Universidade Estadual de Campinas
Instituto de Computação

Klairton de Lima Brito

Ordenação de Permutações com Sinais por Reversões e Transposições

CAMPINAS
2018

Klairton de Lima Brito

Ordenação de Permutações com Sinais por Reversões e Transposições

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

Este exemplar corresponde à versão final da Dissertação defendida por Klairton de Lima Brito e orientada pelo Prof. Dr. Zanoni Dias.

CAMPINAS
2018

Agência(s) de fomento e nº(s) de processo(s): CNPq, 138219/2016-8
ORCID: https://orcid.org/0000-0001-5287-2925

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

B777o

Brito, Klairton de Lima, 1991-
Ordenação de permutações com sinais por reversões e transposições / Klairton de Lima Brito. – Campinas, SP : [s.n.], 2018.

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

1. Biologia computacional. 2. Genomas. 3. Heurística (Computação). 4. Algoritmos de aproximação. I. Dias, Zanoni, 1975-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Sorting signed permutations by reversals and transpositions
Palavras-chave em inglês:
Computational biology
Genomes
Heuristic (Computer science)
Approximation algorithms
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Maria Emília Machado Telles Walter
Guilherme Pimentel Telles
Data de defesa: 12-04-2018
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Klairton de Lima Brito

Ordenação de Permutações com Sinais por Reversões e Transposições

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação - Universidade Estadual de Campinas
- Profa. Dra. Maria Emilia Machado Telles Walter
  Instituto de Ciências Exatas - Universidade de Brasília
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação - Universidade Estadual de Campinas

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 12 de abril de 2018

Dedicatória

Aos meus pais José Airton e Raimunda, com todo o meu amor e gratidão.

Agradecimentos

Aos membros da minha família, em especial meus pais José Airton e Raimunda, por todo o amor e apoio durante estes anos. Por sempre estarem do meu lado, incentivando-me e dando forças para alcançar meus objetivos.

À Camila, pelo carinho, amizade, companheirismo e principalmente pela paciência, qualidade que fiz você utilizar bastante durante todo esse tempo.

Ao meu orientador Zanoni Dias, por todo o apoio, incentivo, ideias, críticas, dedicação e principalmente pela confiança depositada em mim durante este trabalho. Mais que um orientador, Zanoni é um grande amigo que fez minha estadia aqui em Campinas tornar-se muito agradável, mesmo estando tão longe de casa. Não poderia ter tido um melhor orientador.

Ao Ulisses, Andre e Carla, pelas dicas, explicações e principalmente pela ajuda durante este período, que foram fundamentais para este trabalho.

Aos professores Críston e Lucas Ismaily, pelo incentivo para iniciar essa jornada. O apoio de vocês foi de fundamental importância.

Aos meus amigos, novos e velhos, pelo apoio sempre que precisei.

Ao CNPq pelo suporte financeiro durante este período.

A todos que direta ou indiretamente fizeram parte dessa conquista, o meu muito obrigado.

Resumo

Neste trabalho, apresentamos três heurísticas para melhorar os resultados de algoritmos para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições. Nossas heurísticas foram aplicadas tanto na versão clássica do problema quanto nas versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo. A nossa primeira heurística é chamada de *Sliding Window*, executa em tempo linear e fornece bons resultados. A nossa segunda heurística, *Look Ahead*, possui tempo de execução mais elevado dependendo do estimador de distância que é utilizado, mas seus resultados são mais expressivos. Nossa última heurística é chamada de *Iterative Sliding Window* e apresenta melhor custo-benefício entre o tempo de execução e a qualidade da solução em comparação com as duas primeiras heurísticas.

Mostramos que as heurísticas *Look Ahead* e *Sliding Window* podem ser combinadas visando fornecer resultados ainda melhores. Para verificar o comportamento de nossas heurísticas, implementamos diversos algoritmos da literatura e realizamos experimentos práticos onde observamos melhorias em relação aos algoritmos originais. Efetuamos ainda um comparativo de desempenho entre nossas próprias heurísticas.

Investigamos também o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas, adotando os pesos 2 e 3 para os eventos de reversão e transposição, respectivamente. Conseguimos apresentar limitantes inferiores para o problema e quatro algoritmos de aproximação. Dois desses algoritmos adotam uma estratégia gulosa e apresentam fatores de aproximação 3 e 2, enquanto os outros dois utilizam a estrutura de grafo de ciclos para ordenar a permutação e apresentam fatores de aproximação $\frac{5}{3}$ e $\frac{3}{2}$. Realizamos experimentos práticos com diferentes grupos de permutações com características distintas.

Estendemos nossa investigação para considerar outras relações de peso entre os eventos de transposição e reversão. Mostramos também uma análise que considera essas relações distintas de peso e apresentamos o melhor fator de aproximação obtido para cada uma dessas relações.

Abstract

In this work, we present three heuristics to improve existing solutions for the Sorting Signed Permutations by Reversals and Transpositions problem. To assess the heuristics, we implemented algorithms described in the literature to provide initial solutions. We investigated the classical version of the problem as well as versions restricted to prefix operations and restricted to prefix or suffix operations. The first heuristic is called *Sliding Window*, it runs in linear time and provides good results. The second heuristic is called *Look Ahead* and showed remarkable results, but its execution time increases depending on the distance estimator used. Our last heuristic is called *Iterative Sliding Window* and it showed better results when we consider solution quality and execution time.

We showed that the heuristics *Look Ahead* and *Sliding Window* can be combined to further improve the solutions. We performed the experiments and showed the improvement obtained by our heuristics compared with the solutions provided by the original algorithms. We also compared the three heuristics among themselves.

We also investigated the Sorting Signed Permutations by Weighted Reversals and Weighted Transpositions problem using weights 2 and 3 for reversals and transpositions, respectively. We showed lower bounds and developed four approximation algorithms for the problem. Two of these algorithms use a strategy based on breakpoints and have approximation factors 3 and 2. The other two algorithms resort to a structure called cycle graph to sort the permutations and achieved the approximation factors $\frac{5}{3}$ and $\frac{3}{2}$. We executed these algorithms in five groups of permutations with different characteristics.

To conclude this work, we made an analysis that considers different weights for reversal and transposition events and showed the best approximation factor obtained in each case.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Rearranjos de genomas [19] são eventos mutacionais que afetam grandes porções da sequência de DNA de um organismo. Tais eventos ocorrem em um cromossomo quando quebras em um ou mais locais geram pedaços que são remontados em uma ordem diferente. Devido ao *Princípio da Máxima Parcimônia*, é comum considerar o número mínimo de eventos que transforma um genoma em outro como uma aproximação para a distância evolutiva. O problema de Rearranjo de Genomas tem como objetivo buscar a menor sequência de eventos de rearranjo que transforma um genoma em outro, sendo que o tamanho desta sequência é denominado *distância de rearranjo* entre os indivíduos.

Assumindo que não existem genes duplicados, um genoma pode ser representado como uma permutação de inteiros. Quando os genes são dotados de alguma orientação, cada elemento da permutação é associado a um sinal positivo ou negativo, caso contrário, esse sinal é omitido. Uma permutação onde cada elemento está associado a um sinal é denominada de *permutação com sinais* e é denominada de *permutação sem sinais* ou simplesmente *permutação* caso contrário.

Reversões e Transposições são os eventos de rearranjo mais estudados na literatura [4,15]. Uma reversão inverte um segmento do DNA, alterando a posição e a orientação dos genes presentes no segmento. Uma transposição troca dois segmentos consecutivos do DNA de posição, mas sem alterar a ordem e a orientação dos genes presentes nos mesmos.

Quando genomas são mapeados em permutações, tende-se a tratar os problemas do ponto de vista de ordenação, ou seja, o objetivo passa a ser transformar uma permutação α na *permutação identidade* $\iota_n = (+1 \ \ldots \ +n)$.

Em 1999, Hannenhalli e Pevzner [24] provaram que o problema de Ordenação de Permutações com Sinais por Reversões pode ser resolvido em tempo polinomial. Por outro lado, ainda no mesmo ano, Caprara [8] provou que a versão sem sinais do problema pertence à classe de problemas NP-Difíceis. Em 2012, Bulteau e coautores [6] provaram que o problema de Ordenação de Permutações por Transposições também pertence à classe de problemas NP-Difíceis.

Diferente dos problemas citados anteriormente, onde são considerados apenas eventos de reversão ou eventos de transposição separadamente, a complexidade do problema de Ordenação de Permutações com Sinais por Reversões e Transposições ainda permanece desconhecida. O mesmo é verdade para a versão sem sinais do problema. O melhor algoritmo de aproximação para a versão com sinais apresenta um fator de aproximação

2 [32]. Para a versão sem sinais do problema, o melhor algoritmo apresenta um fator de aproximação $2k$, onde k representa o fator de aproximação do algoritmo utilizado para decomposição de ciclos [9].

Em 1995, o conhecido problema de Ordenação de Panquecas foi reinterpretado como um problema de rearranjo de genomas por Pevzner e Waterman [28], onde recebeu o nome de problema de Ordenação de Permutações sem Sinais por Reversões de Prefixo. Nesse novo problema, foi adicionada uma restrição extra, em que qualquer evento de reversão deve sempre afetar o primeiro elemento da permutação. Em 2015, Bulteau e coautores [7] provaram que esse problema pertence à classe de problemas NP-Difíceis. O melhor algoritmo de aproximação possui fator de aproximação 2 [20].

Gates e Papadimitriou [22] apresentaram o problema de Ordenação de Panquecas Queimadas, que posteriormente foi reinterpretado como problema de Ordenação de Permutações com Sinais por Reversões de Prefixo. Esse problema permanece com complexidade desconhecida, e o melhor algoritmo possui fator de aproximação 2 [11].

Em 2002, Dias e Meidanis [14] apresentaram o problema de Ordenação de Permutações por Transposições de Prefixo juntamente com um algoritmo 2-aproximado, o melhor algoritmo conhecido para o problema. A complexidade do problema permanece desconhecida.

O problema de Ordenação de Permutações com Sinais por Reversões e Transposições de Prefixo possui complexidade desconhecida, sendo que o melhor algoritmo apresenta um fator de aproximação $2 + 4/b_p(\pi)$ [27], onde $b_p(\pi)$ é o número de breakpoints de prefixo da permutação π (vide Seção 1.7).

Outra variação do problema considera eventos de prefixo ou sufixo, ou seja, os eventos de rearranjo afetam o primeiro ou o último elemento da permutação.

Considerando os eventos de reversão e transposição de prefixo ou sufixo, quase todas as variações possuem complexidade desconhecida, com exceção do problema de Ordenação de Permutações por Reversões de Prefixo ou Sufixo, que pertence à classe de problemas NP-Difíceis [18].

No restante deste capítulo apresentaremos notações e conceitos que serão utilizados e proporcionam uma melhor compreensão do trabalho.

1.1 Modelo de Rearranjo

Um *modelo de rearranjo* é o conjunto de eventos considerados em um problema de ordenação por rearranjo de genomas. Os primeiros trabalhos em rearranjo de genomas estudaram modelos que permitiam apenas um tipo de rearranjo [4, 15, 24]. Outros estudos consideram modelos com um ou mais tipos de rearranjo [25, 29, 32].

Em problemas de rearranjo de genomas, um genoma é uma n-tupla, em que cada elemento representa um gene ou blocos de genes do genoma. Assumindo que não existem genes duplicados, a n-tupla uma permutação $\pi = (\pi_1\ \pi_2\ \pi_3\ \ldots\ \pi_n)$ de números inteiros, tal que $\pi_i \in \{-n, -(n-1), \ldots, -2, -1, +1, +2, \ldots, +(n-1), +n\}$ e $|\pi_i| \neq |\pi_j| \leftrightarrow i \neq j$. O sinal positivo ou negativo de um elemento informa a orientação do gene. Existem abordagens que não trabalham com a orientação dos genes e, nesse caso, o sinal é omitido.

O problema de distância de ordenação procura a menor sequência de eventos $\delta_1, \ldots, \delta_t$

pertencentes à um modelo de rearranjo M de maneira que $\pi \circ \delta_1 \circ \cdots \circ \delta_t = \sigma$. A *distância* entre as permutações π e σ no modelo M é o tamanho da menor sequência de eventos de rearranjo que transforma π em σ, e é denotada por $d_M(\pi, \sigma)$.

