             senão se π_j = n então
16                 Seja k o elemento com valor absoluto π_1 − 1
17                 Seja i o início da strip contendo k
18                 π ← ρ(i, n)
19                 d ← d + f(ρ(i, n))
20             senão
21                 π ← ρ(1, j)
22                 d ← d + f(ρ(1, j))
23     se π ≠ ι então
24         π ← π ∘ ρ(1, n)
25     retorne d
```

(b) Se $|\pi_i - \pi_{i'}| = 1$, então $\rho(i, i' - 1)$ remove um *breakpoint*.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_i \ \ldots \ \pi_j \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_{j'} \ \ldots \ \pi_n) \\ \pi \circ \rho(i, i' - 1) &= (\pi_1 \ \ldots \ \underline{\pi_{i'-1} \ \ldots \ \pi_j \ \ldots \ \pi_i} \ \pi_{i'} \ \ldots \ \pi_{j'} \ \ldots \ \pi_n) \end{aligned} \tag{3.17}$$

2. Todas as *strips* de π são do mesmo tipo. Nesse caso, não existe reversão que remove *breakpoints* de π. Assim, o algoritmo transforma π em uma permutação π' com custo 1, de forma que exista reversão ρ que satisfaz $\Delta b_r(\pi', \rho) = f(\rho) = 1$. Dessa forma, o algoritmo usa uma sequência de custo 2 para remover um *breakpoint* de π. Seja $s = \langle \pi_1, \ldots, \pi_j \rangle$ a primeira *strip* de π. Dividimos a prova nos seguintes subcasos:

   (a) Se s é crescente e $\pi_j < n$, então a reversão $\rho(1, j)$ transforma π na permutação π'. Na próxima iteração, a reversão $\rho(1, i' - 1)$, onde $\pi'_{i'} = \pi_j + 1$, remove um

Algoritmo 5: Algoritmo genérico para 2-Tg, 2-RTg e 2-$\bar{\text{R}}$Tg

Entrada: modelo M, tipo de *breakpoint* b, permutação π e o seu tamanho n
Saída: custo para ordenar π

```
1 sort(M, b, π, n) :
2     d ← 0
3     enquanto b(π) > 0 faça
4         Seja β uma operação de M, tal que Δb(π,β)/f(β) ≥ Δb(π,β')/f(β') para todo β' ∈ M
5         π ← π ∘ β
6         d ← d + f(β)
7     se π ≠ ι então
8         π ← π ∘ ρ(1, n)
9     retorne d
```

breakpoint de π' com custo 1.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_n) \\ \pi' = \pi \circ \rho(1,j) &= (\underline{\pi_j \ \ldots \ \pi_1} \ \pi_{j+1} \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_n) \\ \pi' \circ \rho(1,i'-1) &= (\underline{\pi_{i'-1} \ \ldots \ \pi_{j+1} \ \pi_1 \ \ldots \ \pi_j} \ \pi_{i'} \ \ldots \ \pi_n) \end{aligned} \tag{3.18}$$

(b) Se s é descrescente e $\pi_j > 1$, então a reversão $\rho(1,j)$ transforma π na permutação π'. Na próxima iteração, a reversão $\rho(1,i'-1)$, onde $\pi'_{i'} = \pi_j - 1$, remove um *breakpoint* de π' com custo 1.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_n) \\ \pi' = \pi \circ \rho(1,j) &= (\underline{\pi_j \ \ldots \ \pi_1} \ \pi_{j+1} \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_n) \\ \pi' \circ \rho(1,i'-1) &= (\underline{\pi_{i'-1} \ \ldots \ \pi_{j+1} \ \pi_1 \ \ldots \ \pi_j} \ \pi_{i'} \ \ldots \ \pi_n) \end{aligned} \tag{3.19}$$

(c) Se s é crescente e $\pi_j = n$, então a reversão $\rho(i',n)$ transforma π na permutação π', tal que $\langle \pi_{i'}, \ldots, \pi_{j'} \rangle$ é a *strip* que contém $\pi_1 - 1$. Na próxima iteração, a reversão $\rho(1, n-(j'-i'+1))$ remove um *breakpoint* de π' com custo 1.

$$\begin{aligned} \pi &= (\pi_1 \ \ldots \ \pi_j \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_{j'} \ \pi_{j'+1} \ \ldots \ \pi_n) \\ \pi' = \pi \circ \rho(i',n) &= (\pi_1 \ \ldots \ \pi_j \ \ldots \ \pi_{i'-1} \ \underline{\pi_n \ \ldots \ \pi_{j'+1} \ \pi_{j'} \ \ldots \ \pi_{i'}}) \\ \pi' \circ \rho(1, n-(j'-i'+1)) &= (\underline{\pi_{j'+1} \ \ldots \ \pi_n \ \pi_{i'-1} \ \ldots \ \pi_j \ \ldots \ \pi_1} \ \pi_{j'} \ \ldots \ \pi_{i'}) \end{aligned} \tag{3.20}$$

(d) Se s é decrescente e $\pi_j = 1$, então a reversão $\rho(i',n)$ transforma π na permutação π', tal que $\langle \pi_{i'}, \ldots, \pi_{j'} \rangle$ é a *strip* que contém $\pi_1 + 1$. Na próxima iteração,

a reversão $\rho(1, n-(j'-i'+1))$ remove um *breakpoint* de π' com custo 1.

$$\begin{aligned}\pi &= (\pi_1 \ \ldots \ \pi_j \ \ldots \ \pi_{i'-1} \ \pi_{i'} \ \ldots \ \pi_{j'} \ \pi_{j'+1} \ \ldots \ \pi_n)\\ \pi' = \pi \circ \rho(i', n) &= (\pi_1 \ \ldots \ \pi_j \ \ldots \ \pi_{i'-1} \ \underline{\pi_n \ \ldots \ \pi_{j'+1} \ \pi_{j'} \ \ldots \ \pi_{i'}})\\ \pi' \circ \rho(1, n-(j'-i'+1)) &= (\underline{\pi_{j'+1} \ \ldots \ \pi_n \ \pi_{i'-1} \ \ldots \ \pi_j \ \ldots \ \pi_1} \ \pi_{j'} \ \ldots \ \pi_{i'})\end{aligned} \tag{3.21}$$

A prova é análoga para o problema SbFW$\bar{\text{R}}$. □

Teorema 5. 2-*Tg é um algoritmo de* 2-*aproximação para SbFWT.*

Demonstração. A cada iteração do algoritmo 2-Tg, existe transposição de custo 2 que remove um *breakpoint* da seguinte forma. Seja $\langle \pi_1, \ldots, \pi_j \rangle$ a primeira *strip* de π. Dividiremos a demonstração em dois casos.

1. Se $\pi_j < n$, então a transposição $\tau(1, j+1, k)$, onde $\pi_k = \pi_j + 1$, remove um *breakpoint* de π, como mostrado a seguir.

$$\begin{aligned}\pi &= (\pi_1 \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1} \ \pi_k \ \ldots \ \pi_n)\\ \pi \circ \tau(1, j+1, k) &= (\underline{\pi_{j+1} \ \ldots \ \pi_{k-1}} \ \underline{\pi_1 \ \ldots \ \pi_j} \ \pi_k \ \ldots \ \pi_n)\end{aligned} \tag{3.22}$$

2. Se $\pi_j = n$, então a transposição $\tau(1, j+1, k+1)$, onde $\pi_k = \pi_1 - 1$, remove um *breakpoint* de π, como mostrado a seguir.

$$\begin{aligned}\pi &= (\pi_1 \ \ldots \ \pi_j \ \pi_{j+1} \ \ldots \ \pi_k \ \pi_{k+1} \ \ldots \ \pi_n)\\ \pi \circ \tau(1, j+1, k+1) &= (\underline{\pi_{j+1} \ \ldots \ \pi_k} \ \underline{\pi_1 \ \ldots \ \pi_j} \ \pi_{k+1} \ \ldots \ \pi_n)\end{aligned} \tag{3.23}$$

□

Teorema 6. 2-*RTg e* 2-$\bar{R}$*Tg são algoritmos de* 2-*aproximação para SbFWRT e SbFW*$\bar{R}$*T, respectivamente.*

Demonstração. Esta prova é similar às provas dos teoremas 4 e 5, já que esses algoritmos usam reversões e transposições. Notamos que no único caso em que não existem reversões que removem um *breakpoint*, todas as *strips* são do mesmo tipo e, portanto, existe transposição de custo 2 que remove pelo menos um *breakpoint* da permutação. □

3.2.5 Resultados Experimentais

Todos os algoritmos apresentados nas seções 3.2.3 e 3.2.4 foram implementados na linguagem de programação C e executados usando os seguintes conjuntos de instâncias:

- UB_n: 1000 permutações sem sinais aleatórias de tamanho n, sendo que cada permutação desse conjunto possui número máximo de *breakpoints*;
- SB_n: 1000 permutações com sinais aleatórias de tamanho n, sendo que cada permutação desse conjunto possui número máximo de *breakpoints*;

- R_n^M: 1000 permutações de tamanho n geradas a partir da aplicação de $n/5$ operações aleatórias na permutação identidade ι, sendo que cada operação utilizada pertence a M. As permutações do conjunto possuem sinais quando M é igual a $\bar{r}$ ou $\bar{r}t$.

Para cada tipo de instância, foram utilizados valores de $n \in \{10, 15, 20, \ldots, 495, 500\}$ e $M \in \{r, t, rt, \bar{r}, \bar{r}t\}$. Notamos que $n/5$ reversões aleatórias ou $n/5$ transposições aleatórias aplicadas em ι adicionam no máximo $2n/5$ *breakpoints* ou $3n/5$ *breakpoints*, respectivamente. Portanto, considerando os limitantes apresentados na Seção 3.2.2, o limitante inferior para a distância das permutações pertencentes a R_n^M é significativamente menor do que o das permutações dos conjuntos UB_n ou SB_n.

Além dos algoritmos apresentados nesta seção, os experimentos também utilizam cinco algoritmos da abordagem tradicional que foram adaptados para a versão ponderada por fragmentações. Para adaptar esses algoritmos, apenas aplicamos a função de custo na sequência de ordenação retornada por cada um deles. Os cinco algoritmos adaptados foram:

- A-R: uma $(1.4193+\epsilon)$-aproximação para a Ordenação de Permutações por Reversões sem Sinais, a qual consiste no algoritmo de decomposição de ciclos do grafo de *breakpoints* [25] seguido pelo algoritmo exato para reversões com sinais [18];

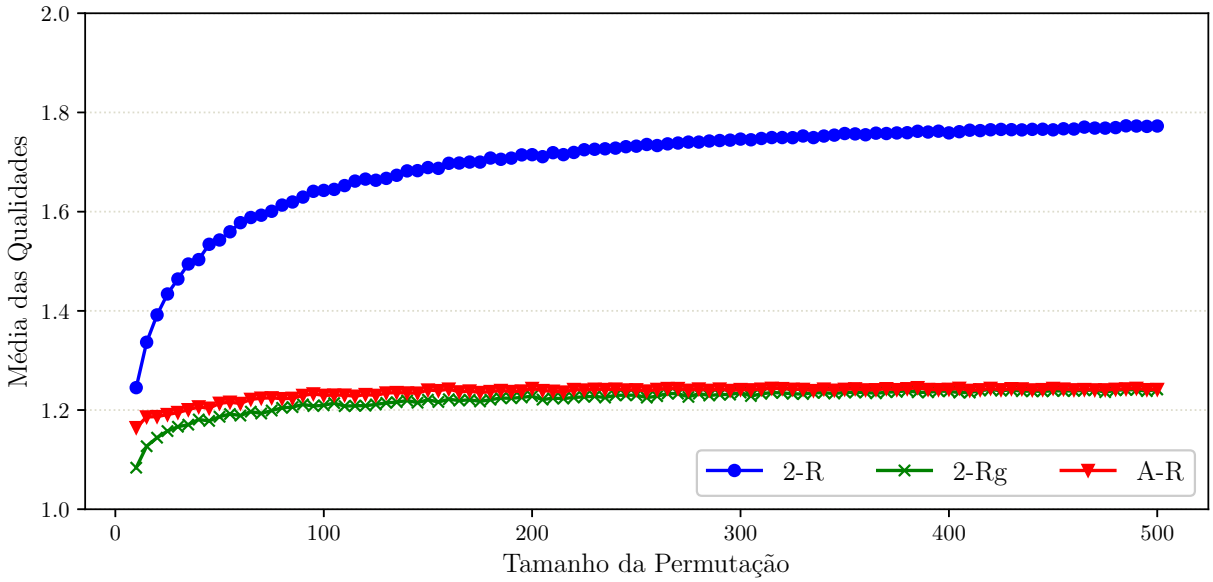
- A-$\bar{\mathrm{R}}$: algoritmo exato para a Ordenação de Permutações por Reversões com Sinais [18];

- A-T: uma 1.5-aproximação para a Ordenação de Permutações por Transposições [3]. Para esse problema, também foram feitos testes com uma 1.375-aproximação [13], porém o algoritmo de 1.375-aproximação obteve resultados práticos inferiores aos resultados do algoritmo de 1.5-aproximação [3];

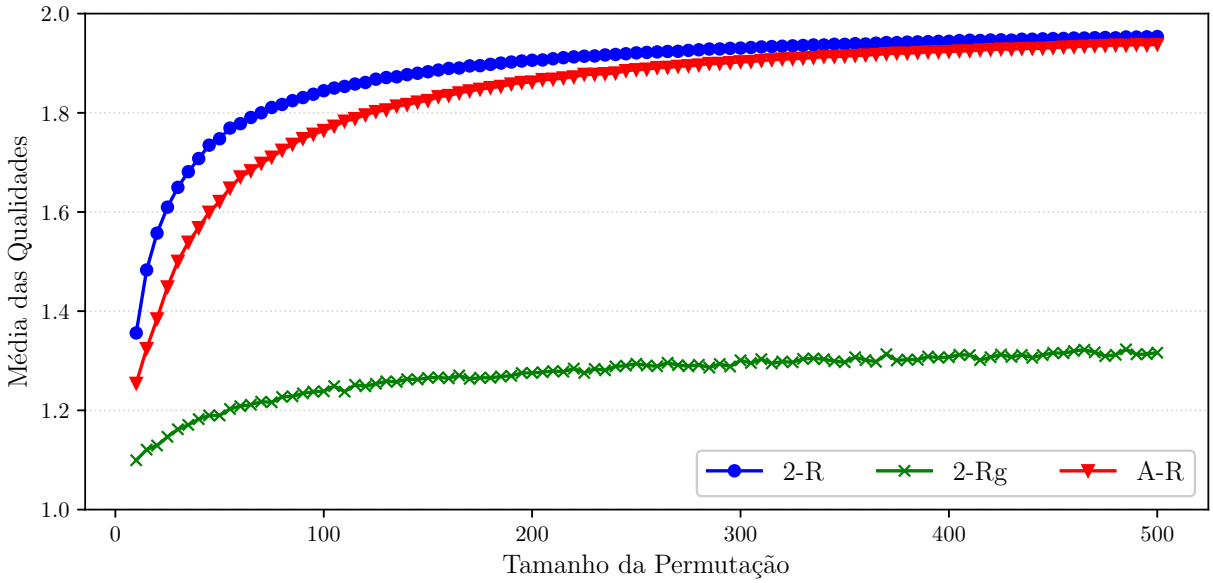
- A-RT: uma $2k$-aproximação, com $k = 1.4193 + \epsilon$ [25], para a Ordenação de Permutações por Reversões sem Sinais e Transposições [32];

- A-$\bar{\mathrm{R}}$T: uma 2-aproximação para a Ordenação de Permutações por Reversões com Sinais e Transposições [32]. Para esse problema, também foram feitos testes com outro algoritmo de 2-aproximação [35]. O algoritmo escolhido foi o que obteve melhores resultados práticos.

Os resultados dos experimentos são apresentados nas figuras 3.1 a 3.5. Para calcular a qualidade dos algoritmos, utilizamos os limitantes inferiores apresentados nos lemas 13, 14 e 15. Os algoritmos gulosos apresentaram os melhores resultados práticos em todos os cinco problemas. Em média, a qualidade para esses algoritmos foi significativamente melhor do que 2 (fator teórico). Considerando as instâncias do tipo R_n^M, os algoritmos adaptados obtiveram resultados próximos aos dos algoritmos gulosos. De forma geral, os algoritmos adaptados obtiveram resultados melhores do que os algoritmos básicos. Além disso, a média das qualidades para os algoritmos básicos foi próximo do fator teórico quando n é próximo de 500. Na maioria dos casos, observamos que a média das qualidades piora à medida que o tamanho da permutação cresce.

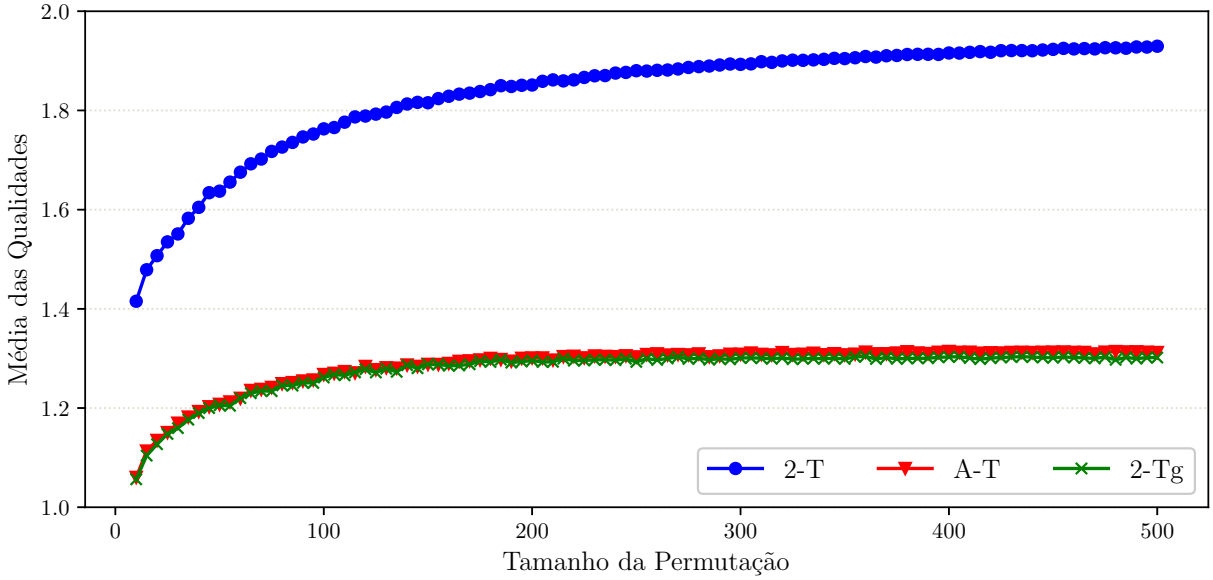


(a) Instâncias R_n^r

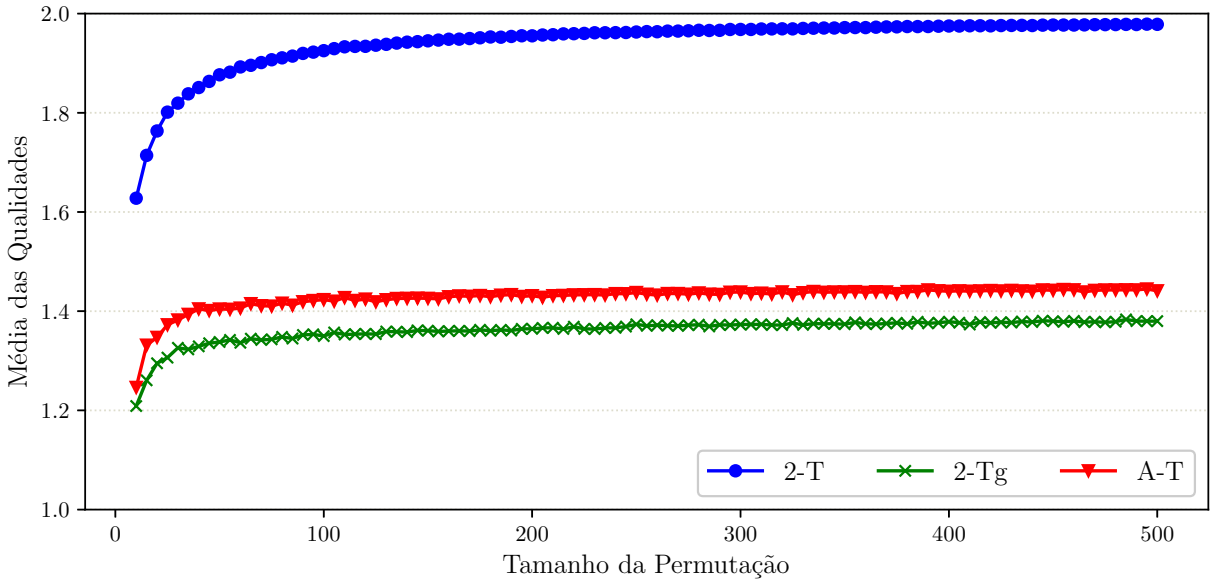


(b) Instâncias UB_n

Figura 3.1: Média das qualidades para os algoritmos 2-R, 2-Rg e A-R.