1.2 Composição e Permutação Inversa

A composição entre as permutações π e σ, definida como $\pi \circ \sigma$, é uma operação que resulta em uma nova permutação α. O Exemplo 1.1 mostra a composição entre as permutações π e σ.

$$\begin{aligned} \pi &= (\pi_1\ \pi_2\ \ldots\ \pi_n) \\ \sigma &= (\sigma_1\ \sigma_2\ \ldots\ \sigma_n) \\ \pi \circ \sigma &= (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \ldots\ \pi_{\sigma_n}) \ = \alpha \end{aligned} \tag{1.1}$$

A composição entre permutações com sinais considera o sinal de cada elemento da permutação σ. Se $\sigma_i < 0$, então $\alpha_i = -\pi_{|\sigma_i|}$, caso contrário $\alpha_i = \pi_{\sigma_i}$. O Exemplo 1.2 mostra a composição entre duas permutações com sinais π e σ.

$$\begin{aligned} \pi &= (+1\ +2\ -4\ -3\ +6\ -5) \\ \sigma &= (-3\ +1\ -6\ -5\ +4\ +2) \\ \pi \circ \sigma &= (+4\ +1\ +5\ -6\ -3\ +2) \end{aligned} \tag{1.2}$$

A permutação inversa computada a partir de uma permutação π é representada como π^{-1}. Essa permutação indica a posição e a orientação de cada elemento i em π. Além disso, temos que $\pi \circ \pi^{-1} = \pi^{-1} \circ \pi = \iota$. No Exemplo 1.3 temos uma permutação sem sinais π e a sua inversa π^{-1}.

$$\begin{aligned} \pi &= (2\ 1\ 4\ 5\ 3\ 6) \\ \pi^{-1} &= (2\ 1\ 5\ 3\ 4\ 6) \end{aligned} \tag{1.3}$$

Pelo Exemplo 1.3, podemos ver que o elemento 1 está na posição 2, o elemento 2 está na posição 1, o elemento 3 está na posição 5, o elemento 4 está na posição 3, o elemento 5 está na posição 4 e o elemento 6 está na posição 6. No Exemplo 1.4 temos uma permutação com sinais π e a sua inversa π^{-1}.

$$\begin{aligned} \pi &= (+2\ -1\ +4\ -5\ +3\ +6) \\ \pi^{-1} &= (-2\ +1\ +5\ +3\ -4\ +6) \end{aligned} \tag{1.4}$$

Pelo Exemplo 1.4, podemos ver que o elemento 1 está na posição 2 com sinal negativo, o elemento 2 está na posição 1 com sinal positivo, o elemento 3 está na posição 5 com sinal positivo, o elemento 4 está na posição 3 com sinal positivo, o elemento 5 está na posição 4 com sinal negativo e o elemento 6 está na posição 6 com sinal positivo.

1.3 Reversões

Uma reversão $\rho(i,j)$, onde $1 \leq i \leq j \leq n$, é um evento que inverte a posição e a orientação dos genes em um bloco do genoma. Quando a orientação dos genes é conhecida, caracteriza-se um problema de Ordenação de Permutações com Sinais por Reversões. Caso contrário, tem-se um problema de Ordenação de Permutações sem Sinais por Reversões. O Exemplo 1.5 mostra uma operação de reversão $\rho(i,j)$ sendo aplicada.

$$\begin{aligned} \pi &= (+\pi_1 \ \ldots \ +\pi_{i-1} \ \underline{+\pi_i \ \ldots \ +\pi_j} \ +\pi_{j+1} \ \ldots \ +\pi_n) \\ \rho(i,j) &= (+1 \ \ldots \ +(i-1) \ \underline{-j \ \ldots \ -i} \ +(j+1) \ \ldots \ +n) \\ \pi \circ \rho(i,j) &= (+\pi_1 \ \ldots \ +\pi_{i-1} \ \underline{-\pi_j \ \ldots \ -\pi_i} \ +\pi_{j+1} \ \ldots \ +\pi_n) \end{aligned} \tag{1.5}$$

O Exemplo 1.6 mostra a menor sequência de reversões que ordena a permutação sem sinais $\pi = (6\ 4\ 2\ 5\ 3\ 1)$.

$$\begin{aligned} \pi^0 &= \pi & &= (6\ 4\ \underline{2\ 5\ 3}\ 1) \\ \pi^1 &= \pi^0 \circ \rho(3,5) & &= (6\ 4\ \underline{3\ 5}\ 2\ 1) \\ \pi^2 &= \pi^1 \circ \rho(3,4) & &= (6\ \underline{4\ 5}\ 3\ 2\ 1) \\ \pi^3 &= \pi^2 \circ \rho(2,3) & &= (\underline{6\ 5\ 4\ 3\ 2\ 1}) \\ \pi^4 &= \pi^3 \circ \rho(1,6) & &= (1\ 2\ 3\ 4\ 5\ 6) \end{aligned} \tag{1.6}$$

O Exemplo 1.7 mostra a menor sequência de reversões que ordena a permutação com sinais $\pi = (-6\ +4\ -2\ +5\ -3\ +1)$.

$$\begin{aligned} \pi^0 &= \pi & &= (-6\ \underline{+4\ -2\ +5}\ -3\ +1) \\ \pi^1 &= \pi^0 \circ \rho(2,4) & &= (-6\ -5\ \underline{+2\ -4\ -3}\ +1) \\ \pi^2 &= \pi^1 \circ \rho(3,5) & &= (-6\ -5\ \underline{+3\ +4}\ -2\ +1) \\ \pi^3 &= \pi^2 \circ \rho(3,4) & &= (\underline{-6\ -5\ -4\ -3\ -2\ +1}) \\ \pi^4 &= \pi^3 \circ \rho(1,6) & &= (\underline{-1}\ +2\ +3\ +4\ +5\ +6) \\ \pi^5 &= \pi^4 \circ \rho(1,1) & &= (+1\ +2\ +3\ +4\ +5\ +6) \end{aligned} \tag{1.7}$$

Uma reversão de prefixo $\rho(1,j)$, onde $1 \leq j \leq n$, é uma reversão que afeta um bloco no início do genoma. Uma reversão de sufixo $\rho(i,n)$, onde $1 \leq i \leq n$, é uma reversão que afeta um bloco no fim do genoma.

1.4 Transposições

Uma transposição $\tau(i,j,k)$, $1 \leq i < j < k \leq n+1$, é um evento que troca de posição dois blocos adjacentes de um mesmo genoma $\pi[i \ldots j-1]$ e $\pi[j \ldots k-1]$, mas sem afetar a orientação e a posição dos genes dentro dos blocos. Por tal motivo, em um modelo de rearranjo cujo conjunto de eventos é formado unicamente pelo evento de transposição, elimina-se a necessidade de associar sinais aos elementos da permutação. O Exemplo 1.8

mostra uma transposição $\tau(i,j,k)$ aplicada na permutação π.

$$\begin{aligned}
\pi &= (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_i \ \ldots \ \pi_{j-1}} \ \underline{\pi_j \ \ldots \ \pi_{k-1}} \ \pi_k \ \ldots \ \pi_n) \\
\tau(i,j,k) &= (1 \ \ldots \ (i-1) \ \underline{j \ \ldots \ (k-1)} \ \underline{i \ \ldots \ (j-1)} \ k \ \ldots \ n) \\
\pi \circ \tau(i,j,k) &= (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \ldots \ \pi_{k-1}} \ \underline{\pi_i \ \ldots \ \pi_{j-1}} \ \pi_k \ \ldots \ \pi_n)
\end{aligned} \tag{1.8}$$

O Exemplo 1.9 mostra a menor sequência de transposições que ordena a permutação sem sinais $\pi = (6\ 4\ 2\ 5\ 3\ 1)$.

$$\begin{aligned}
\pi^0 &= \pi & &= (\underline{6}\ \underline{4\ 2\ 5\ 3\ 1}) \\
\pi^1 &= \pi^0 \circ \tau(1,2,7) & &= (\underline{4\ 2\ 5}\ \underline{3\ 1}\ 6) \\
\pi^2 &= \pi^1 \circ \tau(1,4,6) & &= (\underline{3\ 1\ 4}\ \underline{2}\ 5\ 6) \\
\pi^3 &= \pi^2 \circ \tau(1,4,5) & &= (\underline{2\ 3}\ \underline{1}\ 4\ 5\ 6) \\
\pi^4 &= \pi^3 \circ \tau(1,3,4) & &= (1\ 2\ 3\ 4\ 5\ 6)
\end{aligned} \tag{1.9}$$

Uma transposição de prefixo $\tau(1,j,k)$, $1 < j < k \leq n+1$, é uma transposição que afeta um bloco no início do genoma. Uma transposição de sufixo $\tau(i,j,n+1)$, onde $1 \leq i < j < n+1$, é uma transposição que afeta um bloco no fim do genoma.

1.5 Reversões e Transposições

O Exemplo 1.10 mostra a menor sequência de eventos que ordena a permutação sem sinais π em um modelo de rearranjo em que reversões e transposições são permitidas.

$$\begin{aligned}
\pi^0 &= \pi & &= (6\ 4\ \underline{2}\ \underline{5\ 3}\ 1) \\
\pi^1 &= \pi^0 \circ \tau(3,4,6) & &= (6\ \underline{4\ 5}\ 3\ 2\ 1) \\
\pi^2 &= \pi^1 \circ \rho(2,3) & &= (\underline{6\ 5\ 4\ 3\ 2\ 1}) \\
\pi^3 &= \pi^2 \circ \rho(1,6) & &= (1\ 2\ 3\ 4\ 5\ 6)
\end{aligned} \tag{1.10}$$

O Exemplo 1.11 mostra a menor sequência de eventos que ordena a permutação com sinais π considerando o mesmo modelo de rearranjo.

$$\begin{aligned}
\pi^0 &= \pi & &= (-6\ \underline{+4\ -2\ +5}\ -3\ +1) \\
\pi^1 &= \pi^0 \circ \rho(2,4) & &= (-6\ -5\ \underline{+2}\ \underline{-4\ -3\ +1}) \\
\pi^2 &= \pi^1 \circ \tau(3,4,7) & &= (\underline{-6\ -5\ -4\ -3\ +1\ +2}) \\
\pi^3 &= \pi^2 \circ \rho(1,6) & &= (\underline{-2\ -1}\ +3\ +4\ +5\ +6) \\
\pi^4 &= \pi^3 \circ \rho(1,2) & &= (+1\ +2\ +3\ +4\ +5\ +6)
\end{aligned} \tag{1.11}$$

1.6 Distância ×Ordenação

O problema de Distância de Rearranjo tem como objetivo transformar uma permutação π em uma permutação σ utilizando os eventos definidos por um modelo de rearranjo. O processo de transformar a permutação π na permutação σ é equivalente ao processo

de ordenação de uma permutação α, se tomarmos $\alpha = \sigma^{-1} \circ \pi$. Note que $d_M(\pi, \sigma) = d_M(\sigma^{-1} \circ \pi, \sigma^{-1} \circ \sigma) = d_M(\alpha, \iota_n) = d_M(\alpha)$.