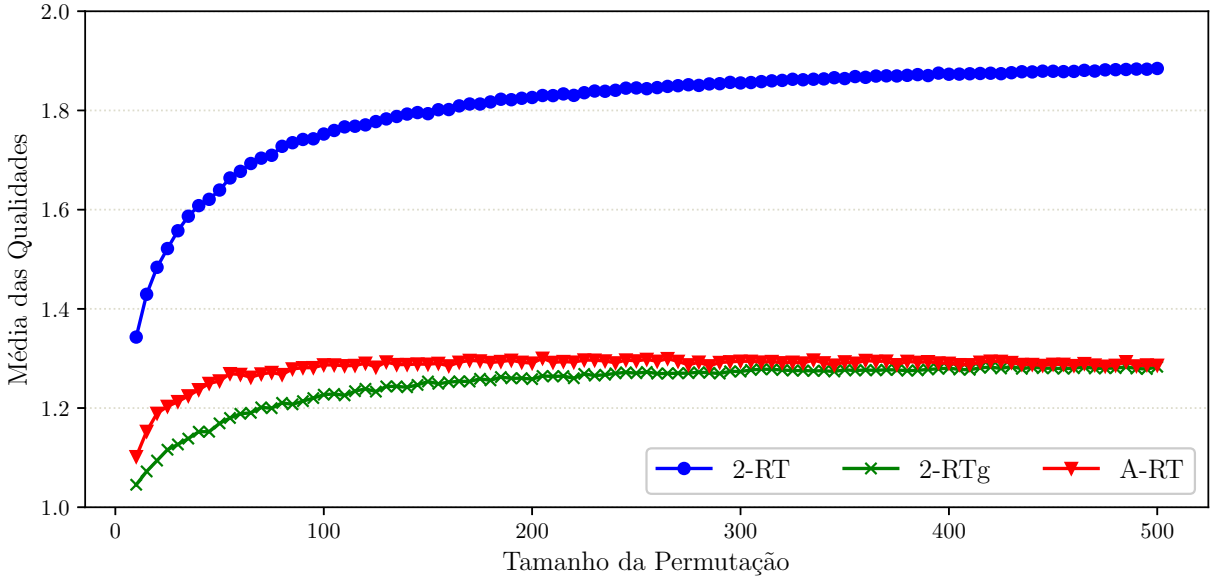


(a) Instâncias R_n^t

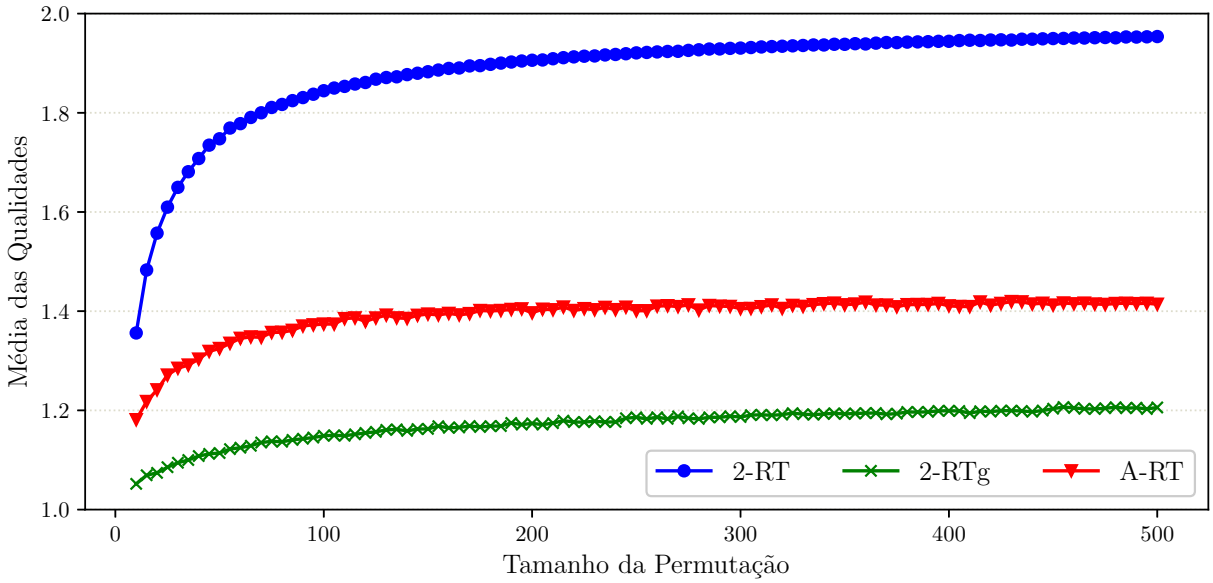


(b) Instâncias UB_n

Figura 3.2: Média das qualidades para os algoritmos 2-T, 2-Tg e A-T.

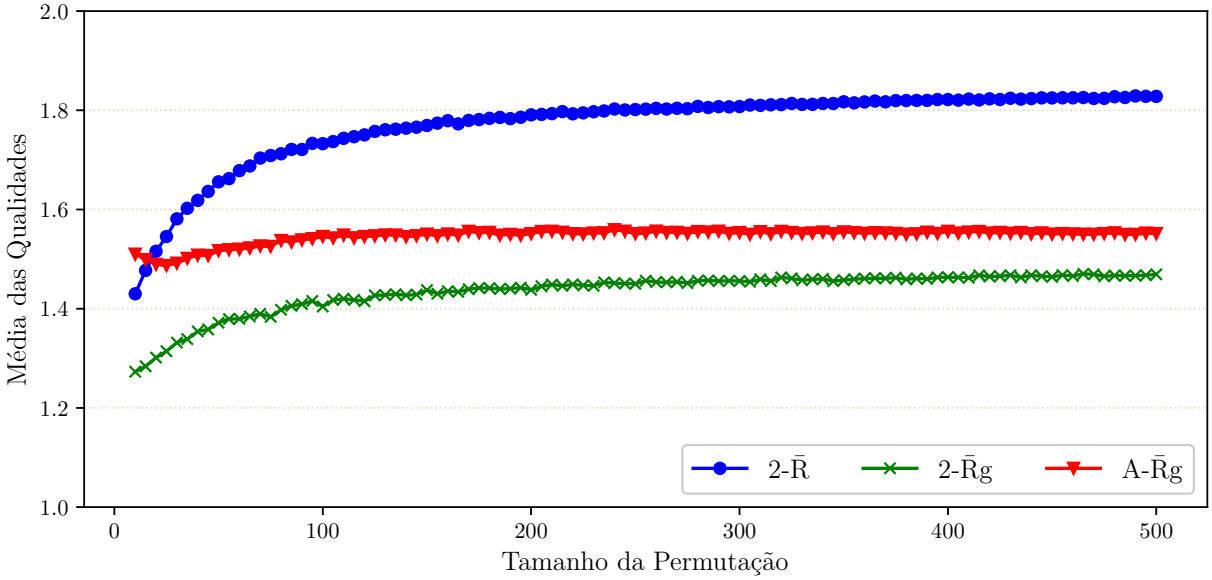


(a) Instâncias R_n^{rt}

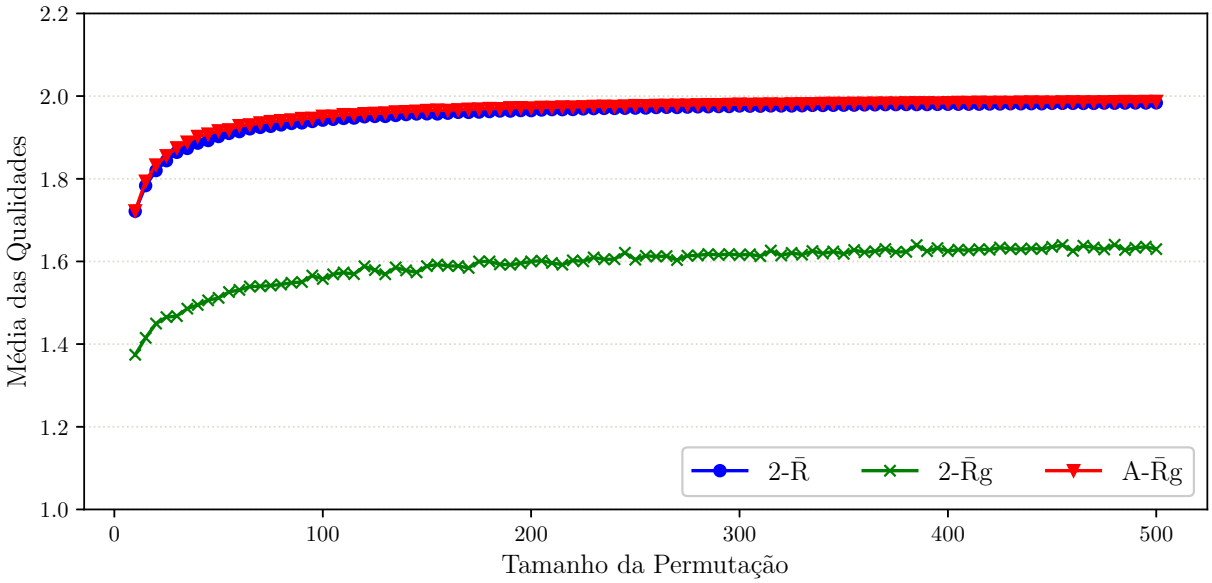


(b) Instâncias UB_n

Figura 3.3: Média das qualidades para os algoritmos 2-RT, 2-RTg e A-RT.

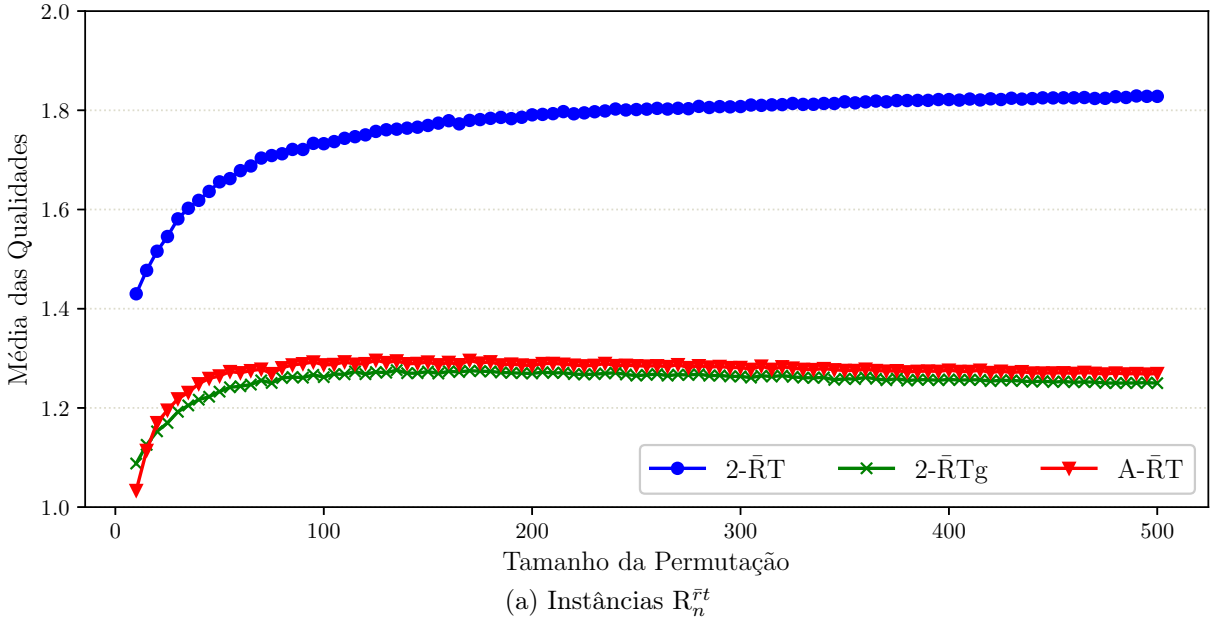


(a) Instâncias $R_n^{\bar{r}}$

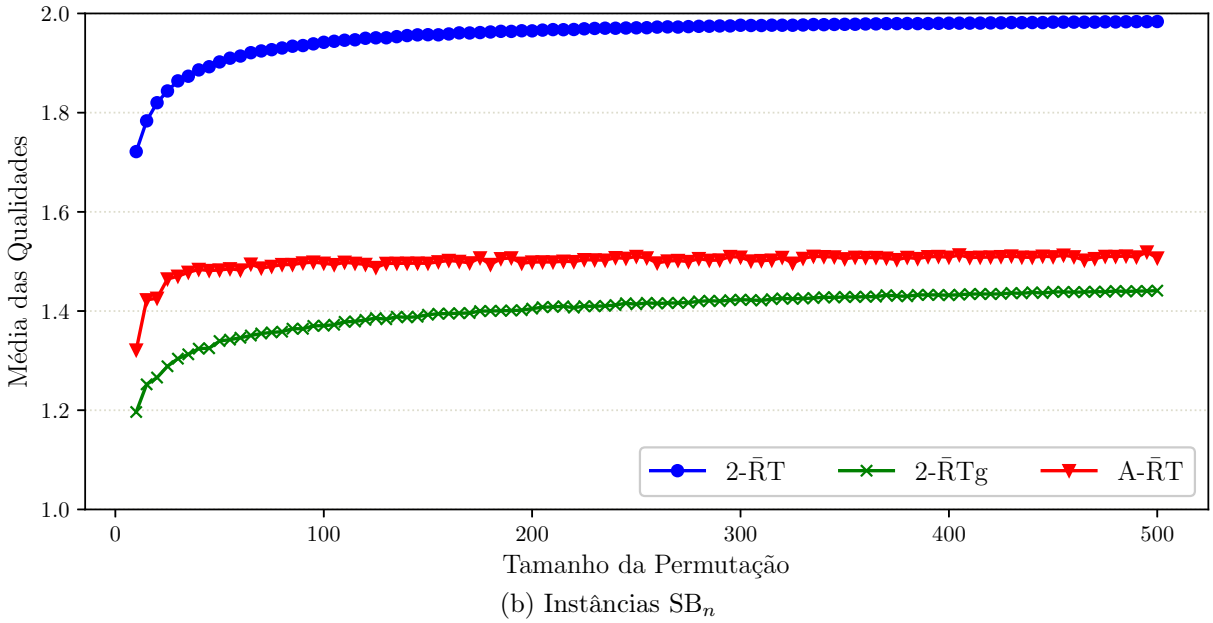


(b) Instâncias SB_n

Figura 3.4: Média das qualidades para os algoritmos 2-$\bar{R}$, 2-$\bar{R}$g e A-$\bar{R}$.



(a) Instâncias $R_n^{\bar{r}t}$



(b) Instâncias SB_n

Figura 3.5: Média das qualidades para os algoritmos 2-R̄T, 2-R̄Tg e A-R̄T.

3.3 Ordenação de Permutações Simples por Operações Ponderadas

Considerando uma classe de permutações chamada de permutações simples, apresentamos um algoritmo de 1.5-aproximação assintótica para os problemas SbFWT e SbFW$\bar{\text{R}}$T. Para isso, introduzimos o conceito de grafo de ciclos de uma permutação e apresentamos propriedades das permutações simples, incluindo um limitante inferior no número de permutações simples de tamanho n.

3.3.1 Grafo de Ciclos

A permutação estendida é obtida a partir de π ao adicionar os elementos $\pi_0 = 0$ e $\pi_{n+1} = n{+}1$. Para permutações com sinais, esses elementos possuem sinal positivo.

Definição 23. *O* grafo de ciclos *de uma permutação π é o grafo não direcionado $G(\pi) = (V, E_p \cup E_c)$, onde*

$$V = \{\pi_0, -\pi_1, \pi_1, \ldots, -\pi_n, \pi_n, -\pi_{n+1}\}, \tag{3.24}$$

$$E_c = \bigcup_{i=1}^{n+1} \{(i{-}1, -i)\}, \tag{3.25}$$

$$E_p = \bigcup_{i=1}^{n+1} \{(-\pi_i, \pi_{i-1})\}. \tag{3.26}$$

Ao usar grafo de ciclos, consideramos que $G(\pi)$ é definido a partir de uma permutação estendida. Dizemos que E_p é o conjunto de *arestas pretas* e E_c é o conjunto de *arestas cinzas* do grafo de ciclos $G(\pi)$. Essa definição de grafo de ciclos é equivalente para permutações com ou sem sinais. Dado um grafo de ciclos $G(\pi)$, rotulamos cada aresta preta $(-\pi_i, \pi_{i-1})$ com o inteiro i.

Convencionamos que o desenho de um grafo de ciclos respeita as seguintes regras. Os vértices são desenhados horizontalmente, respeitando a ordem dos elementos na permutação, de modo que o vértice π_0 é o vértice mais à esquerda e $-\pi_{n+1}$ é o vértice mais à direita. Além disso, as arestas pretas são desenhadas horizontalmente e as arestas cinzas são desenhadas como arcos tracejados. A Figura 3.6 apresenta um exemplo de um grafo de ciclos.

Como cada vértice de $G(\pi)$ é incidente a uma aresta preta e uma aresta cinza, isso implica na existência de uma decomposição única das arestas de $G(\pi)$ em ciclos com arestas de cores alternantes, ou seja, ciclos nos quais quaisquer duas arestas consecutivas possuem cores distintas. Para um ciclo $C = (v_1, v_2, \ldots, v_k)$ em $G(\pi)$, assumimos que v_1 é o vértice mais à direita de C.

Definição 24. *Um k-ciclo é um ciclo de $G(\pi)$ com k arestas pretas. Um k-ciclo é* ímpar *se k é impar e é* par *caso contrário.*

Definição 25. *Dizemos que um k-ciclo é* curto *se $k \leq 3$ e é* longo *caso contrário. Além disso, dizemos que um* 1*-ciclo é um ciclo unitário.*

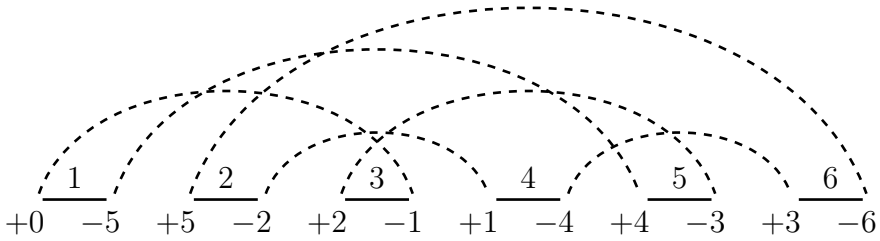


Figura 3.6: Grafo de ciclos $G(\pi)$ da permutação $\pi = (5\ 2\ 1\ 4\ 3)$. As arestas pretas são representadas por linhas contínuas e as arestas cinzas são representadas por linhas tracejadas. O grafo $G(\pi)$ contém os ciclos $C_1 = (-6, +3, -4, +1, -2, +5)$ e $C_2 = (-3, +4, -5, +0, -1, +2)$.

Definição 26. *Uma permutação π é* simples *se todos os ciclos de $G(\pi)$ são curtos.*

Denotamos o número de ciclos, o número de ciclos ímpares e o número de ciclos unitários em $G(\pi)$ por $c(\pi)$, $c_{odd}(\pi)$ e $c_1(\pi)$, respectivamente. Para qualquer permutação π, o número máximo de ciclos em $G(\pi)$ é igual a n+1 [13]. O único grafo de ciclos com n+1 ciclos é obtido quando $\pi = \iota$. Notamos que todos os ciclos em $G(\iota)$ são ciclos unitários. Dada uma permutação π, a variação no número de ciclos causada por uma operação β é denotada por $\Delta c(\pi, \beta) = c(\pi \circ \beta) - c(\pi)$. De forma similar, a variação no número de ciclos unitários causada por β é denotada por $\Delta c_1(\pi, \beta) = c_1(\pi \circ \beta) - c_1(\pi)$.

Definição 27. *Uma* configuração de ciclos *é um subgrafo induzido por um ou mais ciclos de $G(\pi)$.*

3.3.2 Permutações Simples

Nesta subseção, apresentamos um limitante inferior no número de permutações simples de tamanho n. Para isso, explicamos como transformar uma permutação qualquer em uma permutação simples, como descrito em Hartman e Sharan [19].

Dada uma permutação π, sejam $b = (v_b, w_b)$ uma aresta preta e $g = (w_g, v_g)$ uma aresta cinza pertencentes ao mesmo ciclo C em $G(\pi)$. Uma (g, b)-divisão de $G(\pi)$ é a seguinte sequência de operações que resultam no grafo $\hat{G}(\pi)$: (i) remova as aretas b e g em $G(\pi)$, (ii) adicione dois vértices v e w; (iii) adicione as arestas pretas (v_b, v) e (w, w_b); (iv) adicione duas arestas cinzas (w_g, w) e (v, v_g). Essas operações transformam o ciclo C pertencente a $G(\pi)$ em dois ciclos C_1 e C_2 no grafo $\hat{G}(\pi)$. Consequentemente, o grafo $\hat{G}(\pi)$ possui um ciclo a mais que $G(\pi)$.