O exemplo a seguir mostra que se conseguirmos ordenar uma permutação α, também seremos capazes de transformar a permutação π na permutação σ. Sejam π e σ duas permutações tais que $\pi = (+6\ +5\ +1\ +2\ +4\ +3)$ e $\sigma = (+2\ -1\ +4\ -5\ +3\ +6)$. Note que a permutação inversa de σ é $\sigma^{-1} = (-2\ +1\ +5\ +3\ -4\ +6)$. Com isso, podemos computar a permutação $\alpha = \sigma^{-1} \circ \pi = (+6\ -4\ -2\ +1\ +3\ +5)$. A sequência de ordenação apresentada no Exemplo 1.12 é uma sequência ótima que ordena a permutação α.

$$\begin{array}{lll} \alpha^0 & = \alpha & = (+6\ \underline{-4\ -2\ +1}\ +3\ +5) \\ \alpha^1 & = \alpha^0 \circ \rho(2,4) & = (+6\ -1\ +2\ \underline{+4}\ \underline{+3}\ +5) \\ \alpha^2 & = \alpha^1 \circ \tau(4,5,6) & = (\underline{+6}\ \underline{-1\ +2\ +3\ +4\ +5}) \\ \alpha^3 & = \alpha^2 \circ \tau(1,2,7) & = (\underline{-1}\ +2\ +3\ +4\ +5\ +6) \\ \alpha^4 & = \alpha^3 \circ \rho(1,1) & = (+1\ +2\ +3\ +4\ +5\ +6) \end{array} \tag{1.12}$$

No Exemplo 1.13, aplicamos a mesma sequência de eventos que ordena a permutação α na permutação π. Como resultado, obtemos a permutação σ.

$$\begin{array}{lll} \pi^0 & = \pi & = (+6\ \underline{+5\ +1\ +2}\ +4\ +3) \\ \pi^1 & = \pi^0 \circ \rho(2,4) & = (+6\ -2\ -1\ \underline{-5}\ \underline{+4}\ +3) \\ \pi^2 & = \pi^1 \circ \tau(4,5,6) & = (\underline{+6}\ \underline{-2\ -1\ +4\ -5\ +3}) \\ \pi^3 & = \pi^2 \circ \tau(1,2,7) & = (\underline{-2}\ -1\ +4\ -5\ +3\ +6) \\ \pi^4 & = \pi^3 \circ \rho(1,1) & = (+2\ -1\ +4\ -5\ +3\ +6) \quad = \sigma \end{array} \tag{1.13}$$

1.7 Breakpoints

A *permutação estendida* é obtida a partir de π inserindo dois novos elementos: $\pi_0 = +0$ e $\pi_{n+1} = +(n+1)$. Em uma permutação com sinais π e considerando que não há restrições nas operações de reversão e transposição, um *breakpoint* ocorre em um par de elementos π_i e π_{i+1} de π, com $0 \leq i \leq n$, tal que $\pi_{i+1} - \pi_i \neq 1$. Por exemplo, $\pi = (+0 \cdot -2\ -1 \cdot +4\ +5 \cdot -3 \cdot +6)$, onde " $\cdot$ " representa cada *breakpoint* da permutação estendida. O número de *breakpoints* em uma permutação π é denotado por $b(\pi)$. No exemplo acima, temos que $b(\pi) = 4$.

Em uma permutação com sinais π e considerando que apenas operações de prefixo são permitidas, um *breakpoint* ocorre em um par de elementos π_i e π_{i+1} de π, com $0 < i \leq n$, tal que $\pi_{i+1} - \pi_i \neq 1$. Por exemplo, $\pi = (+0\ -2\ -1 \cdot +4\ +5 \cdot -3 \cdot +6)$, onde "$\cdot$" representa cada *breakpoint* da permutação estendida. O número de *breakpoints* de prefixo em uma permutação π é denotado por $b_p(\pi)$. No exemplo acima, temos que $b_p(\pi) = 3$.

Em uma permutação com sinais π e considerando que apenas operações de prefixo ou sufixo são permitidas, um *breakpoint* ocorre em um par de elementos π_i e π_{i+1} de π, com $0 < i < n$, tal que $\pi_{i+1} - \pi_i \neq 1$. Por exemplo, $\pi = (+0\ -2\ -1 \cdot +4\ +5 \cdot -3\ +6)$, onde " $\cdot$ " representa cada *breakpoint* da permutação estendida. O número de *breakpoints* de prefixo ou sufixo em uma permutação π é denotado por $b_{ps}(\pi)$. No exemplo acima, temos que $b_{ps}(\pi) = 2$.

Para o caso restrito ao uso de operações de prefixo ou sufixo, existem apenas duas permutações que não possuem *breakpoints*, sendo elas a permutação identidade ι_n e a *permutação reversa* definida como $\eta_n = (-n \ \ldots \ -1)$. Para os demais casos, a permutação identidade é a única permutação que não possui *breakpoints*.

1.8 Strips

Os *breakpoints* dividem uma permutação em *strips*, que são intervalos maximais sem *breakpoints*. O processo de obter as *strips* de uma permutação não leva em consideração os elementos π_0 e π_{n+1}, inseridos na permutação estendida. Em permutações com sinais, uma *strip* é dita *positiva* se a orientação do seus elementos é positiva e é dita *negativa* caso contrário. No exemplo da permutação estendida com sinais $\pi = (+0 \cdot -2 \ -1 \cdot +4 \ +5 \cdot -3 \cdot +6)$, temos três *strips*: $(-2 \ -1)$, $(+4 \ +5)$ e (-3). No exemplo, $(-2 \ -1)$ e (-3) são strips negativas e $(+4 \ +5)$ é uma strip positiva.

1.9 Permutação Reduzida

Christie [10] mostrou que uma permutação π pode ser transformada em uma permutação reduzida $\pi_{reduced}$. Para qualquer modelo de rearranjo M, $d_M(\pi) \leq d_M(\pi_{reduced})$. A transformação é realizada utilizando as *strips* de uma permutação:

- Remover a primeira *strip*, se ela iniciar com $+1$;
- Remover a última *strip*, se ela terminar com $+n$;
- Substituir cada *strip* pelo menor elemento contido nela;
- Renomear os elementos de modo a obter uma permutação válida.

Por exemplo, a permutação $\pi = (+1 \ +2 \ \cdot \ -9 \ -8 \ \cdot \ +5 \ +6 \ +7 \ \cdot \ +3 \ +4)$ apresenta as seguintes *strips*: $(+1 \ +2)$, $(-9 \ -8)$, $(+5 \ +6 \ +7)$ e $(+3 \ +4)$. Removemos a primeira *strip*, pois ela inicia com $+1$, resultando nas *strips* $(-9 \ -8)$, $(+5 \ +6 \ +7)$ e $(+3 \ +4)$. Em seguida, substituímos cada *strip* pelo menor elemento contido nela, resultando em (-9) $(+5)$ e $(+3)$. Por fim, renomeamos os elementos para obter uma permutação válida e obtemos a permutação reduzida $\pi_{reduced} = (-3 \ +2 \ +1)$.

Para o caso restrito ao uso de operações de prefixo, a primeira *strip* nunca é removida, mesmo que ela inicie com $+1$. De maneira similar, no caso restrito ao uso de operações de prefixo ou sufixo, a primeira e a última *strip* nunca são removidas, mesmo que a primeira *strip* inicie com $+1$ ou a última *strip* termine com $+n$.

1.10 Grafo de Ciclos

A estrutura de *grafo de ciclos* foi apresentada por Bafna e Pevzner [3] em 1998 e é bastante utilizada nos problemas da área de rearranjo de genomas [15, 24]. O *grafo de ciclos* $G(\pi)$ é construído a partir de uma permutação com sinais π. O grafo é formado por um conjunto de vértices $V(\pi)$, um conjunto de arestas pretas $E_p(\pi)$ e um conjunto de arestas cinzas $E_c(\pi)$ de forma que:

- $V(\pi) = \{+\pi_0, -\pi_1, +\pi_1, \ldots, -\pi_n, +\pi_n, -\pi_{n+1}\}$
- $E_p(\pi) = \bigcup_{i=1}^{n+1} \{(-\pi_i, +\pi_{i-1})\}$
- $E_c(\pi) = \bigcup_{i=1}^{n+1} \{(+(i-1), -i)\}$

Algumas regras devem ser respeitadas durante o desenho do grafo de ciclos. Os vértices devem ser desenhados horizontalmente, da esquerda para direita. Inicialmente, adicionamos o vértice $+\pi_0 = +0$. Em seguida, para cada elemento $\pi_i \in \pi$, onde $1 \leq i \leq n$, inserimos os vértices $-\pi_i$ e π_i. Por fim, inserimos o vértice $-\pi_{n+1} = -(n+1)$.

As arestas pretas devem ser desenhadas horizontalmente e são rotuladas de 1 até $n+1$ de maneira que a aresta $(-\pi_i, \pi_{i-1})$ recebe o rótulo i. As arestas cinzas devem ser desenhadas formando arcos acima dos vértices.

A Figura 1.1(a) mostra o grafo de ciclos construído a partir da permutação identidade $\iota_5 = (+1\ +2\ +3\ +4\ +5)$. As arestas desenhadas horizontalmente e as que formam um arco representam as arestas pretas e cinzas, respectivamente.

Note que em cada vértice incide apenas uma aresta preta e uma aresta cinza. Dessa forma, existe uma única decomposição das arestas de $G(\pi)$ em ciclos com arestas de cores alternantes. Perceba também que a permutação identidade ι_n é a única permutação que apresenta $n+1$ ciclos quando decompomos as arestas em ciclos com arestas de cores alternantes.

No decorrer desse trabalho iremos referenciar ciclos com arestas de cores alternantes apenas como ciclos. Para representar um ciclo $c \in G(\pi)$, utilizamos uma sequência de arestas pretas seguindo a ordem com que elas são percorridas, sendo que a primeira aresta preta é aquela que encontra-se mais à direita no ciclo e é percorrida da direita para esquerda. Se uma aresta preta pertencente a um ciclo c é percorrida da esquerda para

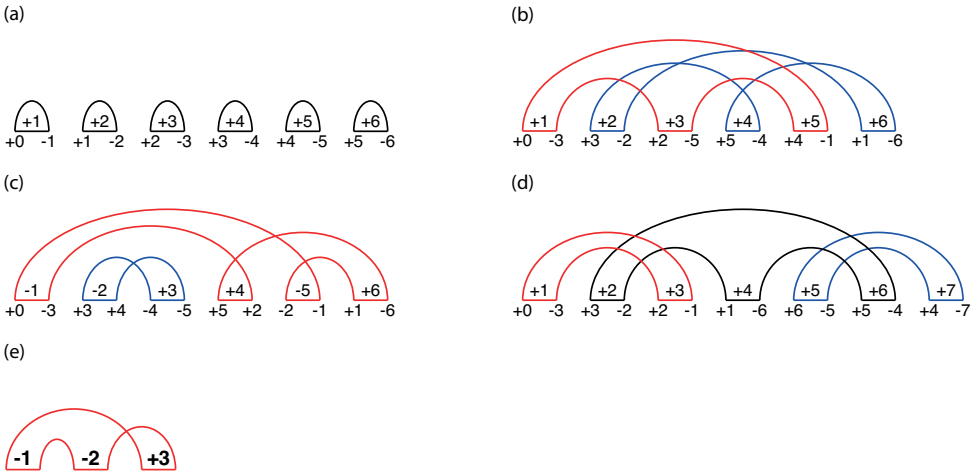

Figura 1.1: Exemplos de grafo de ciclos. (a) grafo de ciclos construído a partir da permutação identidade $\iota_5 = (+1\ +2\ +3\ +4\ +5)$. (b) grafo de ciclos construído a partir da permutação $\pi = (+3\ +2\ +5\ +4\ +1)$. (c) grafo de ciclos construído a partir da permutação $\pi = (+3\ -4\ +5\ -2\ +1)$, (d) grafo de ciclos construído a partir da permutação $\pi = (+3\ +2\ +1\ +6\ +5\ +4)$. (e) grafo de ciclos com um open gate.

direita, seu rótulo é definido como $-i$ e é chamada de *aresta negativa*. Caso contrário, seu rótulo é definido como $+i$ e é chamada de *aresta positiva*.

O tamanho de um ciclo $c \in G(\pi)$ é dado pelo número de arestas pretas no ciclo. Um ciclo de tamanho um é chamado de *trivial*. Na Figura 1.1(a) temos um grafo de ciclos composto apenas por ciclos triviais. Um ciclo é dito *curto* se seu tamanho é menor ou igual a três, e *longo*, caso contrário. Na Figura 1.1(c) temos o ciclo curto $c_1 = (+3, -2)$ e o ciclo longo $c_2 = (+6, -5, -1, +4)$. Um ciclo é ímpar se seu tamanho é ímpar, e par, caso contrário. Na Figura 1.1(d) temos um ciclo ímpar $c_1 = (+6, +4, +2)$ e dois ciclos pares $c_2 = (+3, +1)$ e $c_3 = (+7, +5)$. Definimos o número de ciclos em $G(\pi)$ como $c(\pi)$ e o número de ciclos ímpares em $G(\pi)$ como $c_{odd}(\pi)$.

Duas arestas pretas pertencentes a um ciclo c são chamadas de *convergentes* se elas apresentam o mesmo sinal, caso contrário são chamadas de *divergentes*. Um ciclo c é chamado de *divergente* se nele existe um par de arestas *divergentes* e é chamado de *convergente* caso contrário. Na Figura 1.1(c) temos dois ciclos divergentes $c_1 = (+3, -2)$ e $c_2 = (+6, -5, -1, +4)$, enquanto na Figura 1.1(d) temos três ciclos convergentes $c_1 = (+3, +1)$, $c_2 = (+6, +4, +2)$ e $c_3 = (+7, +5)$.

Além disso, um ciclo convergente $c = (i_1, \ldots, i_k)$, onde $k > 1$, é dito *não orientado* se $i_1 \ldots i_k$ estão em ordem decrescente e é dito *orientado* caso contrário. Na Figura 1.1(b) temos o ciclo não orientado $c_1 = (+5, +3, +1)$ e o ciclo orientado $c_2 = (+6, +2, +4)$. Dizemos que dois ciclos $c_1 = (i_1, ..., i_k)$ e $c_2 = (j_1, ..., j_k)$, com $k > 1$, são *entrelaçados* se $|i_1| > |j_1| > ... > |i_k| > |j_k|$ ou $|j_1| > |i_1| > ... > |j_k| > |i_k|$. Na Figura 1.1(b) os ciclos $c_1 = (+5, +3, +1)$ e $c_2 = (+6, +2, +4)$ estão entrelaçados.

Sejam g_1 e g_2 duas arestas cinzas, sendo que g_1 está ligada as arestas pretas x_1 e y_1, g_2 está ligada as arestas pretas x_2 e y_2, tal que $|x_1| < |y_1|$ e $|x_2| < |y_2|$. Dizemos que g_1 *cruza* g_2 se $|x_1| < |x_2| < |y_1| < |y_2|$ ou $|x_1| < |x_2| = |y_1| < |y_2|$. Dizemos que dois ciclos c_1 e c_2 *cruzam-se* se a aresta cinza $g_1 \in c_1$ cruza a aresta cinza $g_2 \in c_2$. Na Figura 1.1(d) o ciclo $c_1 = (+6, +4, +2)$ está cruzado com os ciclos $c_2 = (+3, +1)$ e $c_3 = (+7, +5)$.

A partir de qualquer permutação π é possível construir um grafo de ciclos $G(\pi)$, mas nem toda configuração de ciclos pode ser construída a partir de uma permutação. Sejam x e y duas arestas pretas em um ciclo $c = (\ldots, x, y, \ldots)$ ou $c = (\ldots, -x, -y, \ldots)$ e g a aresta cinza que conecta as arestas pretas x e y. Dizemos que g é um *open gate* se não existir outra aresta cinza que cruze g no grafo de ciclos. Na Figura 1.1(e) temos um grafo de ciclos com um open gate e que não gera uma permutação, ou seja, um grafo de ciclos que contém open gates não pode ser construído a partir de uma permutação.

Se π é uma permutação sem sinais, podemos atribuir sinais de maneira arbitrária a cada um de seus elementos para obter uma permutação com sinais π'. Dessa forma, podemos construir $G(\pi')$, como explicado previamente, e então atribuímos esse grafo de ciclos a π. Pelo fato de existirem várias atribuições de sinais aos elementos de π, temos que a permutação π pode ter diversas representações de grafo de ciclos. Como o objetivo é transformar a permutação π na permutação identidade, então, dentre todos os grafos ciclos para π, o melhor é aquele que apresenta o maior número de ciclos. Entretanto, encontrar esse grafo de ciclos é um problema NP-Difícil [8] e o melhor algoritmo para esse problema foi apresentado por Chen [9] com um fator de aproximação de $1.4167 + \epsilon$ para $\epsilon > 0$.

1.11 Permutação Simples

Uma permutação é *simples* caso todos os ciclos de $G(\pi)$ sejam curtos. Caso π não seja uma permutação simples, é possível obter uma permutação simples $\hat{\pi}$ quebrando todos os ciclos longos de $G(\pi)$ em ciclos curtos. O processo para quebrar ciclos longos em ciclos curtos é especificado a seguir:

- Sejam b_1, b_2 e b_3 arestas pretas distintas de um ciclo c, sendo que a aresta b_1 está conectada a aresta b_2 por meio de uma aresta cinza e a aresta b_2 está conectada a aresta b_3 por meio de uma aresta cinza.
- Seja g uma aresta cinza que está conectada a aresta b_1, mas não está conectada a aresta b_2.
- Assumindo que $b_3 = (V_b, W_b)$ e $g = (v_g, W_g)$, como mostra a Figura 1.2 a), removemos as arestas b_3 e g.
- Depois disso, inserimos dois novos vértices W e V entre os vértices W_b e V_b e criamos duas novas arestas pretas (W_b, W) e (V, V_B).
- Por fim, adicionamos duas arestas cinzas (W_g, W) e (V, V_g).

Como resultado, temos que o ciclo c é removido e dois novos ciclos c_1 e c_2, são adicionados, sendo que c_1 é formado por 3 arestas pretas e c_2 é formado por $k-2$ arestas pretas, onde k é o número de arestas pretas do ciclo c. Cada vez que repetimos esse processo, aumentamos o número de elementos, arestas pretas e ciclos ímpares da permutação em uma unidade.

A sequência de eventos que ordena $\hat{\pi}$ pode ser adaptada para ordenar π com no máximo a mesma quantidade de operações ignorando os elementos que foram adicionados em $\hat{\pi}$. Por exemplo, se uma reversão que atua na permutação $\hat{\pi}$ afeta os elementos $[a, b, c, d]$, sendo que os elementos b e d foram adicionados em $\hat{\pi}$ e não fazem parte de π, então essa

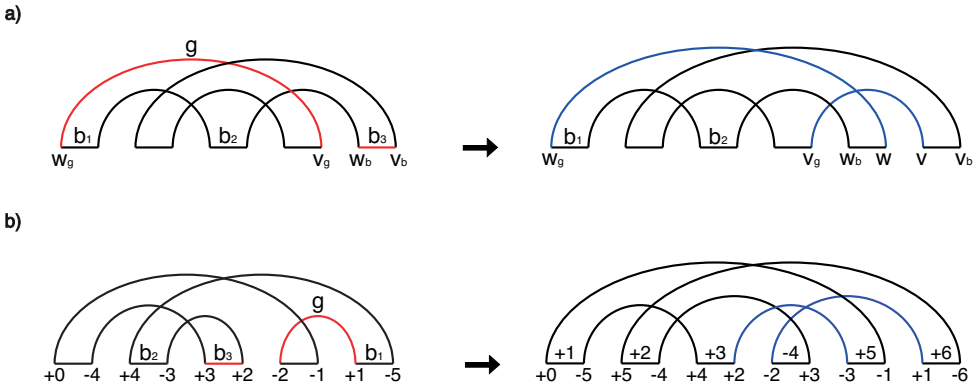

Figura 1.2: a) Exemplo do procedimento para transformar a permutação em uma permutação simples. b) Exemplo com a permutação $\pi = (+4\ +3\ -2\ +1)$ sendo transformada em uma permutação simples $\hat{\pi} = (+5\ +4\ -2\ -3\ +1)$.

reversão deverá ser adaptada para afetar somente os elementos $[a, c]$ de π. Caso a operação afete elementos que existem apenas em $\hat{\pi}$, então nada deve ser aplicado em π.

No exemplo da Figura 1.2 b), podemos ver o processo de quebra de ciclos longos em ciclos curtos sendo aplicado na permutação $\pi = (+4\ +3\ -2\ +1)$ para obtenção da permutação simples $\hat{\pi} = (+5\ +4\ -2\ -3\ +1)$.

Capítulo 2

Heurísticas para Ordenação de Permutação com Sinais por Reversões e Transposições

O uso de heurísticas na área de rearranjo de genomas para melhorar o resultado de algoritmos é uma prática frequente [12, 13]. Neste capítulo, iremos apresentar algumas heurísticas criadas para melhorar o resultado de algoritmos para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições. Abordaremos o problema na sua forma clássica e também nas versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo. Para verificar o comportamento das heurísticas e seus resultados, implementamos diversos algoritmos conhecidos na literatura. Propositalmente, escolhemos alguns que não foram desenvolvidos para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições, mas suas soluções são factíveis ou alguma modificação foi realizada para torná-las factíveis. O intuito de utilizarmos alguns algoritmos que não foram projetados diretamente para o problema em questão é verificar se aplicando nossas heurísticas em tais algoritmos poderíamos conseguir bons resultados para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições.

Este capítulo está dividido da seguinte forma. Na Subseção 2.1 será apresentada uma breve descrição de todos os algoritmos utilizados por nossas heurísticas. Em seguida, na Subseção 2.2, explicaremos como os conjuntos de permutações utilizados como parte da entrada foram construídos, bem como os limitantes inferiores adotados. Dando continuidade no capítulo, nas subseções 2.3, 2.4, 2.5 e 2.6, todas as heurísticas serão explicadas e seus resultados práticos exibidos. Por fim, na Subseção 2.7, um comparativo entre as heurísticas será apresentado.

2.1 Algoritmos Utilizados

Nesta subseção, apresentaremos os algoritmos utilizados por nossas heurísticas especificando seus respectivos fatores de aproximação, autores e para qual problema em específico os mesmos foram projetados. A Tabela 2.1 mostra uma lista dos algoritmos. Em seguida, são apresentadas breves descrições dos mesmos.

Tabela 2.1: Algoritmos utilizados pelas heurísticas. A primeira coluna mostra para qual problema o algoritmo foi desenvolvido. A segunda coluna apresenta o código identificador de cada algoritmo que será utilizado durante todo o capítulo. Na terceira coluna temos as referências dos trabalhos onde os algoritmos foram apresentados. A quarta coluna apresenta as complexidades dos algoritmos e a última coluna mostra o fator de aproximação dos algoritmos para o problema que foram desenvolvidos e também o fator de aproximação para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições.

Problemas Clássicos	**Código**	**Referência**	**Complexidade**	**Aproximação**
Reversão e Transposição	RSH	Rahman *et al.* [29]	$\mathcal{O}(n^3)$	$2k$ / 2
Reversão e Transposição com Sinal	WDM	Walter *et al.* [32]	$\mathcal{O}(n^3)$	2 / 2
	BRPT	Walter *et al.* [32]	$\mathcal{O}(n^2)$	3 / 3
	BRPR	Walter *et al.* [32]	$\mathcal{O}(n^2)$	3 / 3
Reversão com Sinal	HPB	Hannenhalli and Pevzner [24]	$\mathcal{O}(n^2)$	1 / 3
		Bader *et al.* [1]	$\mathcal{O}(n)$	1 / 3
Transposição	BP	Bafna and Pevzner [3]	$\mathcal{O}(n^2)$	1.5/ $\mathcal{O}(n)$
Problemas de Prefixo	**Código**	**Referência**	**Complexidade**	**Aproximação**
Reversão com Sinal	2-SPR	Lintzmayer *et al.* [27]	$\mathcal{O}(n^2)$	2 / 6
Reversão e Transposição com Sinal	2-SPRT	Lintzmayer *et al.* [27]	$\mathcal{O}(n^2)$	$2 + \frac{4}{b_p(\pi)}$ / $2 + \frac{4}{b_p(\pi)}$
	4-SPRT	Este trabalho	$\mathcal{O}(n^2)$	4 / 4
Problemas de Prefixo ou Sufixo	**Código**	**Referência**	**Complexidade**	**Aproximação**
Reversão com Sinal	2-SPSR	Lintzmayer *et al.* [27]	$\mathcal{O}(n^2)$	2 / 6
Reversão e Transposição com Sinal	2-SPSRT	Lintzmayer *et al.* [27]	$\mathcal{O}(n^2)$	$2 + \frac{4}{b_{ps}(\pi)}$ / $2 + \frac{4}{b_{ps}(\pi)}$

- RSH: algoritmo aproximado projetado para o problema de Ordenação de Permutações sem Sinais por Reversões e Transposições com fator de aproximação $2k$, onde k representa o fator de aproximação do algoritmo utilizado para decomposição de ciclos em um grafo construído a partir da permutação π. Esse algoritmo apresenta uma solução factível se aplicado na versão com sinais do problema. Como a decomposição de ciclos em um grafo construído a partir de uma permutação com sinais é única, esse algoritmo apresenta um fator de aproximação 2 para a versão com sinais do problema.