Lema 19 (Hannenhalli e Pevzner [18])**.** *Dada uma permutação π de tamanho n, para qualquer (g, b)-divisão aplicada em $G(\pi)$ resultando em $\hat{G}(\pi)$, existe uma permutação $\hat{\pi}$ de tamanho n+1 tal que $\hat{G}(\pi) = G(\hat{\pi})$. Ou seja, qualquer (g, b)-divisão aplicada em $G(\pi)$ resulta em um grafo de ciclos $G(\hat{\pi})$ que descreve uma nova permutação com um elemento a mais.*

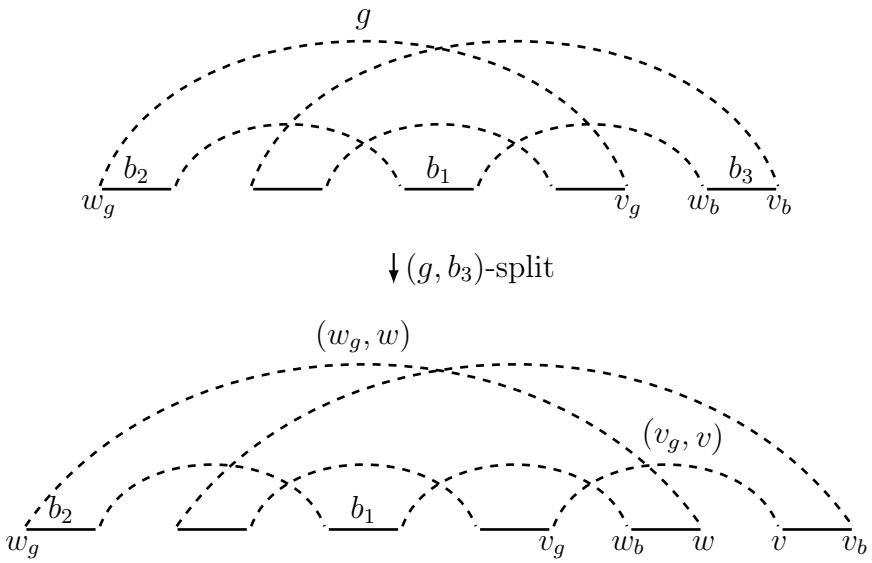


Figura 3.7: Uma (g, b_3)-divisão que transforma um k-ciclo em um 3-ciclo e um $(k-2)$-ciclo, com $k = 5$.

Como um resultado do Lema 19, podemos considerar uma (g, b)-divisão como uma sequência de operações que transforma uma permutação π de tamanho n em uma permutação $\hat{\pi}$ de tamanho $n+1$.

Dada uma permutação π de tamanho n, uma (g, b)-divisão que transforma π em $\hat{\pi}$ é *segura* se $n - c_{odd}(\pi) = n + 1 - c_{odd}(\hat{\pi})$ (observe que $\hat{\pi}$ tem tamanho $n + 1$). Dizemos que uma permutação simples $\hat{\pi}$ é *equivalente* a uma permutação π se $n - c_{odd}(\pi) = \hat{n} - c_{odd}(\hat{\pi})$, onde $\hat{n}$ é o tamanho de $\hat{\pi}$ e n é o tamanho π. O próximo lema mostra como encontrar uma divisão segura para qualquer permutação π não simples.

Lema 20. *Dada uma permutação π, para qualquer k-ciclo C em $G(\pi)$, onde $k > 3$, existe uma (g, b)-divisão que transforma C em um 3-ciclo C_1 e um $(k-2)$-ciclo C_2.*

Demonstração. Sejam b_1, b_2 e b_3 arestas pretas pertencentes a um mesmo k-ciclo C em $G(\pi)$, tal que b_2 e b_3 são conectadas a b_1 por arestas cinzas e $k > 3$. Seja g a aresta cinza que é adjacente a b_2 mas não é adjacente a b_1. Observe que podemos escolher as arestas b_1, b_2 e b_3 de forma que b_3 sempre é a aresta preta do ciclo C com o maior rótulo. Como mostrado na Figura 3.7, uma (g, b_3)-divisão transforma o ciclo C em dois ciclos C_1 e C_2 em $G(\hat{\pi})$. O ciclo C_1 é um 3-ciclo contendo as arestas pretas b_1, b_2 e (w_b, w), e o ciclo C_2 é um $(k-2)$-ciclo. Observe que essa divisão é segura pois o número de ciclos ímpares aumenta em uma unidade e, portanto, $n - c_{odd}(\pi) = n + 1 - c_{odd}(\hat{\pi})$. □

Aplicando repetidamente uma divisão segura em $G(\pi)$ (Lema 20), podemos transformar um permutação π de tamanho n em uma permutação simples $\hat{\pi}$ de tamanho $\hat{n}$. Pela definição de divisão segura, a permutação simples $\hat{\pi}$ é equivalente à permutação π. Dada uma permutação simples $\hat{\pi}$ e uma permutação π equivalente a $\hat{\pi}$, Hannenhalli e Pevzner [18] demonstraram que toda sequência de ordenação S para $\hat{\pi}$ simula uma sequência de ordenação S' para π, tal que $|S| = |S'|$.

O próximo lema mostra um limitante superior no número de (g, b)-divisões necessárias para transformar qualquer permutação em uma permutação simples.

Lema 21. *Para qualquer permutação π, a transformação de π em uma permutação simples equivalente $\hat{\pi}$, usando divisões seguras como descrito no Lema 20, adiciona no máximo $\lfloor \frac{n-c_1(\pi)}{2} \rfloor$ elementos na permutação.*

Demonstração. Suponha, para fins de contradição, que existe permutação π que necessita de $m = \lfloor \frac{n-c_1(\pi)}{2} \rfloor + 1$ dessas divisões para ser transformada em uma permutação simples $\hat{\pi}$. A permutação $\hat{\pi}$ deve possuir $n+m$ elementos e, consequentemente, $G(\hat{\pi})$ possui $n+m+1$ arestas pretas. Toda divisão segura, como descrito no Lema 20, cria um 3-ciclo no grafo de ciclos da permutação e nunca afeta um ciclo unitário no grafo de ciclos. Seja C_i o 3-ciclo adicionado pela i-ésima divisão, com $1 \leq i \leq m$. Note que as arestas pertencentes a C_i nunca são afetadas por uma divisão subsequente, já que C_i é um ciclo curto. Também sabemos que, além do 3-ciclo C_m, a última divisão deve criar um outro ciclo curto C'. Portanto, $G(\hat{\pi})$ deve conter o mesmo número de ciclos unitários de $G(\pi)$, os 3-ciclos C_i, com $1 \leq i \leq m$, e o ciclo curto C'. Já que C' é um 2-ciclo ou um 3-ciclo, o número de arestas pretas em $G(\hat{\pi})$ deve ser de pelo menos

$$\begin{aligned} 3m + c_1(\pi) + 2 &= 2\left(\left\lfloor \frac{n - c_1(\pi)}{2} \right\rfloor + 1\right) + m + c_1(\pi) + 2 \\ &\geq n - c_1(\pi) + m + c_1(\pi) + 2 \\ &= n + m + 2, \end{aligned}$$

o que contradiz o fato de que $G(\hat{\pi})$ possui $n + m + 1$ arestas pretas. □

A seguir, apresentamos um limitante inferior no número de permutações simples de tamanho n.

Teorema 7. *O número de permutações simples sem sinais de tamanho n, com $n > 3$, é maior ou igual a $\lfloor \frac{n}{3} \rfloor !$.*

Demonstração. Sejam $n' = 2 \times \lfloor \frac{n}{3} \rfloor$ e $m = n' + \frac{n'}{2}$. Note que $m \leq n$. Nessa demonstração, construímos um conjunto de permutações simples de tamanho n transformando ou estendendo permutações de tamanho n'. Um limitante inferior para tal conjunto também é um limitante inferior válido para o número de permutações simples de tamanho n.

Seja P^i qualquer conjunto de permutações de tamanho i, com $n' \leq i \leq m$. Iniciamos o conjunto $P^{n'}$ com todas as permutações de tamanho n'. Para $n'+1 \leq i \leq m$, construímos o conjunto P^i a partir das permutações de P^{i-1}. Para toda permutação $\pi \in P^{i-1}$, construímos uma permutação $\hat{\pi} \in P^i$ da seguinte forma:

- Se π é simples, então $\hat{\pi}$ é a permutação simples $(\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ i)$. Note que o último elemento $\pi_i = i$ cria um ciclo unitário na permutação;
- Caso contrário, construímos $\hat{\pi}$ aplicando uma (g, b)-divisão em um k-ciclo longo C em $G(\pi)$, tal que essa divisão remove a aresta preta mais à direita de C, ou seja, a aresta preta com o maior rótulo de C, e transforma C em um 3-ciclo C_1 e um $(k-2)$-ciclo C_2 (Lema 20).

Pelo Lema 21, o conjunto P^m possui apenas permutações simples. No entanto, no procedimento anterior, duas permutações π^1 e π^2 podem ser transformadas na mesma permutação $\hat{\pi}$. Assim, precisamos contar o número máximo de permutações em P^{i-1} que podem ser transformadas para a mesma permutação em P^i.

Seja $\hat{\pi}$ uma permutação pertencente a P^i, com $n' < i \leq m$. Já que $\hat{\pi}$ foi construída a partir da extensão ou pela quebra de um ciclo longo de uma permutação em P^{i-1}, podemos reverter esta operação apenas com a remoção do último elemento de $\hat{\pi}$, se esse elemento está em um ciclo unitário, ou revertendo a divisão que criou um 3-ciclo em $G(\hat{\pi})$. Observe que existe apenas uma forma de reverter uma divisão que criou um 3-ciclo em $\hat{\pi}$, já que sempre escolhemos a aresta preta mais à direita do ciclo afetado pela divisão. Uma permutação de tamanho i possui no máximo $\lfloor \frac{i+1}{3} \rfloor$ 3-ciclos. Assim, existem no máximo $\lfloor \frac{i+1}{3} \rfloor + 1$ permutações em P^{i-1} que podem ter sido transformadas na mesma permutação em P^i. Com isso, podemos derivar um limitante inferior no tamanho de P^i.

$$|P^i| \geq \begin{cases} n'!, & \text{se } k = n' \\ \dfrac{|P^{i-1}|}{\lfloor \frac{i+1}{3} \rfloor + 1}, & \text{caso contrário.} \end{cases}$$

Portanto, temos que

$$|P^m| \geq \frac{n'!}{\prod_{i=n'+1}^{m} \left\lfloor \frac{i+1}{3} \right\rfloor + 1} \geq \left\lfloor \frac{n'}{2} \right\rfloor ! = \left\lfloor \frac{n}{3} \right\rfloor !, \text{ já que } n' = 2 \times \left\lfloor \frac{n}{3} \right\rfloor .$$

Se $n = m$, então $P^n = P^m$. Caso contrário, construímos o conjunto P^n da seguinte forma. Para cada permutação π em P^m, construímos $\hat{\pi} = (\pi_1\ \pi_2\ \ldots\ \pi_m\ (m+1)\ \ldots\ n)$. Dessa forma, toda permutação simples de P^m corresponde a uma permutação simples única em P^n e, consequentemente, $|P^n| = |P^m|$. □

Teorema 8. *O número de permutações simples com sinais de tamanho n, com $n > 3$, é maior ou igual a $\lfloor \frac{n}{3} \rfloor ! \times 2^{2\lfloor n/3 \rfloor}$.*

Demonstração. Similar à prova do Teorema 7. □

3.3.3 Algoritmo de 1.5-Aproximação Assintótica

Para provar o fator de aproximação do algoritmo apresentado nesta seção, os próximos lemas apresentam um novo limitante inferior a partir da comparação entre o número de ciclos unitários e o número de *breakpoints* em uma permutação.

Lema 22. *Para qualquer permutação sem sinais π, $n + 1 - c_1(\pi) \geq b_t(\pi) \geq (n + 1 - c_1(\pi)) - 2$.*

Demonstração. Para toda aresta preta $b_i = (-\pi_i, \pi_{i-1})$, temos que: (i) se b_i não pertence a um ciclo unitário, então $\pi_i - \pi_{i-1} \neq 1$ e, consequentemente, existe um *breakpoint* de transposição entre π_{i-1} e π_i, exceto se $i - 1 = 0$ ou $i = n + 1$; (ii) se b_i pertence a um ciclo unitário, então $\pi_i - \pi_{i-1} = 1$ e, consequentemente, não existe *breakpoint* de transposição

entre π_{i-1} e π_i. Como $G(\pi)$ possui $n+1$ arestas pretas, o número de arestas pretas que não pertencem a um ciclo unitário é igual a $n+1-c_1(\pi)$. Portanto, $n+1-c_1(\pi) \geq b_t(\pi) \geq (n+1-c_1(\pi))-2$. □

Lema 23. *Para qualquer permutação com sinais π, $n+1-c_1(\pi) \geq b_{\bar{r}}(\pi) \geq (n+1-c_1(\pi))-2$.*

Demonstração. Similar à prova do Lema 22. □

Lema 24. *Para qualquer permutação sem sinais π, $d_t^f(\pi) \geq (n+1-c_1(\pi))-2$.*

Demonstração. Segue diretamente dos lemas 14 e 22. □

Lema 25. *Para qualquer permutação com sinais π, $d_{\bar{r}t}^f(\pi) \geq (n+1-c_1(\pi))-2$.*

Demonstração. Segue diretamente dos lemas 15 e 23. □

Agora, apresentamos novos fatores de aproximação a partir de resultados apresentados por Elias e Hartman [13] e Oliveira *et al.* [30].

Lema 26. *Para qualquer permutação simples sem sinais π, existe um algoritmo de tempo polinomial que ordena π com uma sequência de transposições S tal que $f(S) \leq \frac{3}{2}(n+1-c_1(\pi))$.*

Demonstração. Como apresentado por Elias e Hartman [13], para toda permutação simples sem sinais π, existe uma sequência S' com m transposições tal que $c_1(\pi \circ S') - c_1(\pi) \geq 2m$. Eles também apresentaram como achar tal sequência em tempo polinomial listando todos os casos possíveis. Como a permutação identidade é a única com $n+1$ ciclos unitários, um algoritmo que repetidamente aplica tais sequências ordena π usando $(n+1-c_1(\pi))/2$ ou menos transposições. Como o custo de qualquer transposição é menor ou igual a 3, essa sequência de ordenação tem custo menor ou igual a $\frac{3}{2}(n+1-c_1(\pi))$. □

Lema 27. *Para qualquer permutação simples com sinais π, existe um algoritmo de tempo polinomial que ordena π com uma sequência de operações S tal que $f(S) \leq \frac{3}{2}(n+1-c_1(\pi))$.*

Demonstração. Oliveira *et al.* [30] apresentaram oito configurações de ciclos tal que pelo menos uma dessas configurações existe em $G(\pi)$ (Figura 3.8). Para cada configuração, eles apresentaram uma sequência que aumenta o número de ciclos unitários de forma que a permutação resultante também é simples.

Para cada uma das oito configurações, existe uma sequência de operações S que aumenta o número de ciclos unitários em k unidades tal que $f(S) \leq \frac{3}{2}k$. Dessa forma, concluímos que um algoritmo que repetidamente aplica tais sequências ordena π com custo menor ou igual a $\frac{3}{2}(n+1-c_1(\pi))$. □

Teorema 9. *O problema da Ordenação de Permutações Simples por Transposições Ponderadas pelo Número de Fragmentações possui um algoritmo com fator de aproximação assintótica de* 1.5.

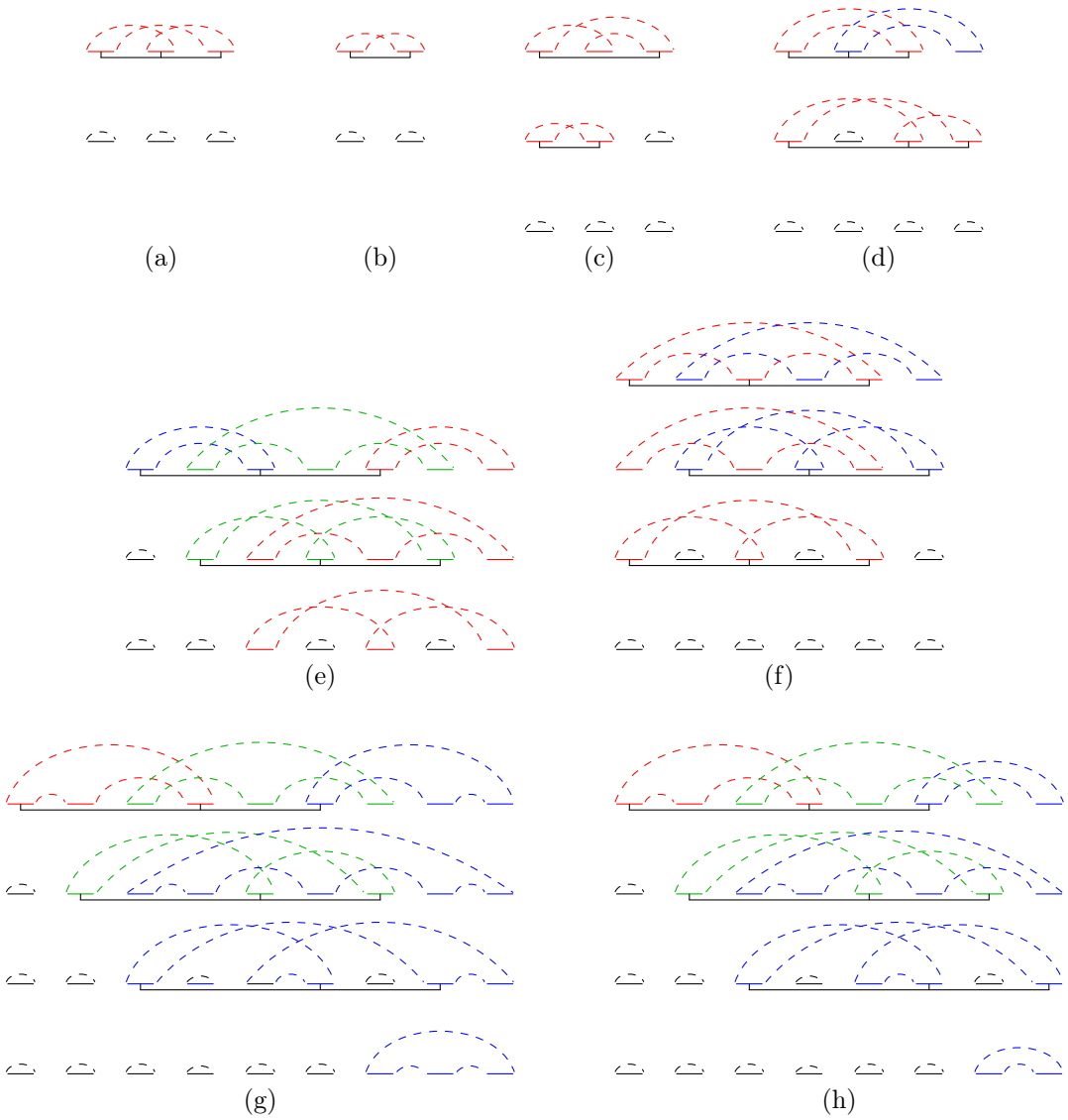


Figura 3.8: Configurações contidas no grafo de ciclos de uma permutação simples com sinais [30]. Para cada uma das oito configurações, é apresentada uma sequência de operações que aumenta o número de ciclos unitários da permutação. Cada operação é representada por uma linha que conecta as arestas pretas afetadas por essa operação. Notamos que uma reversão afeta duas arestas pretas e uma transposição afeta três arestas pretas.

Demonstração. Seja π uma permutação qualquer. Pelo Lema 26, existe uma sequência de ordenação S com $f(S) \leq \frac{3}{2}(n+1-c_1(\pi))$. Como $d_t^f(\pi) \geq (n+1-c_1(\pi)) - 2$ (Lema 24), temos que

$$f(S) \leq \frac{3}{2}(n+1-c_1(\pi)) \leq \frac{3}{2}(d_t^f(\pi)+2) = \frac{3}{2}d_t^f(\pi)+3 \ .$$

Assim, concluímos que esse algoritmo é uma 1.5-aproximação assintótica. □

Teorema 10. *O problema da Ordenação de Permutações Simples por Reversões com Sinais e Transposições Ponderadas pelo Número de Fragmentações possui um algoritmo com fator de aproximação assintótica de* 1.5*.*

Demonstração. Similar à prova do Teorema 9. □

Usamos o seguinte procedimento para adaptar os algoritmos mencionados nos lemas 26 e 27, a fim de serem utilizados para uma permutação qualquer π: (i) transforme π em uma permutação simples $\hat{\pi}$ aplicando divisões seguras (Lema 20); (ii) use o algoritmo do Lema 26 ou do Lema 27 para obter uma sequência de ordenação $\hat{S}$ para $\hat{\pi}$; (iii) transforme $\hat{S}$ em uma sequência de ordenação S para π, tal que $|\hat{S}| = |S|$ [18], e retorne S. Nos referimos a esse procedimento como A-SIMPLE.

Os próximos lemas apresentam o fator de aproximação para A-SIMPLE considerando alguns casos. Esses resultados são similares para os problemas SbFWT e SbFW$\bar{\text{R}}$T.