- WDM: algoritmo aproximado projetado para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições, garantindo um fator de aproximação 2.

- BRPT: algoritmo aproximado projetado para o problema de Ordenação de Permutações com e sem Sinais por Reversões e Transposições, garantindo um fator de aproximação 3. O algoritmo adota uma estratégia gulosa buscando remover o maior número de *breakpoints* da permutação a cada iteração. Caso existam empates entre as operações de reversão e transposição, uma operação de transposição será priorizada.

- BRPR: algoritmo aproximado projetado para o problema de Ordenação de Permutações com e sem Sinais por Reversões e Transposições, garantindo um fator de aproximação 3. Esse algoritmo é uma variação do algoritmo BRPT, diferenciando apenas no caso de empate, onde uma operação de reversão será priorizada.

- HPB: algoritmo exato para o problema de Ordenação de Permutações com Sinais por Reversões. Para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições esse algoritmo garante um fator de aproximação 3. Como a heurística *Look Ahead* (Seção 2.4) requer apenas um estimador para distância das permutações, utilizamos um algoritmo que executa em tempo linear e fornece apenas a distância. A heurística *Sliding Window* (Seção 2.3) necessita de uma sequência inicial de eventos para construir a sequência de permutações. Dessa forma, utilizamos um algoritmo quadrático. As implementações de ambos os algoritmos foram fornecidas por Tesler [30, 31].

- BP: algoritmo aproximado para o problema de Ordenação de Permutações sem Sinais por Transposições, garantindo um fator de aproximação $\frac{3}{2}$. Para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições esse algoritmo garante um fator de aproximação $\mathcal{O}(n)$. Para fornecer uma permutação válida para o algoritmo, todas as *strips* negativas foram transformadas em *strips* positivas utilizando eventos de reversão. Com todas as *strips* positivas, a permutação foi transformada em uma permutação sem sinais e o algoritmo executado. A sequência de ordenação final é composta pelas operações de reversão unidas com a sequência fornecida pelo algoritmo.

- 2-SPR: algoritmo aproximado para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo, garantindo um fator de aproximação 2. Para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo e Transposições de Prefixo esse algoritmo garante um fator de aproximação 6. A sequência de ordenação desse algoritmo é uma sequência válida para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo e Transposições de Prefixo.

- 2-SPRT: algoritmo aproximado para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo e Transposições de Prefixo, garantindo um fator de aproximação $(2 + 4/b_p(\pi))$, onde $b_p(\pi)$ é o número de *breakpoints* de prefixo.

- 4-SPRT: algoritmo aproximado para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo e Transposições de Prefixo, garantindo um fator de aproximação 4. O algoritmo executa de forma gulosa, removendo sempre o último *breakpoint* de prefixo. Caso a *strip* necessária para remover esse *breakpoint* seja negativa então uma reversão de prefixo e um transposição de prefixo são aplicadas, caso contrário, apenas uma transposição de prefixo é utilizada.

- 2-SPSR: algoritmo aproximado para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo ou Sufixo, garantindo um fator de aproximação 2.

Para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo ou Sufixo e Transposições de Prefixo ou Sufixo esse algoritmo garante um fator de aproximação 6. A sequência de ordenação desse algoritmo é uma sequência válida para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo ou Sufixo e Transposições de Prefixo ou Sufixo.

- 2-SPSRT: algoritmo aproximado para o problema de Ordenação de Permutações com Sinais por Reversões de Prefixo ou Sufixo e Transposições de Prefixo e Sufixo, garantindo um fator de aproximação de $(2 + 4/b_{ps}(\pi))$, onde $b_{ps}(\pi)$ é o número de *breakpoints* de prefixo ou sufixo.

No restante deste capítulo, iremos referenciar os algoritmos pelos códigos apresentados na Tabela 2.1.

2.2 Conjuntos de Permutações

Para comparar as heurísticas que serão apresentadas nesse trabalho, criamos conjuntos de permutações que foram utilizados como parte da entrada pelas heurísticas.

Cada conjunto apresenta 1000 permutações de um tamanho específico. O identificador de cada conjunto é o tamanho das permutações contidas no mesmo. As permutações de cada conjunto foram geradas de maneira aleatória, mas mantendo o maior número possível de *breakpoints*. Um total de 18 conjuntos foram criados, com identificadores de 10 até 100 em intervalos de 10, e de 150 até 500 em intervalos de 50.

Foram computados três limitantes inferiores distintos para cada permutação, sendo os mesmos referentes às versões sem restrições nas operações, restrita a operações de prefixo e restrita a operações de prefixo ou sufixo. Para a versão sem restrições nas operações, o limitante inferior adotado foi $\lceil \frac{(n+1)-c(\pi)}{2} \rceil$ [29], onde $c(\pi)$ representa o número de ciclos em um grafo de ciclos. Para a versão restrita a operações de prefixo adotamos $\lceil \frac{b_p(\pi)}{2} \rceil$ [27]. Para a versão restrita a operações de prefixo ou sufixo, o limitante inferior adotado foi $\lceil \frac{b_{ps}(\pi)}{2} \rceil$ [27].

Os conjuntos de permutações estão disponíveis no seguinte endereço eletrônico: `https://www.ic.unicamp.br/~klairton/dataset/SbRT/`.

2.3 Sliding Window

A heurística *Sliding Window* (SW) utiliza uma base de dados construída por Galvão e Dias [21]. Nessa base de dados, encontra-se uma ordenação ótima de qualquer permutação com sinais de tamanho até nove. Além disso, é utilizado o conceito de permutação reduzida que foi apresentado por Christie [10].

A heurística recebe como entrada uma permutação π e um algoritmo alg. Utilizando o algoritmo alg para gerar uma ordenação inicial para a permutação π, uma sequência de permutações $S = [\pi^0, \pi^1, \pi^2, \ldots, \pi^z]$ é construída, tal que $\pi^i \circ \delta = \pi^{i+1}$, onde $\delta \in \{\rho, \tau\}$, para $0 \leq i < z$. A saída dessa heurística é uma sequência de ordenação $S' = [\pi^0, \pi^1, \pi^2, \ldots, \pi^y]$, tal que $y \leq z$ e $\pi^y = \pi^z$. Inicialmente, a heurística encontra uma subsequência de

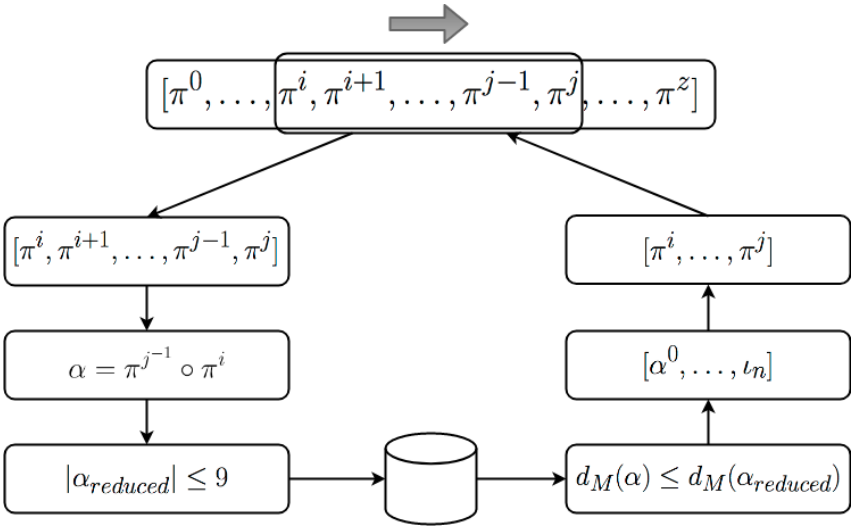

Figura 2.1: Esquema da heurística *Sliding Window*.

permutações S^w que chamamos de janela, onde a subsequência começa em uma permutação π^i e termina em uma permutação π^j, tal que $0 \leq i < j \leq z$. Em seguida, a heurística computa uma permutação $\alpha = \pi^{j^{-1}} \circ \pi^i$. A permutação α é então reduzida em uma permutação $\alpha_{reduced}$ e, caso o tamanho da permutação $\alpha_{reduced}$ seja maior que nove, uma nova subsequência S^w será buscada, reduzindo o tamanho da janela (subsequência S^w). Caso contrário, a ordenação ótima da permutação $\alpha_{reduced}$ será verificada na base de dados. Caso o tamanho da sequência que ordena a permutação $\alpha_{reduced}$ seja menor que o tamanho da subsequência S^w, será construída uma sequência $S^{w'}$ que ordena a permutação α. Cada permutação α' pertencente à sequência $S^{w'}$ será substituída por $\pi^j \circ \alpha'$ e, finalmente, a subsequência S^w será substituída pela sequência $S^{w'}$, reduzindo o tamanho da sequência S. Um esquema dessa heurística pode ser visto na Figura 2.1.

O Algoritmo 1 mostra o pseudocódigo da heurística *Sliding Window*.

O tempo de execução dessa heurística é $\mathcal{O}(n + alg(n))$, onde $alg(n)$ é o tempo de execução do algoritmo fornecido como entrada. Como essa heurística acrescenta apenas um processamento linear ao algoritmo que é dado como entrada, seu tempo de execução é muito rápido.

Algoritmo 1: SlidingWindow(alg,π)

Entrada:
alg: Algoritmo utilizado para construir a sequência de permutações;
π: Permutação para ser ordenada;
Saída:
Sequência de ordenação de π;

```
1  início
2      S ← ConstruaSequenciaPermutacoes(alg, π);
3      i ← 0;
4      j ← 2;
5      enquanto j ≤ Tamanho(S) faça
6          S^w ← ObtenhaSubsequencia(S, i, j);
7          α ← ComputeAlpha(S^w);
8          α_reduced ← ReduzaAlpha(α);
9          se Tamanho(α_reduced) ≤ 9 então
10             S^w' ← OrdenacaoOtima(α_reduced);
11             se Tamanho(S^w') < Tamanho(S^w) então
12                 S^w' ← AtualizePermutacoes(S^w',π^j);
13                 S^w ← S^w';
14                 i ← i + 1;
15                 j ← i + Tamanho(S^w');
16             senão
17                 j ← j + 1;
18         senão
19             i ← i + 1;
20     retorna SequenciaOrdenacao(S);
```

2.3.1 Resultados

A heurística SW foi aplicada em todos os algoritmos mencionados na Tabela 2.1 e executada com todos os conjuntos de permutações. As figuras 2.2, 2.3, 2.4 e 2.5 mostram o fator de aproximação médio dos algoritmos em comparação com o obtido pela heurística SW.