Lema 28. *A-SIMPLE é uma* 9/4*-aproximação assintótica para os problemas SbFWT e SbFW$\bar{R}$T.*

Demonstração. Considere o problema SbFWT e uma permutação sem sinais π. Sejam $\hat{\pi}$ uma permutação simples construída pelo primeiro passo de A-SIMPLE, $\hat{S}$ a sequência de ordenação para $\hat{\pi}$ retornada pelo segundo passo de A-SIMPLE, e S a sequência de ordenação retornada no último passo de A-SIMPLE. Pelo Lema 21, o tamanho $\hat{n}$ de $\hat{\pi}$ é menor ou igual a $n + \lfloor \frac{n-c_1(\pi)}{2} \rfloor$. Pelo Lema 26, $|\hat{S}| \leq \frac{\hat{n}+1-c_1(\pi)}{2}$ e, consequentemente, $f(S) \leq \frac{3}{2}(\hat{n}+1-c_1(\pi))$. Observe que $c_1(\pi) = c_1(\hat{\pi})$ e $|S| = |\hat{S}|$. Usando o Lema 24, temos que

$$\begin{aligned}
f(S) &\leq \frac{3}{2}(\hat{n}+1-c_1(\pi)) \\
&\leq \frac{3}{2}\left(n + \left\lfloor \frac{n-c_1(\pi)}{2} \right\rfloor + 1 - c_1(\pi)\right) \\
&\leq \frac{3}{2}\left(d_t^f(\pi) + 2 + \left\lfloor \frac{d_t^f(\pi)+1}{2} \right\rfloor\right) \\
&\leq \frac{3}{2}\left(\frac{3}{2}d_t^f(\pi) + 3\right) \\
&= \frac{9}{4}d_t^f(\pi) + \frac{9}{2}
\end{aligned}$$

Portanto, concluímos que A-SIMPLE possui fator de aproximação assintótico de 9/4 para SbFWT.

A demonstração é similar para o problema SbFW$\bar{\text{R}}$T. □

Lema 29. *Para qualquer permutação π, tal que π contém apenas k-ciclos com $k \leq 5$, a transformação de π em uma permutação simples equivalente $\hat{\pi}$, usando divisões seguras como descrito no Lema 20, adiciona no máximo $\lfloor \frac{n+1-c_1(\pi)}{4} \rfloor$ elementos na permutação.*

Demonstração. O número de ciclos longos em $G(\pi)$ é de no máximo $\lfloor \frac{n+1-c_1(\pi)}{4} \rfloor$, já que qualquer ciclo longo de $G(\pi)$ deve ser um 4-ciclo ou um 5-ciclo. Para cada ciclo longo C de $G(\pi)$, uma divisão segura aplicada em C transforma esse ciclo em dois ciclos curtos (Lema 20). Portanto, a transformação de π em uma permutação simples equivalente $\hat{\pi}$ usa uma divisão segura para cada ciclo longo em $G(\pi)$ e, consequentemente, adiciona no máximo $\lfloor \frac{n+1-c_1(\pi)}{4} \rfloor$ elementos na permutação. □

Lema 30. *Para qualquer permutação π, tal que π contém apenas k-ciclos com $k \leq 5$, A-SIMPLE é uma 15/8-aproximação assintótica para os problemas SbFWT e SbFW$\bar{R}$T.*

Demonstração. Similar à prova do Lema 28, usando o Lema 29. □

Lema 31. *Para qualquer permutação π, se a transformação de π em uma permutação simples equivalente $\hat{\pi}$, usando divisões seguras como descrito no Lema 20, adiciona $\lfloor \frac{n+1-c_1(\pi)}{3} \rfloor - 3$ elementos ou menos, então A-SIMPLE possui fator de aproximação menor do que 2 para os problemas SbFWT e SbFW$\bar{R}$T.*

Demonstração. Similar à prova do Lema 28. □

3.4 Diâmetro

Nesta seção são apresentados limitantes para o diâmetro dos cinco problemas estudados. Para isso, ao longo desta seção, são apresentadas algumas famílias de permutações.

Para SbFWR, seja

$$\pi_n^r = \begin{cases} (n\ 1\ n{-}2\ n{-}4\ \ldots\ 4\ 2\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}1), & \text{se } n \text{ é par} \\ (n\ 1\ n{-}2\ n{-}4\ \ldots\ 5\ 3\ n{-}3\ n{-}5\ \ldots\ 2\ n{-}1), & \text{se } n \text{ é ímpar.} \end{cases} \tag{3.27}$$

Lema 32. *Para $n \geq 8$, $d_r^f(\pi_n^r) \geq n - 1$.*

Demonstração. Como π_n^r, com $n \geq 8$, possui $n{-}1$ *breakpoints*, o resultado segue diretamente do fato de que $d_r^f(\pi_n^r) \geq b_r(\pi_n^r)$ (Lema 13). □

Lema 33. *Qualquer limitante superior para o diâmetro do problema de Ordenação por Reversões de Prefixo e/ou Sufixo também é um limitante superior para o diâmetro de SbFWR.*

Demonstração. Seja $up(n)$ um limitante superior do problema de Ordenação por Reversões de Prefixo e/ou Sufixo. Esse limitante superior implica que qualquer permutação π possui uma sequência de ordenação S, tal que $|S| \leq up(n)$ e S é formada apenas de reversões de prefixo, sufixo ou completas. Portanto, $f(S) \leq |S| \leq up(n)$, já que o custo de cada operação de S é menor ou igual a 1. Como a sequência S também é uma ordenação válida para SbFWR, temos que $up(n)$ também é um limitante superior do diâmetro de SbFWR. □

Notamos que o uso do prefixo p em um modelo de rearranjo M (*e.g.* pr, pt e $p\bar{r}$) denota que apenas operações de prefixo são permitidas naquele modelo.

Lema 34. *Para $n \geq 8$, $n-1 \leq D_r^f(n) \leq \frac{18n}{11} + O(1)$.*

Demonstração. A partir do limitante inferior da família de permutações π_n^r, temos que $D_r^f(n) \geq n-1$, com $n \geq 8$ (Lema 32). Como existe limitante superior para $D_{pr}(n)$ de $\frac{18n}{11} + O(1)$ [11], temos que $D_r^f(n) \leq \frac{18n}{11} + O(1)$ (Lema 33). □

Para SbFWT, seja

$$\pi_n^t = (n\ n{-}1\ n{-}2\ \ldots\ 2\ 1). \tag{3.28}$$

Lema 35. *Para $n \geq 3$, $d_t^f(\pi_n^t) \geq n$.*

Demonstração. A permutação π_n^t possui $n-1$ *breakpoints* de transposição. Pelo Lema 14, $d_t^f(\pi_n^t) \geq b_t(\pi_n^t)$ e, consequentemente, $d_t^f(\pi_n^t) \geq n-1$. No entanto, esse limitante inferior pode ser melhorado. Note que uma transposição completa de custo 1 não remove nenhum *breakpoint* de π_n^t e qualquer outra transposição pode remover no máximo 1 *breakpoint*. Assim, o custo da remoção do primeiro *breakpoint* é de pelo menos 2 e os outros $n-2$ *breakpoints* são removidos com custo maior ou igual a $n-2$. Portanto, $d_t^f(\pi_n^t) \geq n$. □

Lema 36. *Se $ub_t(n)$ é um limitante superior para o diâmetro do problema de Ordenação por Transposições de Prefixo e/ou Sufixo, então $D_t^f(n) \leq 2ub_t(n)$.*

Demonstração. Similar à prova do Lema 33. Observe que uma transposição de prefixo, sufixo ou completa tem custo de no máximo 2. □

Lema 37. *Para $n \geq 3$, $n \leq D_t^f(n) \leq 2(n - \log_{7/2} n)$.*

Demonstração. A partir do limitante inferior da família π_n^t, temos que $D_t^f(n) \geq n$, com $n \geq 3$ (Lema 35). Como existe limitante superior para $D_{pt}(n)$ de $n - \log_{7/2} n$ [10], pelo Lema 36, temos que $D_t^f(n) \leq 2(n - \log_{7/2} n)$. □

Para SbFWRT, seja

$$\pi_n^{rt} = \begin{cases} (n\ 1\ n{-}2\ n{-}4\ \ldots\ 4\ 2\ n{-}3\ n{-}5\ \ldots\ 3\ n{-}1), & \text{se } n \text{ é par} \\ (n\ 1\ n{-}2\ n{-}4\ \ldots\ 5\ 3\ n{-}3\ n{-}5\ \ldots\ 2\ n{-}1), & \text{se } n \text{ é ímpar.} \end{cases} \tag{3.29}$$

Lema 38. *Para $n \geq 8$, $d_{rt}^f(\pi_n^{rt}) \geq n - 1$.*

Demonstração. Como π_n^{rt}, com $n \geq 8$, possui $n-1$ *breakpoints*, o resultado segue diretamente do fato de que $d_{rt}^f(\pi_n^{rt}) \geq b_r(\pi_n^{rt})$. □

Lema 39. *Para $n \geq 8$, $n - 1 \leq D_{rt}^f(n) \leq \frac{18n}{11} + O(1)$.*

Demonstração. A partir do limitante inferior da família de permutações π_n^{rt}, temos que $D_{rt}^f(n) \geq n-1$, com $n \geq 8$ (Lema 38). Observe que $D_{rt}^f(n) \leq D_r^f(n)$, já que $d_{rt}^f(\pi) \leq d_r^f(\pi)$, para toda permutação π. Portanto, $D_{rt}^f(n) \leq D_r^f(n) \leq \frac{18n}{11} + O(1)$, pelo Lema 34. □

Para SbFW$\bar{\text{R}}$, seja

$$\pi_n^{\bar{r}} = (-1\ -2\ \ldots\ -(n{-}1)\ -n). \tag{3.30}$$

Lema 40. *Para $n \geq 2$, $d_{\bar{r}}^f(\pi_n^{\bar{r}}) \geq n$.*

Demonstração. Seja $S = \langle \bar{\rho}_1, \bar{\rho}_2, \ldots, \bar{\rho}_\ell \rangle$ uma sequência de ordenação ótima de $\pi_n^{\bar{r}}$, tal que S possui no máximo uma reversão do tipo $\bar{\rho}(1,n)$ (Lema 3). Note que não existe reversão com sinais que remova *breakpoints* de $\pi_n^{\bar{r}}$. Sendo assim, $\pi_n^{\bar{r}} \circ \bar{\rho}_1$ também possui $n{-}1$ *breakpoints*. Se $\bar{\rho}_1 \neq \bar{\rho}(1,n)$, então temos um custo de pelo menos 1 na primeira reversão e as próximas reversões de S devem ter um custo de pelo menos $n{-}1$, já que $\pi_n^{\bar{r}} \circ \bar{\rho}_1$ possui $n{-}1$ *breakpoints*. Caso contrário, $\bar{\rho}_1 = \bar{\rho}(1,n)$ e $\pi_n^{\bar{r}} \circ \bar{\rho}_1 = (n\ (n{-}1)\ \ldots\ 2\ 1)$. Nesse caso, também não existe reversão com sinais que remova *breakpoints* de $\pi_n^{\bar{r}} \circ \bar{\rho}_1$, fazendo com que $\pi_n^{\bar{r}} \circ \bar{\rho}_1 \circ \bar{\rho}_2$ também possua $n{-}1$ *breakpoints*. Assim, $\bar{\rho}_1$ e $\bar{\rho}_2$ têm custo maior ou igual a 1, pois S possui no máximo uma reversão completa, e as próximas reversões de S devem ter um custo de pelo menos $n{-}1$, já que $\pi_n^{\bar{r}} \circ \bar{\rho}_1 \circ \bar{\rho}_2$ possui $n{-}1$ *breakpoints*. Portanto, em ambos os casos, temos que $d_{\bar{r}}^f(\pi_n^{\bar{r}}) \geq n$. □

Lema 41. *Qualquer limitante superior para o diâmetro do problema de Ordenação por Reversões com Sinais de Prefixo e/ou Sufixo também é um limitante superior para o diâmetro de SbFW$\bar{R}$.*

Demonstração. Similar à prova do Lema 33. □

Lema 42. *Para $n \geq 2$, $D_{\bar{r}}^f(n) \geq n$ e para $n \geq 16$, $D_{\bar{r}}^f(n) \leq 2n - 6$.*

Demonstração. A partir do limitante inferior da família $\pi_n^{\bar{r}}$, temos que $D_{\bar{r}}^f(n) \geq n$, com $n \geq 2$ (Lema 40). Como existe limitante superior para $D_{p\bar{r}}(n)$ de $2n{-}6$ [12], temos que $D_{\bar{r}}^f(n) \leq 2n - 6$ (Lema 41). □

Para SbFW$\bar{\text{R}}$T, seja

$$\pi_n^{\bar{r}t} = (n\ (n{-}1)\ \ldots\ 2\ 1). \tag{3.31}$$

Lema 43. *Para $n \geq 3$, $d_{\bar{r}t}^f(\pi_n^{\bar{r}t}) \geq n$.*

Demonstração. Seja $S = \langle \beta_1, \beta_2, \ldots, \beta_\ell \rangle$ uma sequência de ordenação ótima de $\pi_n^{\bar{r}t}$, tal que S possui no máximo uma reversão do tipo $\bar{\rho}(1,n)$ (Lema 3). Dividiremos a análise dependendo do tipo da operação β_1.

1. β_1 é uma reversão com sinais. Note que não existe reversão com sinais que remova *breakpoints* de $\pi_n^{\bar{r}t}$. Então, $\pi_n^{\bar{r}t} \circ \beta_1$ também possui $n{-}1$ *breakpoints*. Se $\beta_1 \neq \bar{\rho}(1,n)$, temos que a primeira operação tem custo maior ou igual a 1 e as próximas operações de S possuem custo de pelo menos $n{-}1$, já que $\pi_n^{\bar{r}t} \circ \beta_1$ possui $n{-}1$ *breakpoints*. Caso contrário, $\beta_1 = \bar{\rho}(1,n)$ e $\pi_n^{\bar{r}t} \circ \beta_1 = (-1\ -2\ \ldots\ -(n{-}1)\ -n)$. Nesse caso, não existe reversão ou transposição que remova *breakpoints* de $\pi_n^{\bar{r}t} \circ \beta_1$, fazendo com que $\pi_n^{\bar{r}t} \circ \beta_1 \circ \beta_2$ também possua $n{-}1$ *breakpoints*. Assim, β_1 e β_2 têm custo maior ou igual a 1, pois S possui no máximo uma reversão completa, e as outras operações de S custam no mínimo $n{-}1$, já que $\pi_n^{\bar{r}t} \circ \beta_1 \circ \beta_2$ possui $n{-}1$ *breakpoints*.

Tabela 3.2: Resumo dos limitantes para o diâmetro dos problemas SbFWR, SbFWT, SbFWRT, SbFW$\bar{\text{R}}$ e SbFW$\bar{\text{R}}$T.

Problema	**Limitante Inferior**	**Limitante Superior**
SbFWR	$n-1$ (Lema 34)	$\frac{18n}{11} + O(1)$ (Lema 34)
SbFWT	n (Lema 37)	$2(n - \log_{7/2} n)$ (Lema 37)
SbFWRT	$n-1$ (Lema 39)	$\frac{18n}{11} + O(1)$ (Lema 39)
SbFW$\bar{\text{R}}$	n (Lema 42)	$2n - 6$ (Lema 42)
SbFW$\bar{\text{R}}$T	n (Lema 44)	$2n - 6$ (Lema 44)

2. β_1 é uma transposição. Se β_1 é uma transposição completa, então essa transposição não remove *breakpoints* de $\pi_n^{\bar{r}t}$ e, portanto, $\pi_n^{\bar{r}t} \circ \beta_1$ possui $n-1$ *breakpoints*. Sendo assim, β_1 tem custo 1 e as outras operações de S custam pelo menos $n-1$, já que $\pi_n^{\bar{r}t} \circ \beta_1$ possui $n-1$ *breakpoints*. Caso contrário, β_1 não é completa e tem custo maior ou igual a 2. Note que não existe transposição que consiga remover mais de um *breakpoint* em $\pi_n^{\bar{r}t}$. Assim, a remoção do primeiro *breakpoint* tem um custo maior ou igual a 2 e os outros $n-2$ *breakpoints* são removidos com um custo de pelo menos $n-2$.

Em todos os casos, o custo de S é maior ou igual a n e, portanto, $d^f_{\bar{r}t}(\pi_n^{\bar{r}t}) \geq n$. □

Lema 44. *Para $n \geq 3$, $D^f_{\bar{r}t}(n) \geq n$ e para $n \geq 16$, $D^f_{\bar{r}t}(n) \leq 2n - 6$.*

Demonstração. A partir do limitante inferior da família de permutações $\pi_n^{\bar{r}t}$, temos que $D^f_{\bar{r}t}(n) \geq n$, com $n \geq 3$ (Lema 43). Observe que $D^f_{\bar{r}t}(n) \leq D^f_{\bar{r}}(n)$, já que $d^f_{\bar{r}t}(\pi) \leq d^f_{\bar{r}}(\pi)$, para toda permutação π. Portanto, $D^f_{\bar{r}t}(n) \leq D^f_{\bar{r}}(n) \leq 2n-6$, pelo Lema 42. □

A Tabela 3.2 resume os limitantes apresentados para o diâmetro dos problemas SbFWR, SbFWT, SbFWRT, SbFW$\bar{\text{R}}$ e SbFW$\bar{\text{R}}$T.

Capítulo 4

Operações Curtas Ponderadas pelo Tamanho

Indícios sugerem que, em alguns casos, as operações que ocorreram no processo evolutivo não agem em segmentos muito longos do genoma [6, 24]. Essa observação motivou os problemas da Ordenação de Permutações por Operações Ponderadas pelo Tamanho [16, 20, 21, 22, 34] e Ordenação de Permutações por Operações de Tamanho Limitado [27, 29, 31].

Na Ordenação de Permutações por Operações Ponderadas pelo Tamanho, o custo de uma operação β é igual a $|\beta|^\alpha$, para $\alpha > 0$. Considerando reversões sem sinais e $\alpha = 1$, Pinter e Skiena [31] apresentaram um algoritmo de aproximação com fator de $O(\lg^2 n)$. Considerando o mesmo modelo de rearranjo, Bender *et al.* [4] apresentaram uma $O(\lg n)$-aproximação, para $1 \leq \alpha < 2$, e uma 2-aproximação, para $\alpha \geq 2$. Além disso, Bender *et al.* [4] concluíram que o problema pode ser resolvido em tempo polinomial quando $\alpha \geq 3$.

Considerando reversões com sinais, Swidan *et al.* [33] apresentaram uma $O(\lg n)$-aproximação, para $1 \leq \alpha < 2$, e uma $O(1)$-aproximação, para $\alpha \geq 2$. Considerando reversões sem sinais e transposições ou apenas transposições, Lintzmayer *et al.* [27] apresentaram os seguintes resultados: uma $O(\lg^2 n)$-aproximação, para $\alpha = 1$; uma $O(\lg n)$-aproximação, para $1 < \alpha < 2$; uma 2-aproximação, para $\alpha \geq 2$; e um algoritmo polinomial, para $\alpha \geq 3$.

Dado um $\lambda > 1$, apenas operações de tamanho menor ou igual a λ são permitidas nos problemas de Ordenação de Permutações por Operações de Tamanho Limitado. Como mencionado na Seção 2.2, uma operação é *super curta* se $|\beta| \leq 2$ e *curta* se $|\beta| \leq 3$. Os problemas da Ordenação de Permutações por Operações Super Curtas possuem algoritmos polinomiais [16, 21] considerando todos os cinco modelos de rearranjo envolvendo reversões sem sinais, reversões com sinais e transposições. Os melhores resultados para a Ordenação de Permutações por Reversões Curtas são algoritmos de 5-aproximação [16] e 2-aproximação [20] para permutações com e sem sinais, respectivamente. Já a Ordenação de Permutações por Transposições Curtas possui uma 5/4-aproximação [22]. Para o mesmo problema, existe uma $(1 + n/x)$-aproximação [23], onde n é o tamanho da permutação e x é a quantidade de inversões na permutação. Esse algoritmo apresenta um melhor fator de aproximação para permutações com muitas inversões. Considerando os problemas da Ordenação de Permutações por Reversões Curtas e Transposições Curtas,

os melhores resultados conhecidos são uma 3-aproximação [16] e uma 2-aproximação [34] para permutações com e sem sinais, respectivamente.

Nguyen *et al.* [29] estudaram o problema de Ordenação de Permutações por λ-Reversões sem Sinais Ponderadas pelo Tamanho, o qual combina a ponderação por tamanho com a restrição no tamanho dos rearranjos permitidos. Os resultados apresentados foram uma $O(\lg\ n)$-aproximação, para $\lambda = \Omega(n)$, e uma $(2\lg^2\ n + \lg\ n)$-aproximação, para $\lambda = o(n)$. No melhor do nosso conhecimento, não existem resultados conhecidos para outros modelos de rearranjo ou resultados específicos para $\lambda \leq 3$.

Os problemas estudados neste capítulo combinam a ponderação pelo tamanho do rearranjo com a restrição de que apenas operações curtas são permitidas. Essa variação é chamada de Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho. Os problemas estudados para essa abordagem consideram as operações de reversões sem sinais curtas (SbLWsR), reversões com sinais curtas (SbLWs$\bar{\text{R}}$), transposições curtas (SbLWsT), reversões sem sinais curtas e transposições curtas (SbLWsRsT), reversões com sinais curtas e transposições curtas (SbLWs$\bar{\text{R}}$sT).

Utilizamos a notação $d_M^\ell(\pi)$ para denotar a distância de ordenação, onde M é o modelo de rearranjo utilizado e ℓ indica a ponderação pelo tamanho da operação. Neste capítulo, exceto quando especificado o contrário, a função de custo segue a ponderação pelo tamanho com $\alpha = 1$, ou seja, o custo de uma operação é igual ao seu tamanho.

O restante deste capítulo é dividido da seguinte forma. A Seção 4.1 apresenta conceitos utilizados nas outras seções. A Seção 4.2 relaciona a abordagem não ponderada da Ordenação de Permutações por Operações Curtas com a abordagem ponderada. As seções 4.3 e 4.4 descrevem algoritmos de aproximação para permutações sem sinais e com sinais, respectivamente. A Seção 4.5 apresenta resultados experimentais dos algoritmos desenvolvidos. Por último, a Seção 4.6 introduz uma análise dos problemas quando $\alpha > 1$.

4.1 Preliminares

Esta seção define conceitos e notações geralmente utilizados no estudo de problemas de ordenação por operações curtas.

Definição 28. *Um par de elementos* (π_i, π_j) *de uma permutação* π *é uma* inversão *se* $i < j$ *e* $|\pi_i| > |\pi_j|$. *Usamos* $\text{Inv}(\pi)$ *para denotar o número de inversões em uma permutação* π.

Exemplo 11. *A permutação* $\pi = (3\ 4\ 1\ 2)$ *possui as inversões* $(\pi_1 = 3, \pi_3 = 1)$, $(\pi_1 = 3, \pi_4 = 2)$, $(\pi_2 = 4, \pi_3 = 1)$ *e* $(\pi_2 = 4, \pi_4 = 2)$. *Neste exemplo,* $\text{Inv}(\pi) = 4$.

Para qualquer permutação π, $\text{Inv}(\pi) = 0$ se $|\pi_1| < |\pi_2| < \ldots < |\pi_n|$. Ao considerar permutações sem sinais, apenas a permutação identidade ι possui $\text{Inv}(\iota) = 0$.

Lema 45. *Para qualquer permutação* π, *se* $\text{Inv}(\pi) > 0$, *então existe uma inversão* (π_i, π_{i+1}).

Demonstração. Seja $\langle \pi_1, \ldots, \pi_i \rangle$ uma subsequência maximal tal que $|\pi_j| < |\pi_{j+1}|$, para todo $1 \leq j < i$. Como $\text{Inv}(\pi) > 0$, temos que $i < n$. Portanto, $|\pi_i| > |\pi_{i+1}|$ e (π_i, π_{i+1}) é uma inversão. □

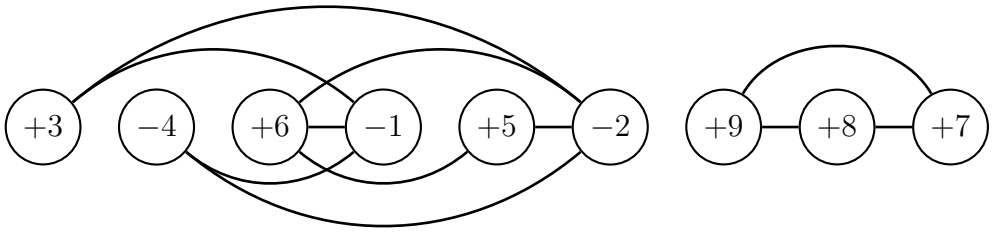


Figura 4.1: Grafo de inversões $G^{inv}(\pi)$ da permutação com sinais $\pi = (+3\ -4\ +6\ -1\ +5\ -2\ +9\ +8\ +7)$. Neste exemplo, $c^{inv}(\pi) = 2$, $c^{inv}_{odd}(\pi) = 1$ e $\gamma(\pi) = 9$.

Definição 29. *Dada uma permutação π, a variação no número de inversões causada por uma operação β é denotada por* $\Delta\text{Inv}(\pi, \beta) = \text{Inv}(\pi) - \text{Inv}(\pi \circ \beta)$.

A partir do Lema 45, para qualquer permutação π tal que $\text{Inv}(\pi) > 0$, concluímos que existe uma operação super curta β com $\Delta\text{Inv}(\pi, \beta) = 1$. O próximo lema mostra limitantes no valor $\Delta\text{Inv}(\pi, \beta)$, onde $|\beta| \leq 3$.

Lema 46 (Galvão *et al.* [16])**.** *Para qualquer permutação π e operação curta β,*

i) $-1 \leq \Delta\text{Inv}(\pi, \beta) \leq 1$ *se β tem tamanho* 2*,*

ii) $-2 \leq \Delta\text{Inv}(\pi, \beta) \leq 2$ *se β é uma transposição de tamanho* 3*, e*

iii) $-3 \leq \Delta\text{Inv}(\pi, \beta) \leq 3$ *se β é uma reversão de tamanho* 3*.*

A seguir, definimos uma estrutura chamada de grafo de inversões. Os vértices desse grafo correspondem aos elementos da permutação e as arestas correspondem às inversões de π.

Definição 30. *O* grafo de inversões *de uma permutação π é o grafo não direcionado $G^{inv}(\pi) = (V, E)$, onde $V = \{\pi_1, \pi_2, \ldots, \pi_n\}$ e $E = \{(\pi_i, \pi_j) : o\ par\ (\pi_i, \pi_j)\ é\ uma\ inversão\ em\ \pi\}$.*

Definição 31. *Dado um grafo de inversões $G^{inv}(\pi)$, um componente conexo C de $G^{inv}(\pi)$ é* ímpar *se ele contém um número ímpar de elementos negativos (vértices negativos) e é* par *caso contrário.*

Usamos $c^{inv}(\pi)$ e $c^{inv}_{odd}(\pi)$ para denotar o número de componentes conexos e o número de componentes conexos ímpares de $G^{inv}(\pi)$, respectivamente. Uma aresta e de $G^{inv}(\pi)$ cuja remoção aumenta $c^{inv}(\pi)$ é chamada de *aresta de corte*. O número de vértices não isolados de $G^{inv}(\pi)$ é denotado por $\gamma(\pi)$. Os vértices não isolados de $G^{inv}(\pi)$ são aqueles que pertencem a pelo menos uma inversão. A Figura 4.1 mostra um exemplo de um grafo de inversões.

Além de inversões e do grafo de inversões, o conceito de entropia também é usado no estudo de problemas de ordenação por operações curtas.

Definição 32. *A* entropia *do elemento π_i é definida como* $\text{ent}(\pi_i) = ||\pi_i| - i|$*, para $1 \leq i \leq n$. Em outras palavras, a entropia de um elemento é igual a distância entre a sua posição em π e a sua posição em ι.*

Dada uma permutação com sinais π, definimos os conjuntos $E_\pi^{even^-} = \{\pi_i : \pi_i < 0 \text{ e } \text{ent}(\pi_i) \text{ é par}\}$ e $E_\pi^{odd^+} = \{\pi_i : \pi_i > 0 \text{ e } \text{ent}(\pi_i) \text{ é ímpar}\}$.

Exemplo 12. *Para* $\pi = (+5\ -4\ +3\ -1\ +2)$, *temos* $\text{ent}(+5) = 4$, $\text{ent}(-4) = 2$, $\text{ent}(+3) = 0$, $\text{ent}(-1) = 3$, *e* $\text{ent}(+2) = 3$, *e os conjuntos* $E_\pi^{even^-} = \{-4\}$ *e* $E_\pi^{odd^+} = \{+2\}$.

Lema 47 (Galvão *et al.* [16])**.** *Para qualquer permutação com sinais* π *e reversão super curta* $\bar{\rho}$, *temos que* $|E_\pi^{even^-}| + |E_\pi^{odd^+}| = |E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|$, *onde* $\pi' = \pi \circ \bar{\rho}$.

4.2 Relação com a Abordagem Não Ponderada

Os próximos lemas apresentam limitantes para $d_M^\ell(\pi)$ a partir de $d_M(\pi)$, onde M é um modelo de rearranjo contendo apenas operações curtas.

Lema 48. *Para qualquer permutação sem sinais* π, $d_{sr}^\ell(\pi) \geq 2d_{sr}(\pi)$, $d_{st}^\ell(\pi) \geq 2d_{st}(\pi)$ *e* $d_{srt}^\ell(\pi) \geq 2d_{srt}(\pi)$.

Demonstração. Considere o problema SbLWsR e uma permutação π qualquer. Observe que uma sequência de ordenação ótima para um problema ponderado também é uma sequência de ordenação válida para o problema não ponderado correspondente. Seja S uma sequência de ordenação ótima para SbLWsR. Pela definição do problema não ponderado, sabemos que $|S| \geq d_{sr}(\pi)$. Como qualquer reversão sem sinais tem tamanho maior ou igual a 2, o custo de S é maior ou igual a $2d_{sr}(\pi)$. Portanto, $d_{sr}^\ell(\pi) \geq 2d_{sr}(\pi)$.

A demonstração é similar para os outros problemas. □

Lema 49. *Para qualquer permutação com sinais* π, $d_{s\bar{r}}^\ell(\pi) \geq d_{s\bar{r}}(\pi)$ *e* $d_{s\bar{r}t}^\ell(\pi) \geq d_{s\bar{r}t}(\pi)$.

Demonstração. Similar à prova do Lema 48. Observe que uma reversão que apenas troca o sinal de um elemento tem tamanho 1. □

Lema 50. *Para qualquer permutação* π, $d_M^\ell(\pi) \leq 3d_M(\pi)$, *para todo* $M \in \{sr, s\bar{r}, st, srt, s\bar{r}t\}$.

Demonstração. Considere o problema SbLWsR e uma permutação π qualquer. Observe que uma sequência de ordenação ótima para um problema não ponderado também é uma sequência de ordenação válida para o problema ponderado correspondente. Seja S uma sequência de ordenação ótima para o problema não ponderado. Como o custo máximo de uma operação curta é igual a 3, o custo de S é menor ou igual a $3|S| = 3d_{sr}(\pi)$. Portanto, $d_{sr}^\ell(\pi) \leq 3d_{sr}(\pi)$. □

A partir desses limitantes, os próximos lemas apresentam fatores de aproximação para os problemas estudados neste capítulo.

Lema 51. *Considerando os problemas SbLWsR, SbLWsT e SbLWsRsT, temos que uma* α*-aproximação para a versão não ponderada do problema é uma* 1.5α*-aproximação para a versão ponderada pelo tamanho.*

Demonstração. Segue diretamente dos lemas 48 e 50. □

Tabela 4.1: Fatores de aproximação dos problemas com operações curtas ponderadas pelo tamanho usando os melhores algoritmos conhecidos para a versão não ponderada.

	Melhor Algoritmo	
	Problema Não Ponderado	**Problema Ponderado**
SbLWsR	2-aproximação [20]	3-aproximação (Lema 51)
SbLWsR̄	5-aproximação [16]	15-aproximação (Lema 52)
SbLWsT	5/4-aproximação [22]	15/8-aproximação (Lema 51)
SbLWsRsT	2-aproximação [34]	3-aproximação (Lema 51)
SbLWsR̄sT	3-aproximação [16]	9-aproximação (Lema 52)

Lema 52. *Considerando os problemas SbLWsR̄ e SbLWsR̄sT, temos que uma α-aproximação para a versão não ponderada do problema é uma 3α-aproximação para a versão ponderada pelo tamanho.*

Demonstração. Segue diretamente dos lemas 49 e 50. □

A partir dos resultados dos lemas 51 e 52, a Tabela 4.1 apresenta os melhores fatores de aproximação obtidos ao adaptar os algoritmos da versão não ponderada.

4.3 Algoritmos de Aproximação para Permutações sem Sinais

Nesta seção, apresentamos algoritmos gulosos de 4/3-aproximação para SbLWsT e de 2-aproximação para SbLWsR e SbLWsRsT. Além disso, também é apresentado um outro algoritmo com melhor fator de aproximação para o problema SbLWsT, considerando permutações com muitas inversões. Para demonstrar os fatores de aproximação, os lemas 53 e 54 mostram limitantes no valor $\frac{\Delta\text{Inv}(\pi,\beta)}{|\beta|}$, para qualquer operação curta β. Com esses resultados, os lemas 55 e 56 dão limitantes inferiores relacionando a distância de ordenação com o número de inversões em uma permutação.

Lema 53. *Para qualquer permutação π e qualquer reversão curta ρ, $\frac{\Delta\text{Inv}(\pi,\rho)}{|\rho|} \leq 1$.*

Demonstração. Segue diretamente do Lema 46. □

Lema 54. *Para qualquer permutação π e qualquer transposição curta τ, $\frac{\Delta\text{Inv}(\pi,\tau)}{|\tau|} \leq \frac{2}{3}$.*

Demonstração. Segue diretamente do Lema 46. □

Lema 55. *Para qualquer permutação π, $d^{\ell}_{sr}(\pi) \geq d^{\ell}_{srt}(\pi) \geq \text{Inv}(\pi)$.*

Demonstração. Pelos lemas 53 e 54, qualquer operação β possui razão $\frac{\Delta\text{Inv}(\pi,\beta)}{|\beta|} \leq 1$. Isso significa que remover uma inversão de π custa pelo menos 1. Como apenas a permutação identidade possui $\text{Inv}(\iota) = 0$, qualquer sequência de ordenação deve remover

Algoritmo 6: Algoritmo genérico para 2-sR, 4/3-sT e 2-sRsT

Entrada: modelo M, permutação π e o seu tamanho n
Saída: custo para ordenar π

```
1 sort(M, π, n) :
2     d ← 0
3     enquanto Inv(π) > 0 faça
4         Seja β uma operação de M, tal que ΔInv(π,β)/|β| ≥ ΔInv(π,β')/|β'| para todo β' ∈ M
5         π ← π ∘ β
6         d ← d + |β|
7     retorne d
```

$\mathrm{Inv}(\pi)$ inversões para ordenar a permutação π, resultando nos limitantes $d^{\ell}_{sr}(\pi) \geq \mathrm{Inv}(\pi)$ e $d^{\ell}_{srt}(\pi) \geq \mathrm{Inv}(\pi)$. Além disso, temos que $d^{\ell}_{sr}(\pi) \geq d^{\ell}_{srt}(\pi)$ pois qualquer sequência de ordenação para SbLWsR também é uma sequência de ordenação válida para SbLWsRsT. □

Lema 56. *Para qualquer permutação π, $d^{\ell}_{st}(\pi) \geq \frac{3}{2}\mathrm{Inv}(\pi)$.*

Demonstração. Similar à prova do Lema 55. □

Os algoritmos de aproximação são nomeados 2-sR, 4/3-sT e 2-sRsT para os problemas SbLWsR, SbLWsT e SbLWsRsT, respectivamente. Enquanto existem inversões na permutação π, a estratégia desses algoritmos é aplicar a operação com melhor razão $\frac{\Delta\mathrm{Inv}(\pi,\beta)}{|\beta|}$. A partir do Lema 45, sabemos que sempre existe pelo menos uma operação curta com $\frac{\Delta\mathrm{Inv}(\pi,\beta)}{|\beta|} > 0$ e, portanto, essa estratégia encontra uma sequência S que transforma π em ι. O Algoritmo 6 apresenta um pseudocódigo genérico para 2-sR, 4/3-sT e 2-sRsT.

Como o número de inversões em uma permutação π é no máximo $\binom{n}{2}$, esses algoritmos executam $O(n^2)$ iterações. Cada iteração gasta tempo $O(n)$ para escolher uma operação dentre todas as possíveis. Portanto, 2-sR, 4/3-sT e 2-sRsT possuem complexidade de tempo de $O(n^3)$. Observe que podemos calcular a variação no número de inversões em tempo constante, já que no máximo três elementos são afetados.

Os próximos teoremas demonstram o fator de aproximação para 2-sR, 4/3-sT e 2-sRsT.

Teorema 11. *2-sR e 2-sRsT são algoritmos de 2-aproximação para SbLWsR e SbLWsRsT, respectivamente.*

Demonstração. Se $\mathrm{Inv}(\pi) = 0$, então a permutação π já está ordenada. Se $\mathrm{Inv}(\pi) > 0$, então existe uma inversão (π_i, π_{i+1}) (Lema 45). Assim, a transposição $\tau(i, i+1, i+2)$ ou a reversão $\rho(i, i+1)$ possuem custo 2 e $\Delta\mathrm{Inv}(\pi, \tau(i, i+1, i+2)) = \Delta\mathrm{Inv}(\pi, \rho(i, i+1)) = 1$, já que a inversão (π_i, π_{i+1}) é removida. Como esses algoritmos sempre aplicam a operação que mais reduz o número de inversões usando o menor custo, uma sequência encontrada por eles tem custo menor ou igual a $2\mathrm{Inv}(\pi)$. Pelo Lema 55, 2-sR e 2-sRsT são algoritmos de 2-aproximação para os seus respectivos problemas. □

Teorema 12. *4/3-sT é um algoritmo de 4/3-aproximação para SbLWsT.*

Demonstração. Similar à prova do Teorema 11. □

Algoritmo 7: Algoritmo de $((1+\gamma(\pi)/(3\,\mathrm{Inv}(\pi))))$-aproximação para SbLWsT

Entrada: permutação π e o seu tamanho n
Saída: custo para ordenar π

```
1  ManyInversions-sT(π, n) :
2      d ← 0
3      para i de 1 até n faça
4          se π_i ≠ i então
5              Seja k a posição do elemento i
6              se k − i é par então
7                  π ← π ∘ τ(k−2, k, k+1) ∘ τ(k−4, k−2, k−1) ∘ ... ∘ τ(i, i+2, i+3)
8              senão
9                  π ← π ∘ τ(k−2, k, k+1) ∘ τ(k−4, k−2, k−1) ∘ ... ∘ τ(i, i+1, i+2)
10             d ← d + 3⌊(k−i)/2⌋ + 2 ((k − i) mod 2)
11     retorne d
```

4.3.1 SbLWsT: Aproximação para Permutações com Muitas Inversões

O Algoritmo 7 descreve o algoritmo de aproximação ManyInversions-sT. Para cada elemento $\pi_k = i$, com $1 \leq i \leq n$, ManyInversions-sT aplica uma sequência de transposições curtas para mover o elemento π_k para a sua posição correta. Dessa forma, a permutação π está ordenada no final desse laço. Esse algoritmo tem complexidade de tempo de $O(n^2)$, já que ele possui n iterações e, em cada iteração, $O(n)$ operações curtas são aplicadas em π.

O Lema 57 apresenta um limitante superior para o custo da sequência de ordenação usado por ManyInversions-sT.

Lema 57. *Para qualquer permutação π, ManyInversions-sT acha uma sequência de ordenação com custo menor ou igual a $\frac{3}{2}\mathrm{Inv}(\pi) + \frac{\gamma(\pi)}{2}$.*

Demonstração. Nessa demonstração, analisamos o custo médio para remover uma inversão em cada iteração do algoritmo.

Considere a i-ésima iteração. Seja π a permutação no início dessa iteração e π' a permutação no final dessa iteração. Seja $\pi_k = i$ o elemento movido nessa iteração. Sabemos que $\pi = (1\ 2\ \ldots\ (i-1)\ \pi_i\ \ldots\ \pi_k\ \ldots\ \pi_n)$ e que os pares (π_j, π_k) são inversões, para todo $i \leq j < k$, já que $j < k$ e $\pi_k < \pi_j$. Após mover π_k para a sua posição correta, temos a permutação $\pi' = (1\ 2\ \ldots\ (i-1)\ i\ \pi_i\ \ldots\ \pi_{k-1}\ \pi_{k+1}\ \ldots\ \pi_n)$.

Seja S' a sequência de operações usada pelo algoritmo para mover π_k para a sua posição correta. Se $k-i$ é par, então S' contém $\frac{k-i}{2}$ transposições de tamanho 3 e o seu custo é igual a $\frac{3}{2}(k-i)$. Caso contrário, S' contém $\lfloor\frac{k-i}{2}\rfloor$ transposições de tamanho 3 e uma transposição de tamanho 2, fazendo com que o custo de S' seja igual a $3\lfloor\frac{k-i}{2}\rfloor + 2 = \frac{3}{2}(k-i) + \frac{1}{2}$, já que $k-i$ é ímpar. Como S' remove $k-i$ inversões e não adiciona nenhuma inversão na permutação, concluímos que S' possui um custo médio de $\frac{3}{2}$ para remover uma inversão e, se $k-i$ é ímpar, S' possui um custo adicional de $\frac{1}{2}$. Portanto, o custo da sequência de

ordenação gerada pelo algoritmo é igual a $\frac{3}{2}\mathrm{Inv}(\pi) + \frac{1}{2}m$, onde m é o número de iterações em que uma transposição de tamanho 2 é utilizada.