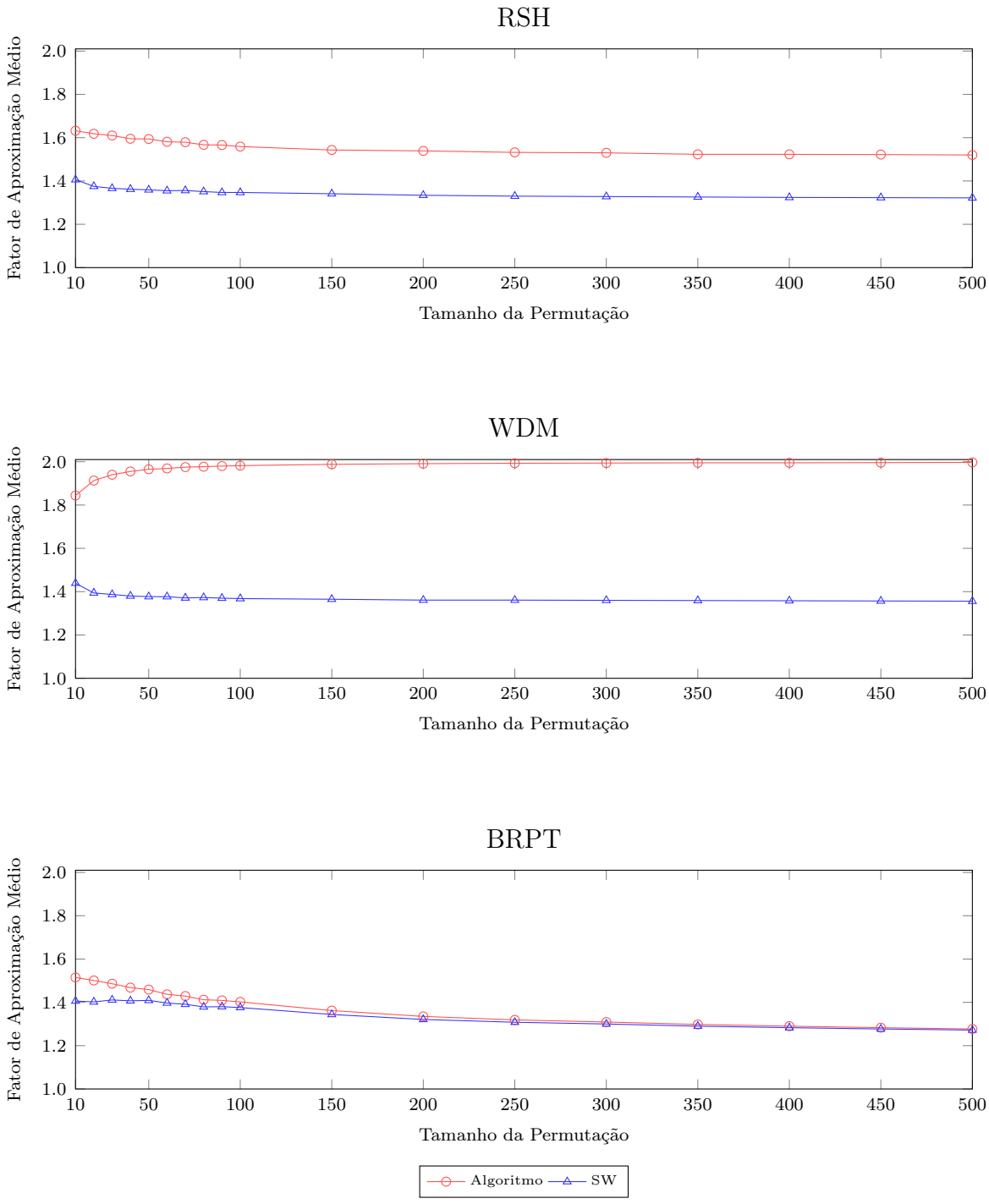

Figura 2.2: Fator de aproximação médio da heurística *Sliding Window* aplicada nos algoritmos RSH, WDM e BRPT.

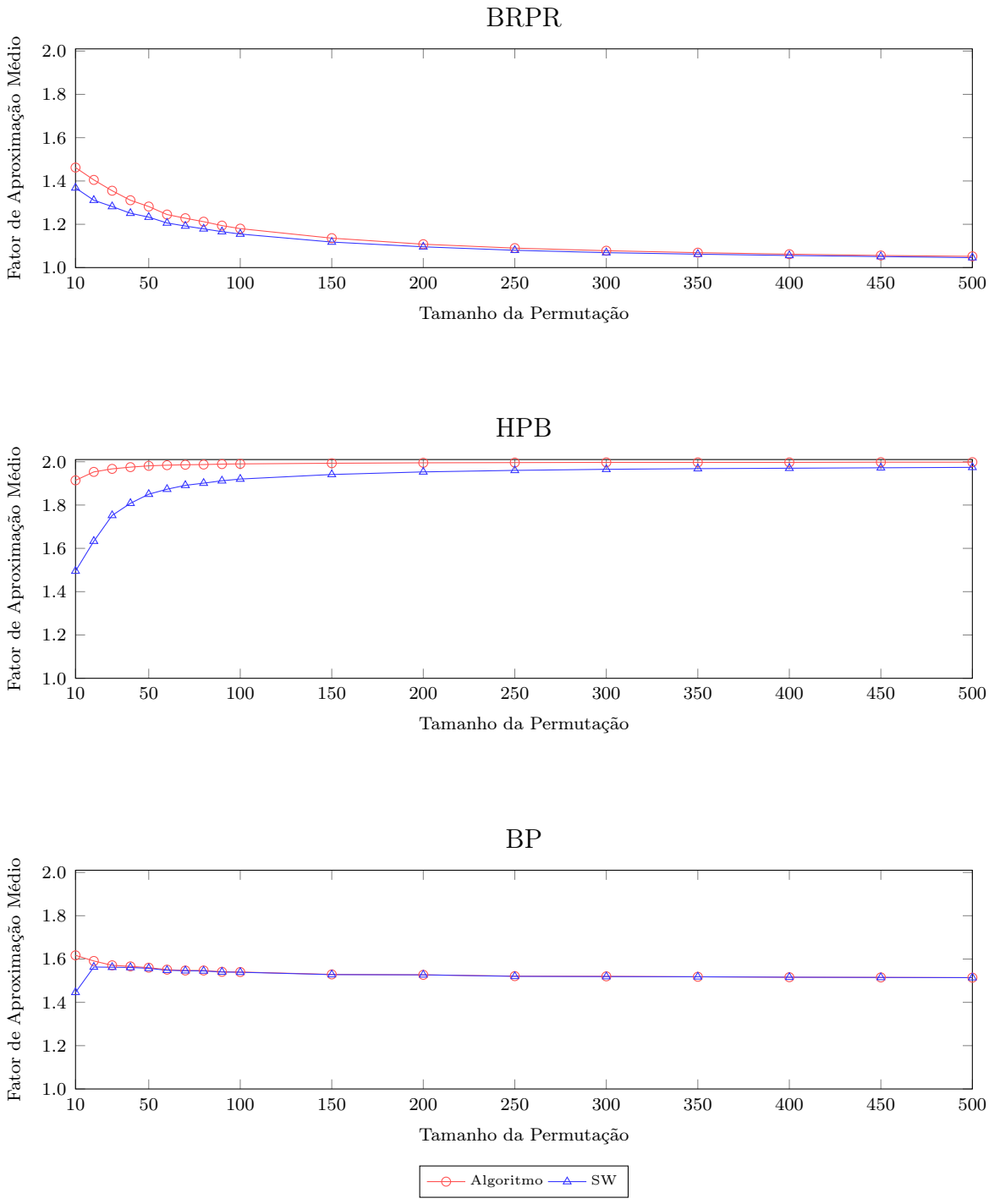

Figura 2.3: Fator de aproximação médio da heurística *Sliding Window* aplicada nos algoritmos BRPR, HPB e BP.

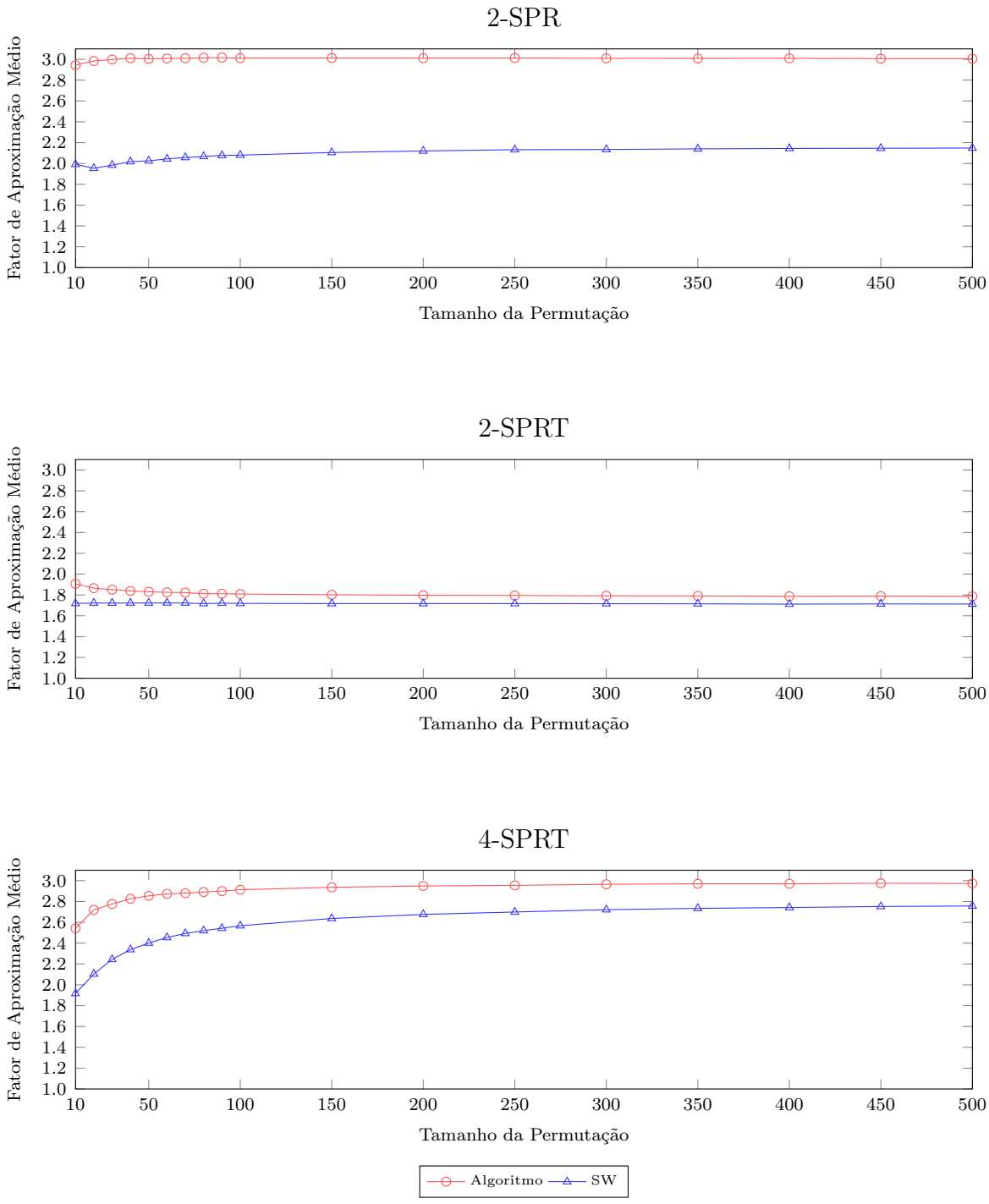

Figura 2.4: Fator de aproximação médio da heurística *Sliding Window* com restrição de operações de prefixo.