Como qualquer transposição usada por ManyInversions-sT remove pelo menos uma inversão e nenhuma inversão é adicionada, qualquer elemento isolado em $G^{inv}(\pi)$ não é afetado por S. Então, a permutação π não é alterada em pelo menos $n - \gamma(\pi)$ iterações do algoritmo. Consequentemente, $m \leq \gamma(\pi)$ e S tem custo menor ou igual a $\frac{3}{2}\mathrm{Inv}(\pi) + \frac{1}{2}\gamma(\pi)$. □

Teorema 13. *ManyInversions-sT é uma* $(1+\gamma(\pi)/(3\,\mathrm{Inv}(\pi)))$*-aproximação para SbLWsT.*

Demonstração. Segue diretamente dos lemas 56 e 57. □

Se $\mathrm{Inv}(\pi) \geq cn^2$, onde c é uma constante positiva, então o fator de aproximação do algoritmo ManyInversions-sT é igual a

$$1 + \frac{\gamma(\pi)}{3\,\mathrm{Inv}(\pi)} \leq 1 + \frac{n}{3\,cn^2} = 1 + \frac{1}{3\,cn}.$$

Exemplo 13. *Considere uma permutação* π *com* 200 *elementos e* $\mathrm{Inv}(\pi) = 10000$*. O fator de aproximação de ManyInversions-sT para essa permutação é de aproximadamente* 1.007*.*

Lema 58. *O fator de aproximação do algoritmo ManyInversions-sT é no máximo* $\frac{5}{3}$*.*

Demonstração. O grafo $G^{inv}(\pi)$ possui pelo menos $\frac{\gamma(\pi)}{2}$ arestas. Portanto, o fator de aproximação é igual a

$$1 + \frac{\gamma(\pi)}{3\,\mathrm{Inv}(\pi)} \leq 1 + \frac{\gamma(\pi)}{3\,\gamma(\pi)/2} = 1 + \frac{2}{3} = \frac{5}{3}.$$

□

4.4 Algoritmos de Aproximação para Permutações com Sinais

Nesta seção, descrevemos algoritmos gulosos de 3-aproximação para SbLWsR̄ e SbLWsR̄sT chamados de 3-sR̄ e 3-sR̄sT, respectivamente. Além disso, apresentamos um algoritmo guloso de 7/3-aproximação para SbLWsR̄sT chamado de 7/3-sR̄sT.

4.4.1 3-Aproximação para SbLWsR̄ e SbLWsR̄sT

Para descrevermos o funcionamento desses algoritmos, definimos a função de pontuação ϕ baseada no grafo de inversões de π e nos conjuntos $E_\pi^{even^-}$ e $E_\pi^{odd^+}$.

Algoritmo 8: Algoritmo genérico para 3-s$\bar{\text{R}}$ e 3-s$\bar{\text{R}}$sT

Entrada: modelo M, permutação π e o seu tamanho n
Saída: custo para ordenar π

```
1 sort(M, π, n) :
2     d ← 0
3     enquanto π ≠ ι faça
4         Seja β uma operação de M, tal que φ(π, β) ≥ φ(π, β') para todo β' ∈ M
5         π ← π ∘ β
6         d ← d + |β|
7     retorne d
```

Definição 33. *Dada uma permutação com sinais π e uma sequência S, seja $\pi' = \pi \circ S$. A função de pontuação é definida como*

$$\phi(\pi, S) = \frac{\left(\sum_{v \in G^{inv}(\pi)} d(v) + |E_\pi^{even^-}| + |E_\pi^{odd^+}|\right) - \left(\sum_{v \in G^{inv}(\pi')} d(v) + |E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|\right)}{\sum_{\beta \in S} |\beta|}$$

$$= \frac{(2\,\mathrm{Inv}(\pi) + |E_\pi^{even^-}| + |E_\pi^{odd^+}|) - (2\,\mathrm{Inv}(\pi') + |E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|)}{\sum_{\beta \in S} |\beta|},$$

já que $\sum_{v \in G^{inv}(\pi)} d(v) = 2\mathrm{Inv}(\pi)$*, para qualquer permutação* π*.*

A permutação identidade é a única permutação com $\mathrm{Inv}(\iota) = |E_\iota^{even^-}| = |E_\iota^{odd^+}| = 0$. Dada uma permutação com sinais $\pi \neq \iota$, se S é uma sequência de ordenação ótima, então $\phi(\pi, S) \geq \phi(\pi, S')$, para qualquer sequência de ordenação S'. Sendo assim, enquanto a permutação π não está ordenada, a escolha gulosa de 3-s$\bar{\text{R}}$ e 3-s$\bar{\text{R}}$sT é aplicar a operação curta β com maior pontuação $\phi(\pi, \beta)$. O Algoritmo 8 apresenta o pseudocódigo genérico para 3-s$\bar{\text{R}}$ e 3-s$\bar{\text{R}}$sT. Os próximos lemas demonstram limitantes para a função de pontuação ϕ.

Os algoritmos 3-s$\bar{\text{R}}$ e 3-s$\bar{\text{R}}$sT possuem complexidade de tempo de $O(n^3)$. Essa análise é similar à usada para os algoritmos sem sinais. Notamos que a função de pontuação ϕ pode ser calculada em tempo constante para qualquer operação curta, já que no máximo três elementos são afetados.

Lema 59. *Para qualquer permutação com sinais π e qualquer reversão com sinais curta $\bar{\rho}$, temos que $\phi(\pi, \bar{\rho}) \leq 3$.*

Demonstração. Seja $\pi' = \pi \circ \bar{\rho}$ e $\bar{\rho} = \bar{\rho}(i, j)$. Observe que $\Delta\mathrm{Inv}(\pi, \bar{\rho}) = \mathrm{Inv}(\pi) - \mathrm{Inv}(\pi')$. Podemos dividir a demonstração nos seguintes casos:

1. Se $|\bar{\rho}| = 1$, então o número de inversões em π' é o mesmo que em π, já que uma 1-reversão apenas muda o sinal do elemento π_i. Como a paridade de $\mathrm{ent}(\pi_i)$ não

é alterada, temos que $(|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}|) - (|E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|) \leq 1$. Portanto, $\phi(\pi, \bar{\rho}) \leq 1$.

2. Se $|\bar{\rho}| = 2$, então $\bar{\rho}$ pode remover no máximo uma inversão e, consequentemente, $2(\Delta\text{Inv}(\pi, \bar{\rho})) \leq 2$ (Lema 46). Além disso, pelo Lema 47, temos que $(|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}|) - (|E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|) = 0$. Portanto, $\phi(\pi, \bar{\rho}) \leq 1$.

3. Se $|\bar{\rho}| = 3$, então $\bar{\rho}$ pode remover no máximo três inversões e, consequentemente, $2(\Delta\text{Inv}(\pi, \bar{\rho})) \leq 6$. Além disso, temos que $(|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}|) - (|E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|) \leq 3$, já que apenas três elementos são afetados por $\bar{\rho}$ e $E_{\pi}^{even^-} \cap E_{\pi}^{odd^+} = \emptyset$. Portanto, $\phi(\pi, \bar{\rho}) \leq \frac{9}{3} = 3$.

Em qualquer caso, temos que $\phi(\pi, \bar{\rho}) \leq 3$. □

Lema 60. *Para qualquer permutação com sinais π e qualquer transposição curta τ, temos que $\phi(\pi, \tau) \leq \frac{7}{3}$.*

Demonstração. Seja $\pi' = \pi \circ \tau$. Podemos dividir a demonstração nos seguintes casos:

1. Se $|\tau| = 2$, então τ pode remover no máximo uma inversão. Além disso, temos que $(|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}|) - (|E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|) \leq 2$, já que apenas dois elementos são afetados. Portanto, $\phi(\pi, \tau) \leq \frac{4}{2} = 2$.

2. Se $|\tau| = 3$, então τ pode remover no máximo duas inversões. Além disso, temos que $(|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}|) - (|E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|) \leq 3$, já que apenas três elementos são afetados. Portanto, $\phi(\pi, \tau) \leq \frac{7}{3}$.

Em qualquer caso, temos que $\phi(\pi, \tau) \leq \frac{7}{3}$. □

Lema 61. *Para qualquer permutação com sinais $\pi \neq \iota$, existe reversão com sinais curta $\bar{\rho}$ com $\phi(\pi, \bar{\rho}) \geq 1$.*

Demonstração. Seja $\pi' = \pi \circ \bar{\rho}$. Se $\text{Inv}(\pi) = 0$, então $\text{ent}(\pi_i) = 0$, para todo $1 \leq i \leq n$, e $|E_{\pi}^{odd^+}| = 0$. Como $\pi \neq \iota$, existe um elemento $\pi_i < 0$ em $E_{\pi}^{even^-}$. Após a aplicação de $\bar{\rho}(i, i)$ em π, o elemento π_i se torna positivo em π' e, consequentemente, $|E_{\pi}^{even^-}| - |E_{\pi'}^{even^-}| = 1$. Portanto, $\phi(\pi, \bar{\rho}) = 1$.

Se $\text{Inv}(\pi) > 0$, então existe uma inversão (π_i, π_{i+1}) em π (Lema 45). Desse modo, a 2-reversão com sinais $\bar{\rho}(i, i+1)$ remove essa inversão, resultando em $2\text{Inv}(\pi) = 2$ e $|E_{\pi}^{even^-}| + |E_{\pi}^{odd^+}| = |E_{\pi'}^{even^-}| + |E_{\pi'}^{odd^+}|$ (Lema 47). Portanto, $\phi(\pi, \bar{\rho}) = 1$. □

Teorema 14. *3-s$\bar{R}$ e 3-s$\bar{R}$sT são algoritmos de 3-aproximação para SbLWs$\bar{R}$ e SbLWs$\bar{R}$sT, respectivamente.*

Demonstração. Como 3-s$\bar{\text{R}}$ sempre busca a reversão com sinais curta β com maior pontuação $\phi(\pi, \beta)$, a cada iteração o algoritmo encontra uma reversão com sinais curta $\bar{\rho}$ com $\phi(\pi, \bar{\rho}) \geq 1$ (Lema 61). O valor de $\phi(\pi, S)$, para qualquer sequência de ordenação S, é menor ou igual a 3 e o algoritmo encontra uma sequência com pontuação maior ou igual a 1, o que garante um fator de aproximação de 3.

A prova é análoga para 3-s$\bar{\text{R}}$sT. □

Algoritmo 9: Algoritmo 7/3-sR̄sT

Entrada: modelo M, permutação π e o seu tamanho n
Saída: custo para ordenar π

```
7/3-sR̄sT(M, π, n) :
    d ← 0
    enquanto π ≠ ι faça
        Seja β uma operação de M, tal que ψ(π, β) ≥ ψ(π, β') para todo β' ∈ M
        π ← π ∘ β
        d ← d + |β|
    retorne d
```

4.4.2 7/3-Aproximação para SbLWsR̄sT

Para descrevermos o funcionamento desse algoritmo, definimos uma nova função de pontuação que usa apenas o grafo de inversões de π.

Definição 34. *Dada uma permutação com sinais π e uma sequência S, seja $\pi' = \pi \circ S$. A função de pontuação é definida como*

$$\begin{aligned}\psi(\pi, S) &= \frac{\left(\sum_{v \in G^{inv}(\pi)} d(v) + c^{inv}_{odd}(\pi)\right) - \left(\sum_{v \in G^{inv}(\pi')} d(v) + c^{inv}_{odd}(\pi')\right)}{\sum_{\beta \in S} |\beta|} \\ &= \frac{(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi'))}{\sum_{\beta \in S} |\beta|},\end{aligned}$$

já que $\sum_{v \in G^{inv}(\pi)} d(v) = 2\mathrm{Inv}(\pi)$*, para qualquer permutação π.*

A permutação identidade é a única permutação com $\mathrm{Inv}(\iota) = c^{inv}_{odd}(\iota) = 0$. Sendo assim, enquanto a permutação π não está ordenada, a escolha gulosa de 7/3-sR̄sT é aplicar a operação curta β com maior pontuação $\psi(\pi, \beta)$. O Algoritmo 9 apresenta o pseudocódigo para 7/3-sR̄. Os próximos lemas demonstram limitantes para a função de pontuação ψ.

O algoritmo 7/3-sR̄ possui complexidade de tempo de $O(n^4)$. Essa análise é similar à usada para os algoritmos sem sinais, exceto que é preciso de tempo linear para calcular a variação no número de componentes ímpares causado por uma operação curta [16].

Lema 62. *Para qualquer permutação com sinais π e qualquer operação curta β, temos que $\psi(\pi, \beta) \leq \frac{7}{3}$.*

Demonstração. Seja $\pi' = \pi \circ \beta$. Podemos dividir a demonstração nos seguintes casos:

1. β é uma reversão com sinais de tamanho 1. Nesse caso, o número de inversões em π' é o mesmo que em π. Além disso, como apenas o componente contendo o elemento afetado pode se tornar par, temos que $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 1$. Portanto, $\psi(\pi, \bar{\rho}) \leq 1$.

2. β possui tamanho 2. Sejam π_i e π_{i+1} os elementos afetados por β. Dividimos nossa análise nos seguintes subcasos:

   (a) Se (π_i, π_{i+1}) é uma inversão, então β remove a inversão (π_i, π_{i+1}) e, consequentemente, $2(\Delta\mathrm{Inv}(\pi, \beta)) = 2$. Além disso, note que π_i e π_{i+1} pertencem ao mesmo componente de $G^{inv}(\pi)$. Portanto, β afeta apenas um componente e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 1$.

   (b) Se (π_i, π_{i+1}) não é uma inversão, então β adiciona a inversão (π'_i, π'_{i+1}) e, como π_i e π_{i+1} pertencem a componentes distintos de $G^{inv}(\pi)$, β afeta apenas dois componentes de $G^{inv}(\pi)$. Portanto, temos que $2(\Delta\mathrm{Inv}(\pi, \beta)) = -2$ e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 2$.

   Em ambos os casos, é verdade que $(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi')) \leq 3$. Portanto, $\psi(\pi, \beta) \leq \frac{3}{2}$.

3. β é um rearranjo de tamanho 3. Sejam π_i, π_{i+1} e π_{i+2} os elementos afetados por β. Dividimos nossa análise nos seguintes subcasos:

   (a) Se todos os elementos afetados pertencem ao mesmo componente de $G^{inv}(\pi)$, então $2(\Delta\mathrm{Inv}(\pi, \beta)) \leq 6$ (Lema 46) e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 1$.

   (b) Se dois elementos de $\{\pi_i, \pi_{i+1}, \pi_{i+2}\}$ pertencem ao mesmo componente C_1 e o elemento restante pertence a um outro componente C_2 de $G^{inv}(\pi)$, então $2(\Delta\mathrm{Inv}(\pi, \beta)) \leq 2$, já que existe apenas uma aresta entre esses três elementos. Além disso, β afeta apenas dois componentes de $G^{inv}(\pi)$, fazendo com que $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 2$.

   (c) Se todos os elementos afetados pertencem a componentes distintos de $G^{inv}(\pi)$, então $2(\Delta\mathrm{Inv}(\pi, \beta)) \leq -4$ e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') \leq 3$. Note que uma transposição curta $\tau(i, j, i+3)$, com $i < j < i+3$, adiciona duas inversões na permutação e uma reversão com sinais curta $\bar{\rho}(i, i+2)$ adiciona três inversões na permutação.

   Em qualquer caso, é verdade que $(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi')) \leq 7$. Portanto, $\psi(\pi, \beta) \leq \frac{7}{3}$.

Em todos os casos, temos que $\psi(\pi, \beta) \leq \frac{7}{3}$. □

Lema 63. *Para qualquer permutação com sinais $\pi \neq \iota$, existe operação curta β com $\psi(\pi, \beta) \geq 1$.*

Demonstração. Seja $\pi' = \pi \circ \beta$. Se $\mathrm{Inv}(\pi) = 0$, então não existem arestas em $G^{inv}(\pi)$ e cada componente de $G^{inv}(\pi)$ contém um único vértice. Como $\pi \neq \iota$, existe um componente ímpar $C = \{\pi_i\}$ e, após a aplicação de $\beta = \bar{\rho}(i, i)$ em π, o componente C se torna par. Portanto, $(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi')) = 1$ e $\psi(\pi, \beta) \geq 1$.

Se $\mathrm{Inv}(\pi) > 0$, então existe uma aresta $e = (\pi_i, \pi_{i+1})$ em $G^{inv}(\pi)$ (Lema 45). Seja C o componente que contém e. Suponha que e não é uma aresta de corte. Então, $\beta = \tau(i, i+1, i+2)$ remove a inversão (π_i, π_{i+1}) e, como e não é uma aresta de corte, os componentes de $G^{inv}(\pi')$ são os mesmos componentes de $G^{inv}(\pi)$. Além disso, a paridade

dos componentes permanece a mesma, pois uma transposição não altera os sinais dos elementos. Portanto, $(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi')) = 2$ e $\psi(\pi, \beta) \geq 1$.

Agora, suponha que e é uma aresta de corte. Sejam C_1 e C_2 os componentes de $C - e$. Como e é uma aresta de corte, os elementos π_i e π_{i+1} pertencem a componentes distintos de $C - e$. Dividimos a demonstração nos seguintes subcasos:

1. Se C_1 e C_2 são ambos pares, então $2(\Delta\mathrm{Inv}(\pi, \beta)) = 2$ e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') = 0$ para $\beta = \tau(i, i+1, i+2)$. Note que C também é par e a paridade dos componentes C_1 e C_2 permanece a mesma.

2. Se C_1 e C_2 são ambos ímpares, então $2(\Delta\mathrm{Inv}(\pi, \beta)) = 2$ e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') = 0$ para $\beta = \bar{\rho}(i, i+1)$. Note que C é par e os componentes C_1 e C_2 se tornam pares depois da aplicação da reversão com sinais $\bar{\rho}(i, i+1)$ em π.

3. Se C_1 e C_2 possuem paridades distintas, então $2(\Delta\mathrm{Inv}(\pi, \beta)) = 2$ e $c^{inv}_{odd}(\pi) - c^{inv}_{odd}(\pi') = 0$ para $\beta = \tau(i, i+1, i+2)$. Note que C é ímpar e a paridade dos componentes C_1 e C_2 permanece a mesma.

Em todos os casos, temos que $(2\,\mathrm{Inv}(\pi) + c^{inv}_{odd}(\pi)) - (2\,\mathrm{Inv}(\pi') + c^{inv}_{odd}(\pi')) = 2$. Portanto, existe operação curta β com $\psi(\pi, \beta) \geq 1$. □

Teorema 15. *7/3-$s\bar{R}sT$ é um algoritmo de 7/3-aproximação para SbLWs$\bar{R}sT$.*

Demonstração. Similar à prova do Teorema 14. Note que essa demonstração utiliza os limitantes definidos nos lemas 62 e 63. □

4.5 Resultados Experimentais

Todos os algoritmos apresentados nas seções 4.3 e 4.4 foram implementados na linguagem de programação C e executados usando as instâncias dos conjuntos $\mathrm{SR01}^M_n$, $\mathrm{SR02}^M_n$, $\mathrm{SR03}^M_n$ e $\mathrm{SR04}^M_n$. Cada conjunto $\mathrm{SR0X}^M_n$ possui 1000 permutações de tamanho n geradas a partir da aplicação de uma quantidade estipulada de operações aleatórias na permutação identidade ι, sendo que cada operação utilizada pertence a M. A quantidade estipulada de operações aleatórias foi de $\lceil \sqrt{n} \rceil$ para $\mathrm{SR01}^M_n$, $\lceil \frac{n}{2} \rceil$ para $\mathrm{SR02}^M_n$, n para $\mathrm{SR03}^M_n$, e n^2 para $\mathrm{SR04}^M_n$.

Para cada tipo de instância, foram utilizados valores de $n \in \{10, 15, 20, \ldots, 495, 500\}$ e $M \in \{sr, st, srt, s\bar{r}, s\bar{r}t\}$. As permutações possuem sinais quando M é igual a $s\bar{r}$ ou $s\bar{r}t$.

Os resultados dos experimentos são apresentados nas figuras 4.2 a 4.8. Para calcular a qualidade dos algoritmos, utilizamos os limitantes inferiores apresentados neste capítulo.

Para os algoritmos 2-sR (Figura 4.2), 3-s$\bar{\mathrm{R}}$ (Figura 4.3) e 3-s$\bar{\mathrm{R}}$sT (Figura 4.4), os experimentos indicam que, em média, a qualidade para esses algoritmos é melhor para os conjuntos em que são aplicadas poucas operações aleatórias. Considerando cada tipo de instância, houve pouca variação na média das qualidades para os conjuntos $\mathrm{SR02}^M_n$ e $\mathrm{SR03}^M_n$. Para o conjunto $\mathrm{SR01}^M_n$, observamos uma melhora na média das qualidades à medida que o tamanho da permutação cresce. Já para o conjunto $\mathrm{SR04}^M_n$, observamos que a média das qualidades piora à medida que o tamanho da permutação cresce.