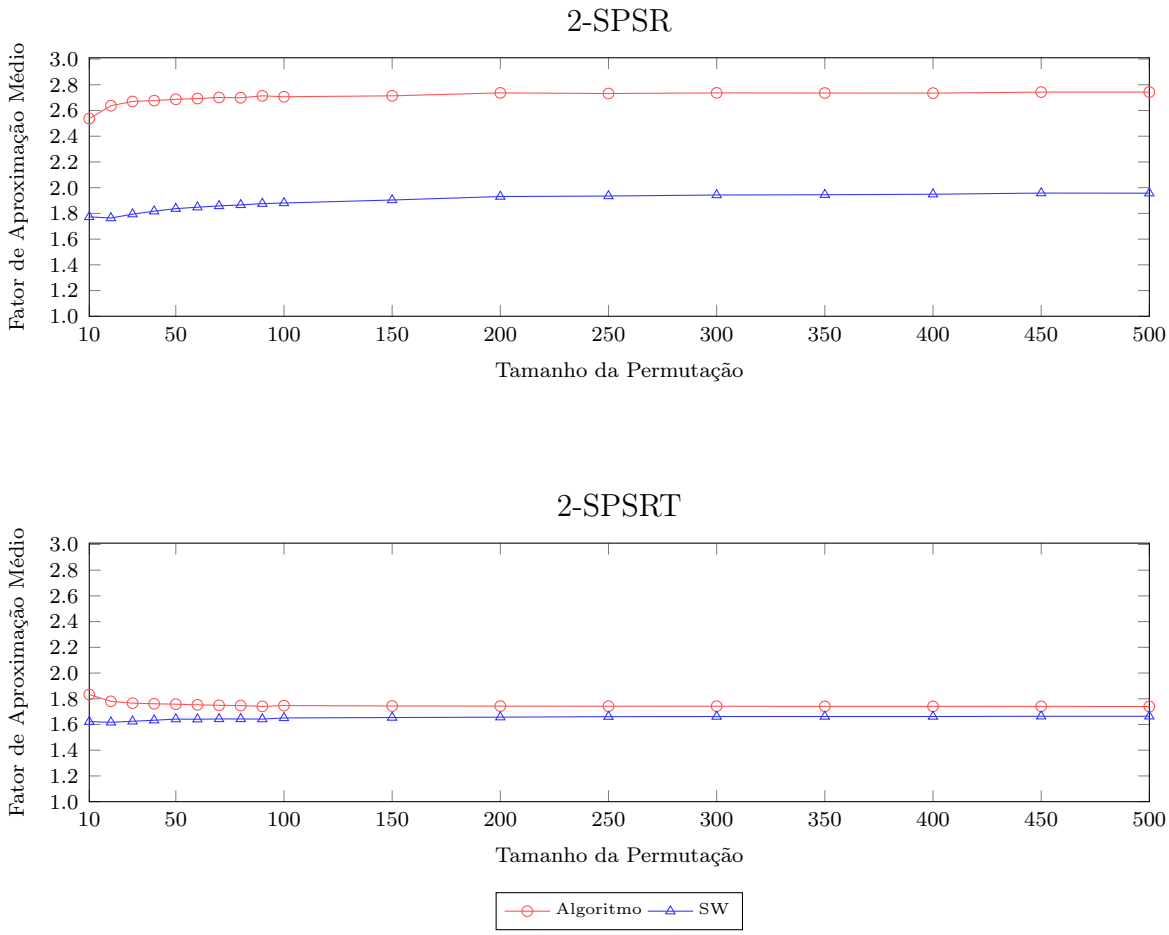

Figura 2.5: Fator de aproximação médio da heurística *Sliding Window* com restrição de operações de prefixo ou sufixo.

Podemos ressaltar a melhoria significativa no fator de aproximação médio dos algoritmos RSH e WDM para a versão sem restrição nas operações. Para as versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo, essa melhoria pode ser observada em todos os algoritmos. A Figura 2.6 mostra a porcentagem de soluções que foram melhoradas utilizando a heurística SW.

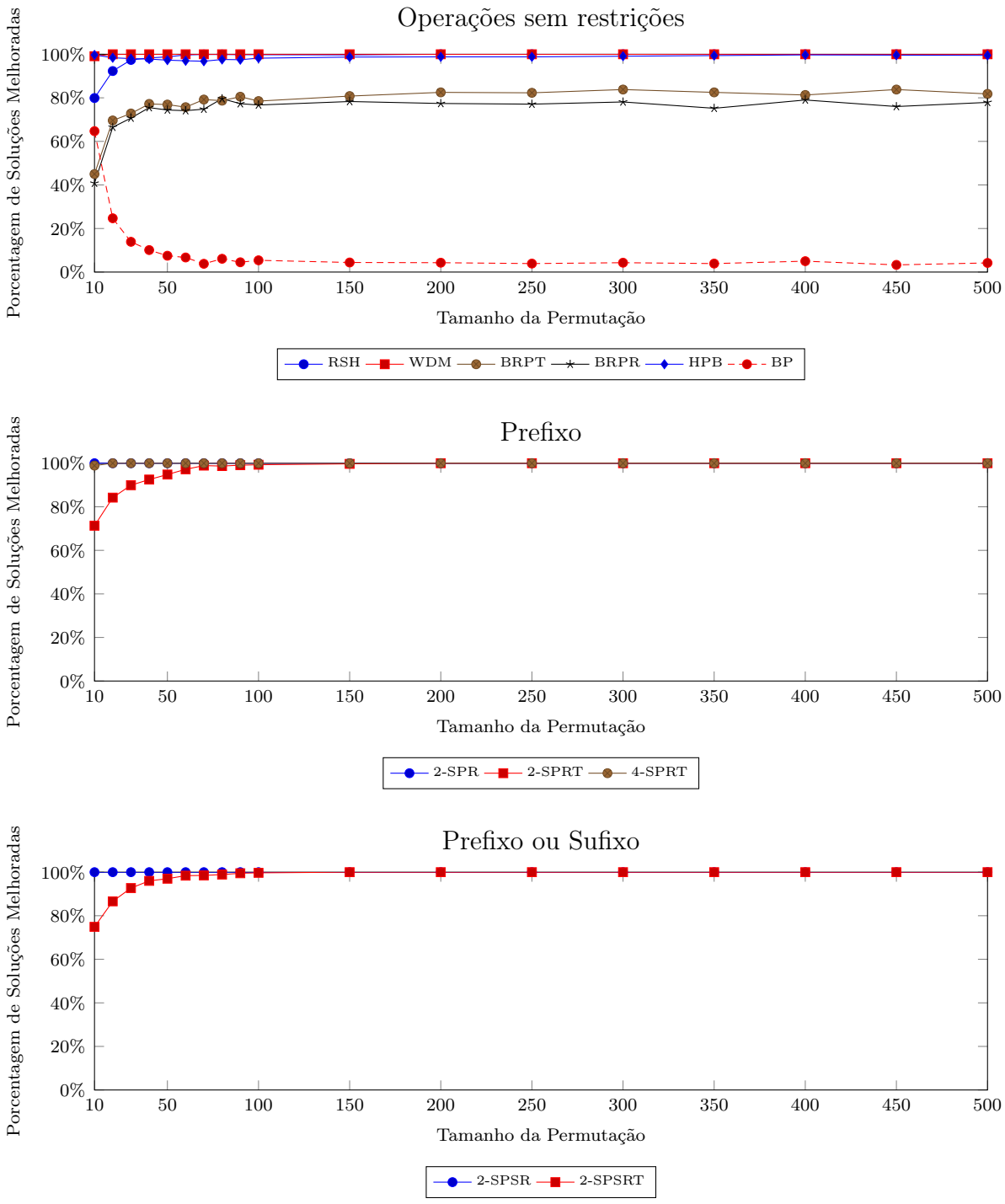

Figura 2.6: Porcentagem de soluções melhoradas utilizando a heurística *Sliding Window*.

Note que, para a versão sem restrições nas operações, com exceção do algoritmo BP, foi possível melhorar as soluções iniciais dos algoritmos em mais de 70% dos casos para permutações com tamanho maior que 20. Para as versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo esse valor sobe para mais de 90%, sem exceção de nenhum algoritmo. A Tabela 2.2 mostra o tempo médio, em segundos, gasto em cada permutação pela heurística SW.

Tabela 2.2: Tempo de execução médio, em segundos, da heurística *Sliding Window*.

Operações sem Restrições						
	Algoritmo					
Tamanho	RSH	WDM	BRPT	BRPR	HPB	BP
50	0.007s	0.007s	0.004s	0.003s	0.003s	0.003s
100	0.016s	0.017s	0.007s	0.007s	0.009s	0.005s
150	0.021s	0.025s	0.008s	0.007s	0.012s	0.011s
200	0.028s	0.031s	0.009s	0.008s	0.016s	0.014s
250	0.039s	0.042s	0.011s	0.012s	0.022s	0.017s
300	0.045s	0.049s	0.013s	0.013s	0.028s	0.021s
350	0.051s	0.059s	0.015s	0.015s	0.032s	0.025s
400	0.062s	0.068s	0.017s	0.017s	0.037s	0.028s
450	0.073s	0.080s	0.019s	0.017s	0.043s	0.031s
500	0.085s	0.092s	0.021s	0.019s	0.049s	0.035s

Operações de Prefixo			
	Algoritmo		
Tamanho	2-SPR	2-SPRT	4-SPRT
50	0.007s	0.001s	0.001s
100	0.017s	0.003s	0.005s
150	0.025s	0.006s	0.009s
200	0.032s	0.009s	0.014s
250	0.038s	0.014s	0.019s
300	0.045s	0.021s	0.024s
350	0.051s	0.023s	0.029s
400	0.060s	0.026s	0.033s
450	0.066s	0.029s	0.037s
500	0.072s	0.033s	0.041s

Operações de Prefixo ou Sufixo		
	Algoritmo	
Tamanho	2-SPSR	2-SPSRT
50	0.016s	0.004s
100	0.029s	0.010s
150	0.032s	0.012s
200	0.034s	0.014s
250	0.037s	0.016s
300	0.039s	0.018s
350	0.040s	0.020s
400	0.042s	0.021s
450	0.044s	0.023s
500	0.046s	0.025s

Pela Tabela 2.2, percebemos que a heurística SW executa de maneira extremamente rápida e que, na média, forneceu uma solução em menos de um décimo de segundo. As tabelas 2.3, 2.4 e 2.5 mostram, respectivamente, o fator de aproximação mínimo, médio e máximo da heurística SW em comparação com os algoritmos sem nenhuma heurística aplicada.

Tabela 2.3: Fator de aproximação mínimo da heurística *Sliding Window*.

Operações sem Restrições						
Algoritmo						
Tamanho	RSH/SW	WDM/SW	BRPT/SW	BRPR/SW	HPB/SW	BP/SW
50	1.304/1.160	1.750/1.200	1.120/1.080	1.120/1.120	1.957/1.625	1.280/1.280
100	1.306/1.220	1.900/1.240	1.122/1.102	1.080/1.060	1.979/1.780	1.380/1.380
150	1.284/1.253	1.933/1.253	1.123/1.123	1.067/1.067	1.986/1.849	1.373/1.373
200	1.313/1.263	1.950/1.265	1.120/1.120	1.060/1.050	1.990/1.867	1.400/1.400
250	1.303/1.258	1.960/1.256	1.122/1.098	1.048/1.048	1.992/1.895	1.392/1.384
300	1.311/1.257	1.973/1.287	1.134/1.128	1.040/1.040	1.993/1.913	1.420/1.420
350	1.275/1.249	1.977/1.283	1.086/1.086	1.040/1.040	1.994/1.925	1.428/1.428
400	1.330/1.268	1.975/1.291	1.131/1.131	1.035/1.030	1.995/1.925	1.437/1.434
450	1.300/1.259	1.978/1.286	1.143/1.125	1.031/1.031	1.995/1.933	1.418/1.418
500	1.294/1.263	1.984/1.293	1.081/1.077	1.028/1.028	1.996/1.931	1.440/1.440

Operações de Prefixo			
Algoritmo			
Tamanho	2-SPR/SW	2-SPRT/SW	4-SPRT/SW
50	2.600/1.720	1.600/1.480	2.400/2.040
100	2.680/1.820	1.620/1.560	2.580/2.340
150	2.747/1.907	1.653/1.587	2.653/2.440
200	2.800/1.890	1.670/1.600	2.640/2.490
250	2.784/1.984	1.680/1.616	2.736/2.560
300	2.793/1.960	1.700/1.627	2.767/2.567
350	2.851/2.023	1.686/1.623	2.754/2.606
400	2.845/2.015	1.705/1.625	2.815/2.635
450	2.862/2.036	1.707/1.636	2.822/2.649
500	2.852/2.020	1.712/1.636	2.820/2.632

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	2-SPSR/SW	2-SPSRT/SW
50	2.240/1.480	1.480/1.360
100	2.360/1.600	1.560/1.480
150	2.400/1.667	1.587/1.507
200	2.440/1.730	1.610/1.530
250	2.456/1.704	1.616/1.544
300	2.460/1.727	1.627/1.547
350	2.503/1.766	1.623/1.549
400	2.485/1.750	1.645/1.565
450	2.493/1.791	1.618/1.551
500	2.484/1.764	1.648/1.560

Pela Tabela 2.3, podemos perceber que a heurística SW conseguiu reduzir de maneira significativa o fator de aproximação mínimo em grande parte dos algoritmos, com exceção dos algoritmos BRPR, HPB e BP. Vale ressaltar que o fator de aproximação mínimo fornecido pelo algoritmo BRPR é muito próximo do ótimo, sendo assim, difícil de obter valores menores.