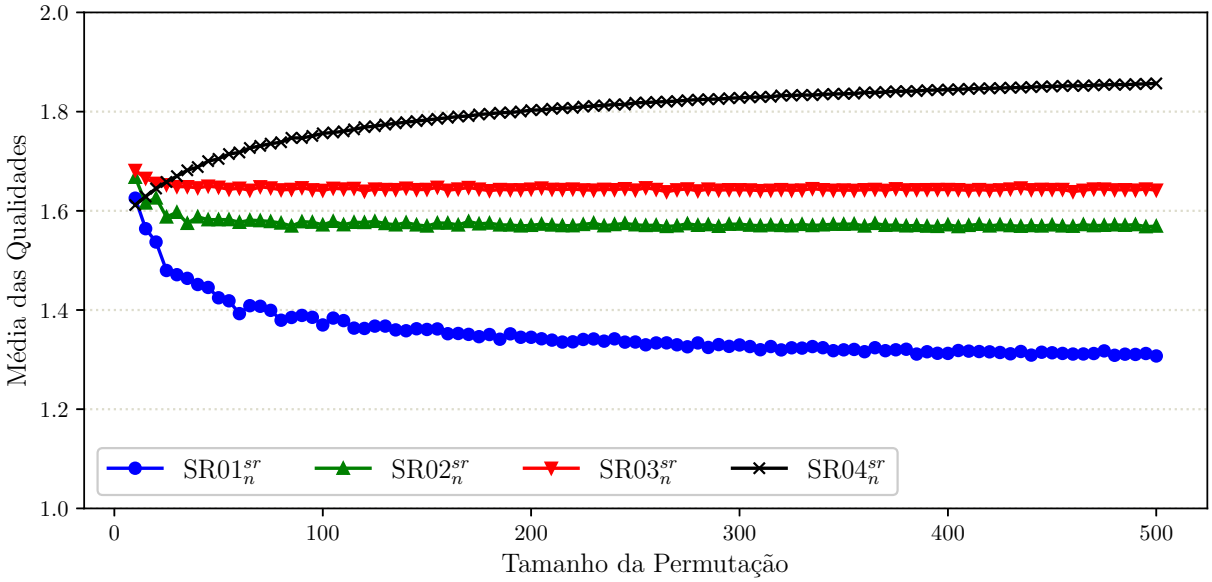


Figura 4.2: Média das qualidades para o algoritmo 2-sR.

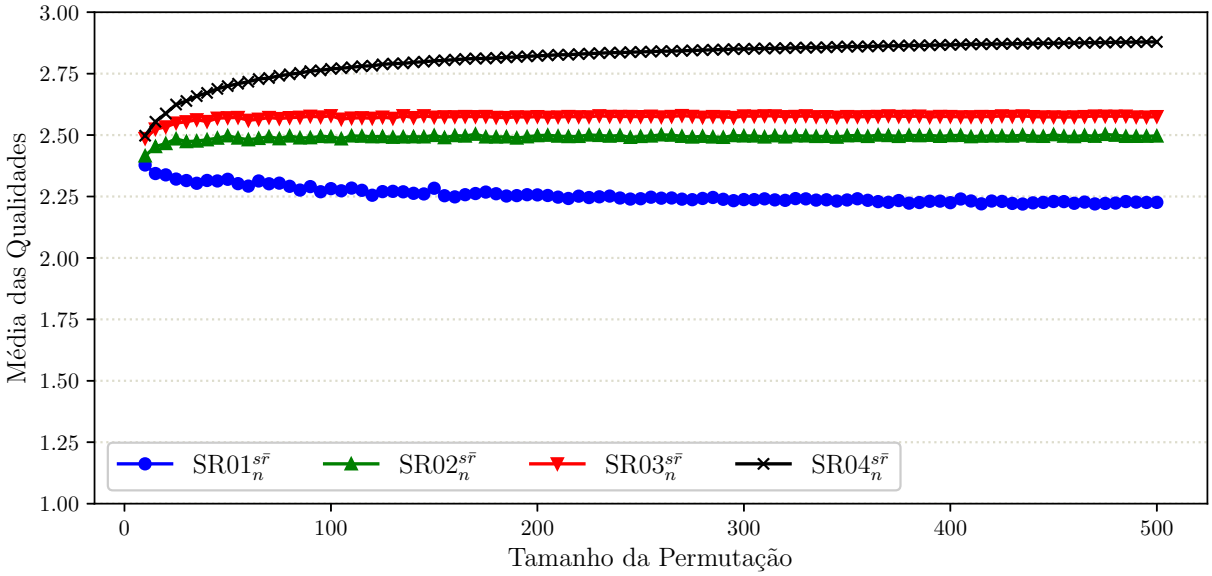


Figura 4.3: Média das qualidades para o algoritmo 3-s$\bar{\text{R}}$.

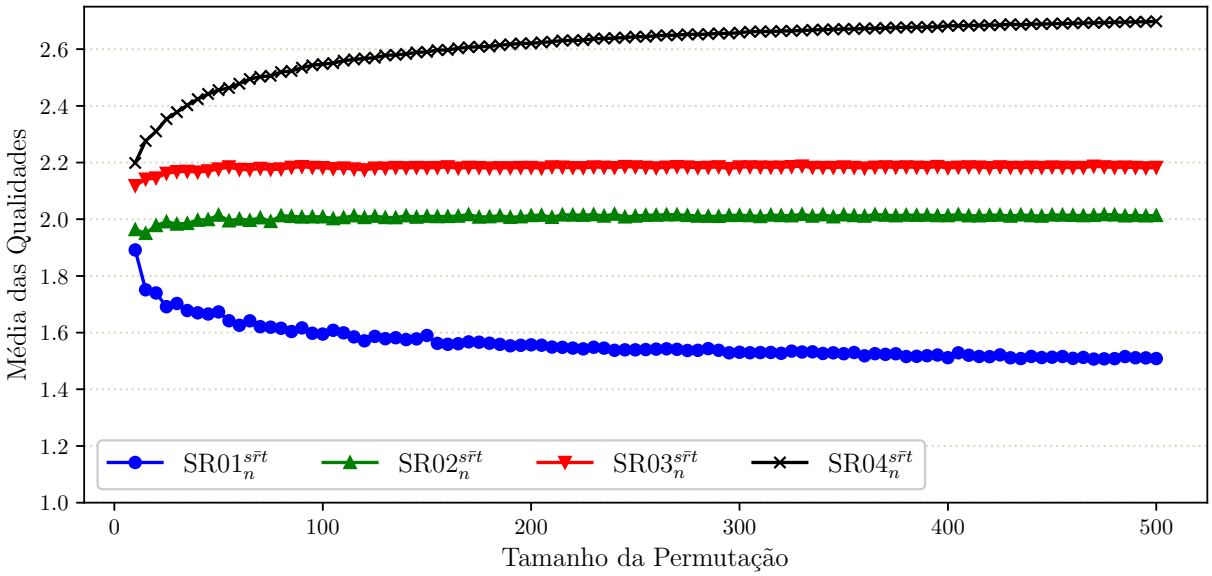


Figura 4.4: Média das qualidades para o algoritmo 3-sR̄sT.

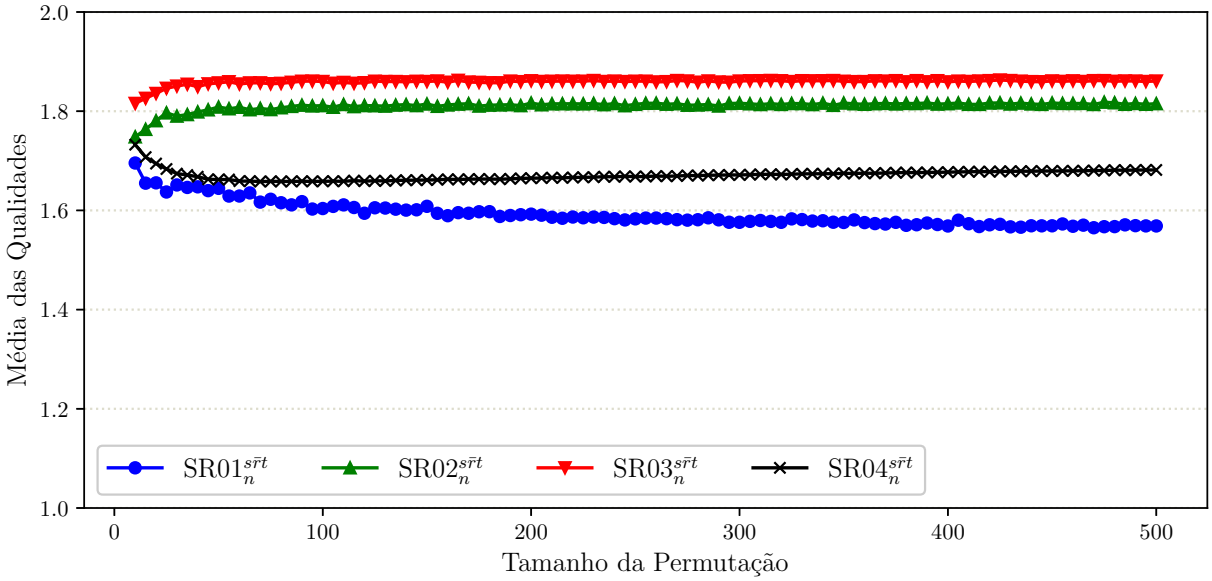


Figura 4.5: Média das qualidades para o algoritmo 7/3-sR̄sT.

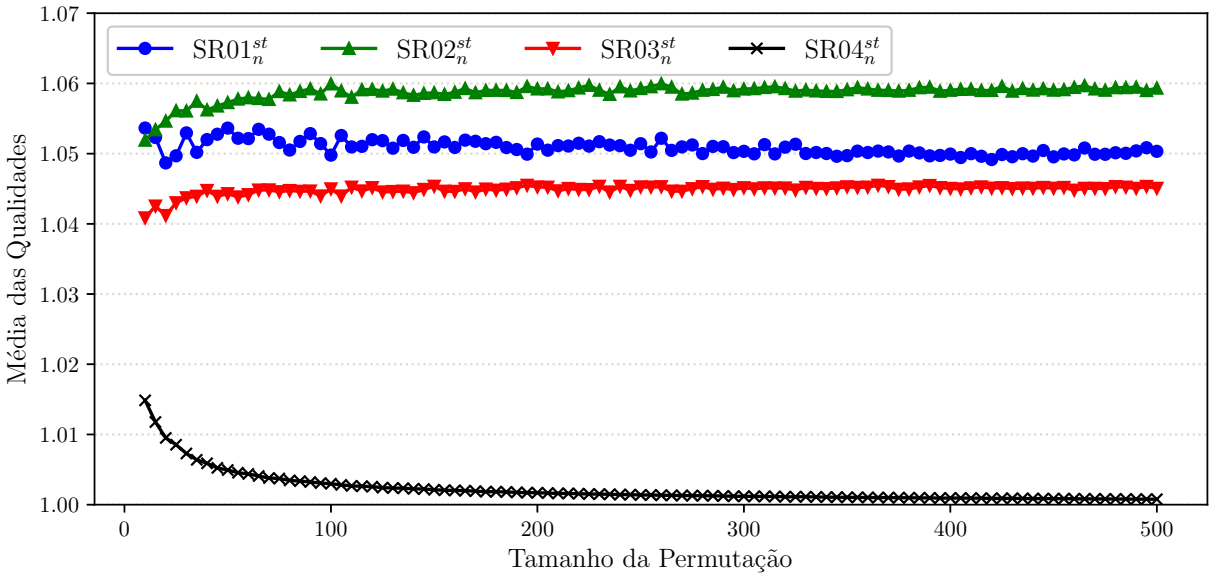


Figura 4.6: Média das qualidades para o algoritmo 4/3-sT.

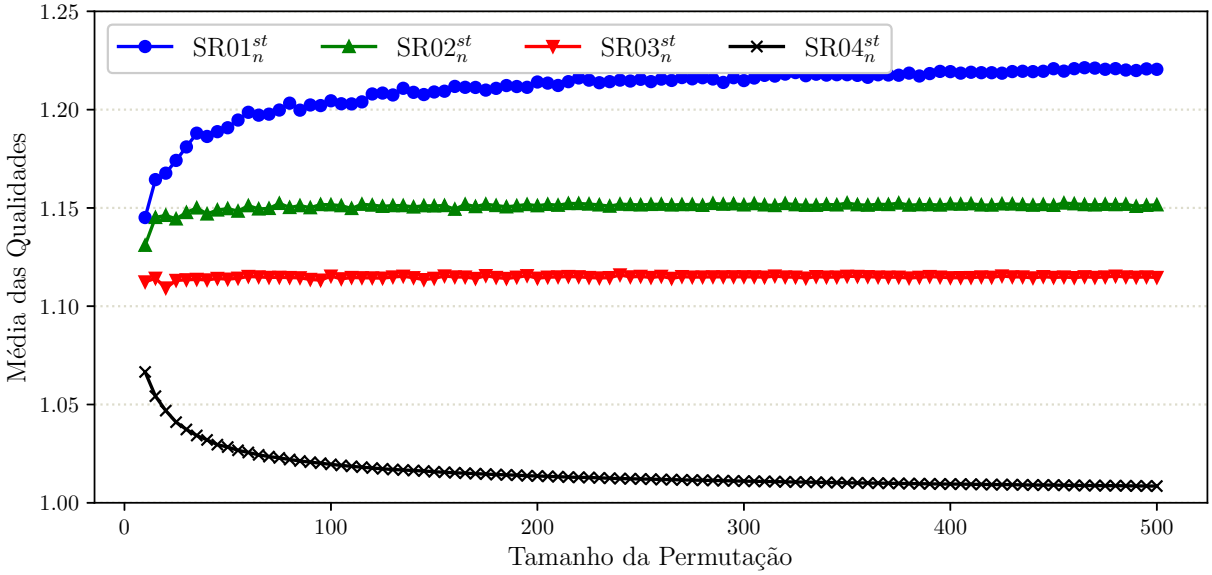


Figura 4.7: Média das qualidades para o algoritmo ManyInversions-sT.

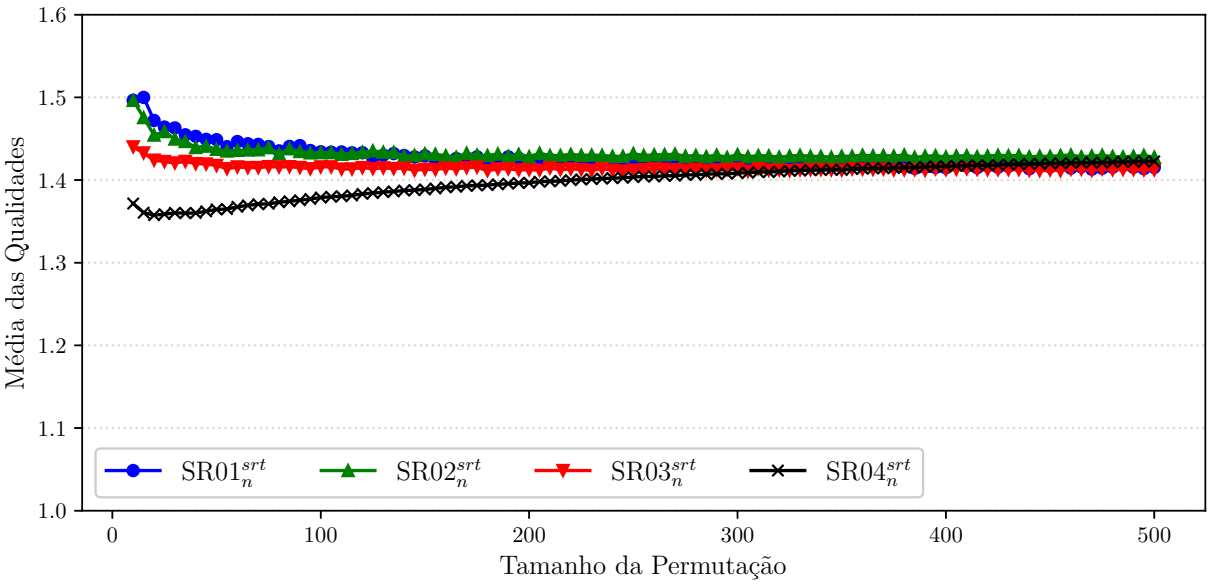


Figura 4.8: Média das qualidades para o algoritmo 2-sRsT.

Considerando o algoritmo 7/3-s$\bar{\text{R}}$sT (Figura 4.5), observamos resultados similares aos descritos anteriormente, com exceção ao comportamento do algoritmo para as instâncias SR04$_n^M$. Nesse caso, 7/3-s$\bar{\text{R}}$sT obteve, em média, melhor qualidade para as instâncias SR04$_n^M$ em comparação com as instâncias SR02$_n^M$ e SR03$_n^M$. Ao analisarmos as sequências de ordenação utilizadas por 7/3-s$\bar{\text{R}}$sT para as instâncias SR04$_n^M$, observamos que o algoritmo conseguiu utilizar muitas transposições de tamanho 3 que removem duas inversões e não alteram a quantidade de componentes ímpares, o que aumentou consideravelmente o valor da função de pontuação. Ao comparar os algoritmos 7/3-s$\bar{\text{R}}$sT (Figura 4.5) e 3-s$\bar{\text{R}}$sT (Figura 4.4), percebemos que 7/3-s$\bar{\text{R}}$sT obteve melhores resultados, com exceção às instâncias de SR01$_n^M$.

Para os algoritmos 4/3-sT (Figura 4.6) e ManyInversions-sT (Figura 4.7), os experimentos indicam que a média das qualidades é melhor para os conjuntos em que são aplicadas muitas operações aleatórias. De forma similar ao que aconteceu com 7/3-s$\bar{\text{R}}$sT para as instâncias SR04$_n^M$, observamos que esses algoritmos conseguiram utilizar muitas transposições de tamanho 3 que removem duas inversões, sendo que a frequência dessas operações aumenta em permutações que sofreram maior alteração. Além disso, a qualidade para esses dois algoritmos foi melhor para as instâncias SR04$_n^M$ à medida que o tamanho das permutações cresce. Apesar do fator teórico de ManyInversions-sT ser melhor do que o de 4/3-sT para permutações com muitas inversões, os resultados indicam que 4/3-sT obteve, em média, melhor qualidade em todos os casos.

Considerando o algoritmo 2-sRsT (Figura 4.8), os resultados foram similares aos dos algoritmos para o problema SbLWsT, com exceção dos resultados para as instâncias SR04$_n^M$. Nesse caso, observamos uma piora na média das qualidades à medida que o tamanho da permutação cresce.

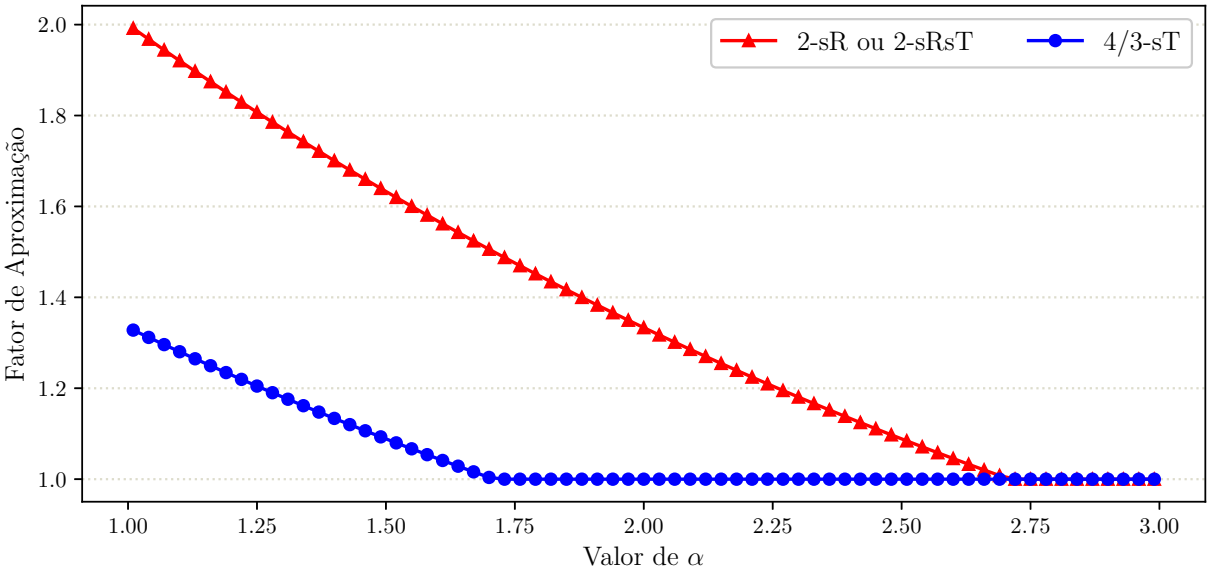


Figura 4.9: Fator de aproximação dos algoritmos para permutações sem sinais considerando a função de custo $|\beta|^\alpha$, para valores de $\alpha > 1$.

4.6 Análise dos Algoritmos para $\alpha > 1$

Nesta seção, investigamos o fator de aproximação dos algoritmos apresentados anteriormente quando o custo de uma operação β é igual a $|\beta|^\alpha$, para $\alpha > 1$. Para isso, os próximos lemas apresentam novos limitantes para os problemas SbLWsR, SbLWsT e SbLWsRsT.

Lema 64. *Para qualquer permutação π, $d^\ell_{sr}(\pi) \geq d^\ell_{srt}(\pi) \geq \min\left(\frac{\text{Inv}(\pi)}{2^\alpha}, \frac{3\text{Inv}(\pi)}{3^\alpha}\right)$.*

Demonstração. Similar à prova do Lema 55. □

Lema 65. *Para qualquer permutação π, $d^\ell_{st}(\pi) \geq \min\left(\frac{\text{Inv}(\pi)}{2^\alpha}, \frac{2\text{Inv}(\pi)}{3^\alpha}\right)$.*

Demonstração. Similar à prova do Lema 56. □

Lema 66. *Para qualquer permutação π, os algoritmos 2-sR, 4/3-sT e 2-sRsT encontram sequências de ordenação S com custo menor ou igual a $\frac{\text{Inv}(\pi)}{2^\alpha}$.*

Demonstração. Similar às provas dos teoremas 11 e 12. □

A partir desses limitantes, a Figura 4.9 apresenta a variação no fator de aproximação dos algoritmos 2-sR, 4/3-sT e 2-sRsT, considerando $1 < \alpha \leq 3$. Observamos que o fator de aproximação melhora à medida que o valor de α cresce. A partir de $\alpha \geq \log_{3/2} 3 \approx 2.71$, os problemas SbLWsR e SbLWsRsT são resolvidos em tempo polinomial por 2-sR e 2-sRsT, respectivamente. Já o problema SbLWsT é resolvido em tempo polinomial por 4/3-sT para $\alpha \geq \log_{3/2} 2 \approx 1.71$. Nesses casos, os três algoritmos sempre utilizarão uma operação super curta que remove uma inversão.

Considerando permutações com sinais, os próximos lemas apresentam novos limitantes para ϕ e ψ.

Lema 67. *Para qualquer permutação com sinais π e qualquer sequência de ordenação S contendo apenas reversões com sinais, temos que $\phi(\pi, S) \leq \max\left(1, \frac{2}{2^\alpha}, \frac{9}{3^\alpha}\right)$.*

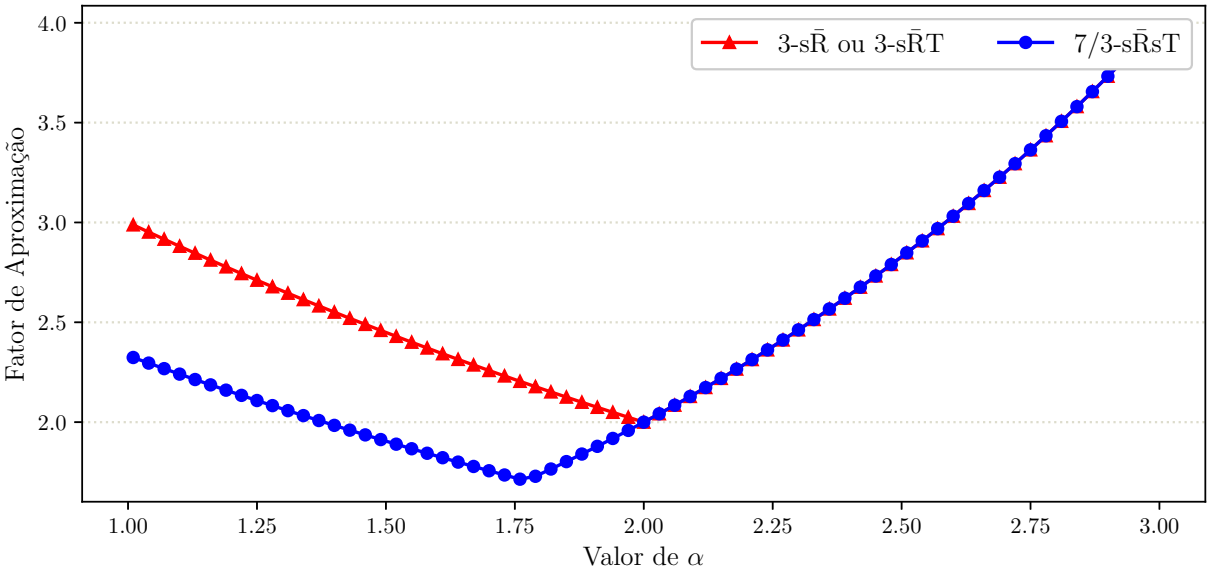


Figura 4.10: Fator de aproximação dos algoritmos para permutações com sinais considerando a função de custo $|\beta|^\alpha$, para valores de $\alpha > 1$.

Demonstração. Similar à prova do Lema 59. □

Lema 68. *Para qualquer permutação com sinais π e qualquer sequência de ordenação S, temos que $\phi(\pi, S) \leq \max\left(1, \frac{4}{2^\alpha}, \frac{9}{3^\alpha}\right)$.*

Demonstração. Similar à prova dos lemas 59 e 60. □

Os limitantes dados pelos lemas 67 e 68 são iguais para qualquer valor de $\alpha > 0$.

Lema 69. *Os algoritmos 3-$s\bar{R}$ e 3-$s\bar{R}sT$ encontram sequência de ordenação S com $\phi(\pi, S) \geq \min\left(1, \frac{2}{2^\alpha}\right)$.*

Demonstração. Similar à prova do Lema 61. □

Lema 70. *Para qualquer permutação com sinais π e qualquer sequência de ordenação S, temos que $\psi(\pi, S) \leq \max\left(1, \frac{3}{2^\alpha}, \frac{7}{3^\alpha}\right)$.*

Demonstração. Similar à prova do Lema 62. □

Lema 71. *O algoritmo 7/3-$s\bar{R}sT$ encontra sequência de ordenação S com $\psi(\pi, S) \geq \min\left(1, \frac{2}{2^\alpha}\right)$.*

Demonstração. Similar à prova do Lema 63. □

A partir desses limitantes para ϕ e ψ, a Figura 4.10 apresenta a variação no fator de aproximação dos algoritmos 3-sR̄, 3-sR̄sT e 7/3-sR̄sT, considerando $1 < \alpha \leq 3$. Para valores de α no intervalo $(1, 2]$, observamos que o fator de aproximação dos algoritmos 3-sR̄ e 3-sR̄sT diminui à medida que α cresce. Para valores de α no intervalo $(2, 3]$, o fator de aproximação aumenta à medida que α cresce. O melhor fator de aproximação ocorre quando $\alpha = 2$ e é igual a 2. Esse aumento acontece pois a pontuação máxima de uma 1-reversão com sinais sempre é a mesma, mas a pontuação máxima de uma operação de

tamanho 2 ou 3 diminui à medida que α cresce. Algo similar acontece para o algoritmo 7/3-s$\bar{\text{R}}$sT, com exceção de que o melhor fator de aproximação para esse algoritmo ocorre quando $\alpha = \log_3 7$ e é de $2^{\log_3(7/3)} \approx 1.71$.

A seguir, apresentamos um algoritmo polinomial para o problema SbLWs$\bar{\text{R}}$ quando $\alpha \geq 3$.

Lema 72. *Para $\alpha \geq 3$ e qualquer 3-reversão com sinais $\bar{\rho}$, existe uma sequência de reversões com sinais super curtas S, tal que $\pi \circ S = \pi \circ \bar{\rho}$ e o custo de S é menor ou igual ao custo de $\bar{\rho}$, para qualquer permutação π.*

Demonstração. Sejam π_i, π_{i+1} e π_{i+2} os elementos afetados por $\bar{\rho}$, ou seja, $\bar{\rho} = \bar{\rho}(i, i+2)$. Essa operação transforma π na permutação $(\pi_1 \ \ldots \ \pi_{i-1} \ -\pi_{i+2} \ -\pi_{i+1} \ -\pi_i \ \pi_{i+3} \ \ldots \ \pi_n)$.

Seja $S = \langle \bar{\rho}(i, i{+}1), \bar{\rho}(i{+}1, i{+}2), \bar{\rho}(i, i{+}1), \bar{\rho}(i, i), \bar{\rho}(i{+}1, i{+}1), \bar{\rho}(i{+}2, i{+}2) \rangle$. A sequência S possui o mesmo efeito da reversão com sinais $\bar{\rho}(i, i+2)$, como mostrado a seguir.

$$
\begin{aligned}
\pi^0 = \pi \qquad\qquad\quad &= (\pi_1 \ \ldots \ \pi_{i-1} \ \ \pi_i \ \ \pi_{i+1} \ \ \pi_{i+2} \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^1 = \pi^0 \circ \bar{\rho}(i, i{+}1) \ &= (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{-\pi_{i+1} \ -\pi_i} \ \ \pi_{i+2} \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^2 = \pi^1 \circ \bar{\rho}(i{+}1, i{+}2) &= (\pi_1 \ \ldots \ \pi_{i-1} \ -\pi_{i+1} \ \underline{-\pi_{i+2} \ \ \pi_i} \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^3 = \pi^2 \circ \bar{\rho}(i, i{+}1) \ &= (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_{i+2} \ \ \pi_{i+1}} \ \ \pi_i \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^4 = \pi^3 \circ \bar{\rho}(i, i) \qquad &= (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{-\pi_{i+2}} \ \ \pi_{i+1} \ \ \pi_i \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^5 = \pi^4 \circ \bar{\rho}(i{+}1, i{+}1) &= (\pi_1 \ \ldots \ \pi_{i-1} \ -\pi_{i+2} \ \underline{-\pi_{i+1}} \ \ \pi_i \ \ \pi_{i+3} \ \ldots \ \pi_n) \\
\pi^6 = \pi^5 \circ \bar{\rho}(i{+}2, i{+}2) &= (\pi_1 \ \ldots \ \pi_{i-1} \ -\pi_{i+2} \ -\pi_{i+1} \ \underline{-\pi_i} \ \ \pi_{i+3} \ \ldots \ \pi_n)
\end{aligned}
$$

Notamos que o custo de S é igual a $3 \times 2^\alpha + 3 \times 1^\alpha$, que é menor ou igual a 3^α para $\alpha \geq 3$. Portanto, $\pi \circ \bar{\rho} = \pi \circ S$ e o custo de S é menor ou igual ao custo de $\bar{\rho}$. □

Lema 73. *Considerando $\alpha \geq 3$, existe uma sequência de ordenação ótima contendo apenas operações super curtas para SbLWs$\bar{\text{R}}$.*

Demonstração. Segue diretamente do Lema 72. □

Lema 74. *Considerando $\alpha \geq 3$, existe algoritmo polinomial para SbLWs$\bar{\text{R}}$.*

Demonstração. A partir do Lema 73, quando $\alpha \geq 3$, podemos reduzir o problema SbLWs$\bar{\text{R}}$ para o problema em que apenas reversões com sinais super curtas são permitidas. Seja S uma sequência de ordenação ótima para tal problema. Sabemos que a sequência S remove $\text{Inv}(\pi)$ inversões e diminui o tamanho de $E_\pi^{even^-} \cup E_\pi^{odd^+}$ em $|E_\pi^{even^-}| + |E_\pi^{odd^+}|$ unidades.

Como uma 2-reversão com sinais não altera o tamanho do conjunto $E_\pi^{even^-} \cup E^{odd^+}$ (Lema 47) e uma 1-reversão com sinais diminui o tamanho desse conjunto em no máximo uma unidade, temos que S possui no mínimo $|E_\pi^{even^-}| + |E_\pi^{odd^+}|$ reversões com sinais de tamanho 1.

Além disso, como uma 1-reversão com sinais não altera o número de inversões e uma 2-reversão com sinais remove no máximo uma inversão, temos que S possui no mínimo $\text{Inv}(\pi)$ reversões com sinais de tamanho 2. Sendo assim, o custo de S é maior ou igual a $|E_\pi^{even^-}| + |E_\pi^{odd^+}| + (2^\alpha \, \text{Inv}(\pi))$.

Usando uma ideia similar à prova do Lema 61, encontramos sequência de ordenação S' com custo $|E_\pi^{even^-}| + |E_\pi^{odd^+}| + (2^\alpha \operatorname{Inv}(\pi))$. A partir do limitante inferior apresentado para S, concluímos que S' é uma sequência de ordenação ótima. □

Capítulo 5

Considerações Finais

Esta dissertação apresentou um estudo dos problemas da Ordenação de Permutações por Operações Ponderadas pelo Número de Fragmentações e Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho. Consideramos modelos de rearranjo envolvendo reversões, transposições, e a combinação de reversões e transposições para permutações com e sem sinais. Para cada um desses problemas, apresentamos uma relação com a abordagem não ponderada, algoritmos de aproximação e resultados experimentais dos algoritmos desenvolvidos. As tabelas 5.1 e 5.2 apresentam um resumo dos melhores algoritmos de aproximação obtidos para cada variação dos problemas.

Considerando a ponderação pelo número de fragmentações, apresentamos resultados para o diâmetro, propriedades das permutações simples e uma 1.5-aproximação assintótica considerando permutações simples para dois modelos de rearranjo. Já considerando a ponderação pelo tamanho com a restrição de operações curtas, realizamos uma análise do fator de aproximação dos algoritmos desenvolvidos quando o custo de uma operação é igual a ℓ^α, onde ℓ é o tamanho da operação e $\alpha > 1$ é uma constante.

Parte dos resultados do Capítulo 3 constituem o artigo *Approximation Algorithms for Sorting Permutations by Fragmentation-Weighted Operations* [1], apresentado na *International Conference on Algorithms for Computational Biology* (AlCoB'2018). Já os resultados do Capítulo 4 constituem o artigo *Approximation Algorithms for Sorting Permutations by Length-Weighted Short Rearrangements*, aceito para apresentação no *Latin and American Algorithms, Graphs and Optimization Symposium* (LAGOS'2019). Além disso, também contribuímos no artigo *Sorting λ-Permutations by λ-Operations* [28], apresentado no *Brazilian Symposium on Bioinformatics* (BSB'2018). Nesse artigo, estudamos uma nova abordagem do problema de rearranjo de genomas, em que todos os elementos das permutações estão a uma distância menor do que λ da sua posição correta e apenas operações de tamanho menor ou igual a λ são permitidas.

Um trabalho futuro importante é o estudo da complexidade desses problemas, assim como o estudo da complexidade de outras variações dos problemas de rearranjo de genomas. Para a ponderação por fragmentações, julgamos importante o desenvolvimento de novos limitantes inferiores usando o grafo de ciclos. Como trabalho futuro para operações curtas ponderadas pelo tamanho, sugerimos o estudo do diâmetro desses problemas e melhores limitantes para os problemas que consideram permutações com sinais.

Tabela 5.1: Resumo dos melhores algoritmos de aproximação obtidos para os problemas da Ordenação de Permutações por Operações Ponderadas pelo Número de Fragmentações.

Modelo de Rearranjo	**Melhor Algoritmo de Aproximação**
Reversões sem Sinais	2-aproximação (teoremas 1 e 4)
Reversões com Sinais	2-aproximação (teoremas 2 e 4)
Transposições	2-aproximação (teoremas 3 e 5)
Reversões sem Sinais e Transposições	2-aproximação (teoremas 1 e 6)
Reversões com Sinais e Transposições	2-aproximação (teoremas 2 e 6)

Tabela 5.2: Resumo dos melhores algoritmos de aproximação obtidos para os problemas da Ordenação de Permutações por Operações Curtas Ponderadas pelo Tamanho.

Modelo de Rearranjo	**Melhor Algoritmo de Aproximação**
Reversões Curtas sem Sinais	2-aproximação (Teorema 11)
Reversões Curtas com Sinais	3-aproximação (Teorema 14)
Transposições Curtas	4/3-aproximação (Teorema 12)
Reversões Curtas sem Sinais e Transposições Curtas	2-aproximação (Teorema 11)
Reversões Curtas com Sinais e Transposições Curtas	7/3-aproximação (Teorema 15)

Referências Bibliográficas

[1] Alexsandro Oliveira Alexandrino, Carla Negri Lintzmayer, and Zanoni Dias. Approximation Algorithms for Sorting Permutations by Fragmentation-Weighted Operations. In *Algorithms for Computational Biology*, volume 10849, pages 53–64. Springer International Publishing, 2018.

[2] Martin Bader and Enno Ohlebusch. Sorting by Weighted Reversals, Transpositions, and Inverted Transpositions. *Journal of Computational Biology*, 14(5):615–636, 2007.

[3] Vineet Bafna and Pavel A. Pevzner. Sorting by Transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[4] Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y. Pinter, Steven S. Skiena, and Firas Swidan. Improved Bounds on Sorting by Length-Weighted Reversals. *Journal of Computer and System Sciences*, 74(5):744–774, 2008.

[5] Piotr Berman, Sridhar Hannenhalli, and Marek 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. Springer-Verlag Berlin Heidelberg New York, Berlin/Heidelberg, Germany, 2002.

[6] Mathieu Blanchette, Takashi Kunisawa, and David Sankoff. Parametric Genome Rearrangement. *Gene*, 172(1):GC11–GC17, 1996.

[7] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Sorting by Transpositions is Difficult. *SIAM Journal on Computing*, 26(3):1148–1180, 2012.

[8] Alberto Caprara. Sorting Permutations by Reversals and Eulerian Cycle Decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[9] Xin Chen. On Sorting Unsigned Permutations by Double-Cut-and-Joins. *Journal of Combinatorial Optimization*, 25(3):339–351, 2013.

[10] Bhadrachalam Chitturi. Tighter Upper Bound for Sorting Permutations with Prefix Transpositions. *Theoretical Computer Science*, 602:22–31, 2015.

[11] Bhadrachalam Chitturi, William Fahle, Z. Meng, Linda Morales, Charles O. Shields, Ivan H. Sudborough, and Walter Voit. An (18/11)n Upper Bound for Sorting by Prefix Reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009.

[12] Josef Cibulka. On Average and Highest Number of Flips in Pancake Sorting. *Theoretical Computer Science*, 412(8-10):822–834, 2011.

[13] Isaac Elias and Tzvika Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[14] Niklas Eriksen. Combinatorics of Genome Rearrangements and Phylogeny. Teknologie Licentiat Thesis, Kungliga Tekniska Högskolan, Stockholm, 2001.

[15] Niklas Eriksen. $(1+\epsilon)$-Approximation of Sorting by Reversals and Transpositions. *Theoretical Computer Science*, 289(1):517–529, 2002.

[16] Gustavo R. Galvão, Orlando Lee, and Zanoni Dias. Sorting Signed Permutations by Short Operations. *Algorithms for Molecular Biology*, 10(1):1–17, 2015.

[17] Qian-Ping Gu, Shietung Peng, and Ivan H. Sudborough. A 2-Approximation Algorithm for Genome Rearrangements by Reversals and Transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.

[18] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[19] Tzvika Hartman and Roded Sharan. A 1.5-Approximation Algorithm for Sorting by Transpositions and Transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005.

[20] Lenwood S. Heath and John Paul C. Vergara. Sorting by Short Swaps. *Journal of Computational Biology*, 10(5):775–789, 2003.

[21] Mark R. Jerrum. The Complexity of Finding Minimum-length Generator Sequences. *Theoretical Computer Science*, 36(2-3):265–289, 1985.

[22] Haitao Jiang, Haodi Feng, and Daming 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)*, volume 8889 of *Lecture Notes in Computer Science*, pages 491–503. Springer International Publishing, 2014.

[23] Haitao Jiang, Daming Zhu, and Binhai Zhu. A (1+e)-Approximation Algorithm for Sorting by Short Block-Moves. *Theoretical Computer Science*, 437:1–8, 2012.

[24] Jean-François Lefebvre, Nadia El-Mabrouk, Elisabeth R. M. Tillier, and David Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, 19(1):i190–i196, 2003.

[25] Guohui Lin and Tao Jiang. A Further Improved Approximation Algorithm for Breakpoint Graph Decomposition. *Journal of Combinatorial Optimization*, 8(2):183–194, 2004.

[26] Carla N. Lintzmayer, Guillaume Fertin, and Zanoni Dias. Approximation Algorithms for Sorting by Length-Weighted Prefix and Suffix Operations. *Theoretical Computer Science*, 593:26–41, 2015.

[27] Carla Negri Lintzmayer, Guillaume Fertin, and Zanoni Dias. Sorting permutations and binary strings by length-weighted rearrangements. *Theoretical Computer Science*, 715:35–59, 2018.

[28] Guilherme Henrique Santos Miranda, Alexsandro Oliveira Alexandrino, Carla Negri Lintzmayer, and Zanoni Dias. Sorting λ-permutations by λ-operations. In *Advances in Bioinformatics and Computational Biology*, pages 1–13. Springer International Publishing, 2018.

[29] Thach C. Nguyen, Hieu T. Ngo, and Nguyen B. Nguyen. Sorting by Restricted-Length-Weighted Reversals. *Genomics Proteomics & Bioinformatics*, 3(2):120–127, 2005.

[30] Andre Rodrigues Oliveira, Klairton Lima Brito, Zanoni Dias, and Ulisses Dias. Sorting by weighted reversals and transpositions. In *Advances in Bioinformatics and Computational Biology*, pages 38–49. Springer International Publishing, 2018.

[31] Ron Y. Pinter and Steven Skiena. Genomic Sorting with Length-Weighted Reversals. *Genome Informatics*, 13:2002, 2002.

[32] Atif Rahman, Swakkhar Shatabda, and Masud Hasan. An Approximation Algorithm for Sorting by Reversals and Transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.

[33] Firas Swidan, Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, and Ron Y. Pinter. Sorting by Length-Weighted Reversals: Dealing with Signs and Circularity. In *Combinatorial Pattern Matching (CPM'2004)*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin Heidelberg, Heidelberg, Germany, 2004.

[34] John P. C. Vergara. *Sorting by Bounded Permutations*. PhD thesis, Virginia Polytechnic Institute and State University, 1998.

[35] Maria E. M. T. Walter, Zanoni Dias, and João 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, Los Alamitos, CA, USA, 1998. IEEE Computer Society.