Tabela 2.4: Fator de aproximação médio da heurística *Sliding Window*.

Operações sem Restrições						
	Algoritmo					
Tamanho	RSH/SW	WDM/SW	BRPT/SW	BRPR/SW	HPB/SW	BP/SW
50	1.594/1.359	1.965/1.377	1.459/1.409	1.282/1.233	1.981/1.850	1.560/1.557
100	1.559/1.347	1.982/1.368	1.402/1.376	1.180/1.155	1.990/1.920	1.540/1.539
150	1.543/1.341	1.988/1.365	1.362/1.344	1.136/1.118	1.993/1.941	1.529/1.528
200	1.539/1.334	1.991/1.361	1.335/1.321	1.108/1.096	1.995/1.953	1.527/1.527
250	1.532/1.330	1.993/1.361	1.319/1.308	1.090/1.080	1.996/1.960	1.521/1.520
300	1.530/1.328	1.994/1.360	1.309/1.300	1.078/1.069	1.997/1.965	1.520/1.519
350	1.523/1.326	1.995/1.359	1.298/1.290	1.069/1.062	1.997/1.968	1.518/1.518
400	1.523/1.324	1.995/1.358	1.290/1.283	1.062/1.056	1.997/1.970	1.516/1.516
450	1.522/1.323	1.996/1.357	1.283/1.277	1.056/1.051	1.998/1.972	1.515/1.515
500	1.520/1.322	1.997/1.356	1.277/1.272	1.052/1.046	1.998/1.974	1.514/1.514

Operações de Prefixo			
	Algoritmo		
Tamanho	2-SPR/SW	2-SPRT/SW	4-SPRT/SW
50	3.004/2.025	1.831/1.723	2.854/2.401
100	3.011/2.080	1.809/1.719	2.913/2.567
150	3.012/2.105	1.802/1.717	2.936/2.637
200	3.011/2.120	1.797/1.717	2.949/2.676
250	3.012/2.132	1.795/1.717	2.955/2.699
300	3.009/2.134	1.792/1.716	2.965/2.721
350	3.008/2.140	1.791/1.715	2.970/2.734
400	3.009/2.144	1.787/1.712	2.970/2.742
450	3.006/2.146	1.789/1.714	2.975/2.752
500	3.006/2.148	1.787/1.713	2.973/2.757

Operações de Prefixo ou Sufixo		
	Algoritmo	
Tamanho	2-SPSR/SW	2-SPSRT/SW
50	2.687/1.837	1.758/1.642
100	2.707/1.881	1.747/1.651
150	2.714/1.904	1.744/1.654
200	2.737/1.931	1.743/1.657
250	2.732/1.935	1.742/1.661
300	2.737/1.943	1.742/1.662
350	2.736/1.945	1.741/1.662
400	2.735/1.949	1.741/1.662
450	2.743/1.958	1.741/1.664
500	2.743/1.957	1.740/1.664

Pela Tabela 2.4, podemos perceber que existe uma grande diferença na qualidade da solução entre os algoritmos originais RSH e WDM. Depois que a heurística SW foi aplicado nesses algoritmos, a qualidade da solução melhorou significativamente e não existe mais uma grande diferença entre o valor de aproximação médio desses algoritmos. Melhorias relevantes no fator de aproximação médio podem ser observadas em quase todos os algoritmos, com exceção dos algoritmos BRPT, BRPR, HPB e BP.

Tabela 2.5: Fator de aproximação máximo da heurística *Sliding Window*.

Operações sem Restrições						
Algoritmo						
Tamanho	RSH/SW	WDM/SW	BRPT/SW	BRPR/SW	HPB/SW	BP/SW
50	1.880/1.565	2.000/1.652	1.875/1.720	1.522/1.478	2.000/2.000	1.913/1.913
100	1.800/1.479	2.000/1.531	1.653/1.633	1.306/1.286	2.000/2.000	1.714/1.714
150	1.733/1.432	2.000/1.507	1.608/1.595	1.222/1.205	2.000/2.000	1.676/1.676
200	1.740/1.430	2.000/1.485	1.535/1.515	1.192/1.162	2.000/2.000	1.663/1.663
250	1.744/1.403	2.000/1.467	1.492/1.468	1.163/1.138	2.000/2.000	1.637/1.637
300	1.725/1.403	2.000/1.463	1.487/1.453	1.134/1.121	2.000/2.000	1.624/1.624
350	1.703/1.382	2.000/1.453	1.460/1.437	1.110/1.098	2.000/2.000	1.618/1.613
400	1.685/1.382	2.000/1.424	1.437/1.426	1.102/1.091	2.000/2.000	1.603/1.603
450	1.700/1.375	2.000/1.430	1.411/1.406	1.103/1.080	2.000/2.000	1.601/1.601
500	1.692/1.373	2.000/1.440	1.378/1.375	1.085/1.072	2.000/2.000	1.592/1.592

Operações de Prefixo			
Algoritmo			
Tamanho	2-SPR/SW	2-SPRT/SW	4-SPRT/SW
50	3.440/2.360	2.040/1.920	3.320/2.760
100	3.280/2.300	1.960/1.860	3.180/2.780
150	3.240/2.307	1.933/1.867	3.200/2.853
200	3.230/2.320	1.940/1.830	3.210/2.830
250	3.240/2.280	1.904/1.824	3.144/2.832
300	3.240/2.287	1.887/1.827	3.140/2.827
350	3.200/2.297	1.880/1.811	3.154/2.863
400	3.190/2.265	1.865/1.800	3.130/2.860
450	3.191/2.271	1.862/1.804	3.124/2.836
500	3.156/2.280	1.856/1.796	3.108/2.860

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	2-SPSR/SW	2-SPSRT/SW
50	3.240/2.240	2.000/1.840
100	3.180/2.240	1.960/1.820
150	3.187/2.253	1.933/1.800
200	3.160/2.210	1.890/1.780
250	3.080/2.200	1.872/1.768
300	3.127/2.273	1.853/1.807
350	3.154/2.229	1.851/1.766
400	3.060/2.190	1.840/1.790
450	3.040/2.231	1.849/1.760
500	3.088/2.192	1.824/1.752

Observando a Tabela 2.5, podemos ver que o único algoritmo em que não foi possível reduzir o fator de aproximação máximo foi o HPB.

2.4 Look Ahead

Look Ahead (LA) é uma heurística que recebe como entrada uma permutação π e um algoritmo alg. Como saída, a heurística fornece uma sequência de eventos de rearranjo que, se aplicados na permutação π, produz como resultado uma permutação ordenada, ou seja, a permutação identidade ι_n. A heurística funciona da seguinte forma: chamamos a permutação π, dada como entrada, de permutação atual. Enquanto essa permutação não estiver ordenada, a heurística irá aplicar todos os possíveis eventos de reversão e transposição na permutação atual. Em decorrência da aplicação desses eventos, uma série

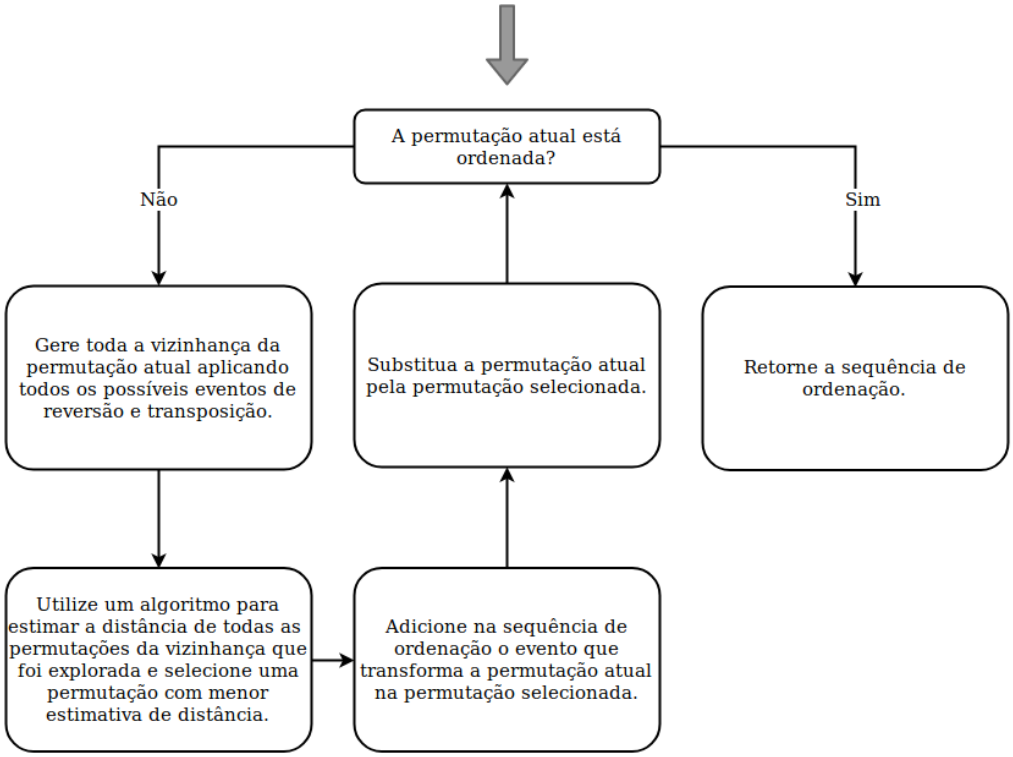

Figura 2.7: Esquema da heurística *Look Ahead*.

de permutações que estão a um evento de distância da permutação atual serão geradas. Para cada uma dessas permutações, é estimada a distância utilizando o algoritmo alg que foi dado como entrada. Após estimar a distância de cada permutação gerada, aquela que apresentar a menor estimativa de distância será escolhida (em caso de empates, uma delas será escolhida arbitrariamente). A permutação escolhida será a permutação atual na iteração seguinte. O processo termina quando a permutação atual estiver ordenada.

Note que a heurística necessita de um estimador de distância para aplicar uma operação a cada iteração. Caso essas estimativas fornecidas sejam ruins, a qualidade da solução da heurística pode ser impactada negativamente.

Na versão clássica de ordenação, a heurística apresenta um tempo de execução de $\mathcal{O}(n^3 \times alg(n))$, onde $alg(n)$ é o tempo de execução do algoritmo fornecido como entrada. Já nas versões de prefixo e prefixo ou sufixo o tempo de execução é dado por $\mathcal{O}(n^2 \times alg(n))$, onde $alg(n)$ é o tempo de execução do algoritmo fornecido como entrada. Devido ao fato da complexidade dessa heurística estar diretamente ligada à complexidade do algoritmo que é fornecido como entrada, em determinados casos, o seu uso pode tornar-se proibitivo. Um esquema dessa heurística pode ser visto na Figura 2.7.

O Algoritmo 2 mostra o pseudocódigo da heurística *Look Ahead*.

Algoritmo 2: LookAhead(alg,π)

Entrada:
alg: Algoritmo utilizado para estimar a distância de permutações;
π: Permutação para ser ordenada;
Saída:
Sequência de ordenação de π;

```
1  início
2      S ← {};
3      enquanto π não estiver ordenada faça
4          N ← GereVizinhanca(π);
5          EstimeDistancia(N,alg);
6          π' ← SelecionePermutacao(N);
7          e ← ObtenhaEvento(π,π');
8          S ← S ∪ {e};
9          π ← π';
10     retorna S;
```

2.4.1 Resultados

A heurística LA foi executada com todos os algoritmos descritos na Tabela 2.1 e com os conjuntos de permutações com identificadores de 10 até 100. Os conjuntos de permutações com identificadores de 150 até 500 não foram utilizados pelo fato do tempo de execução gasto por essa heurística ser mais elevado. As figuras 2.8, 2.9, 2.10 e 2.11 mostram o fator de aproximação médio dos algoritmos em comparação com o obtido pela heurística LA.

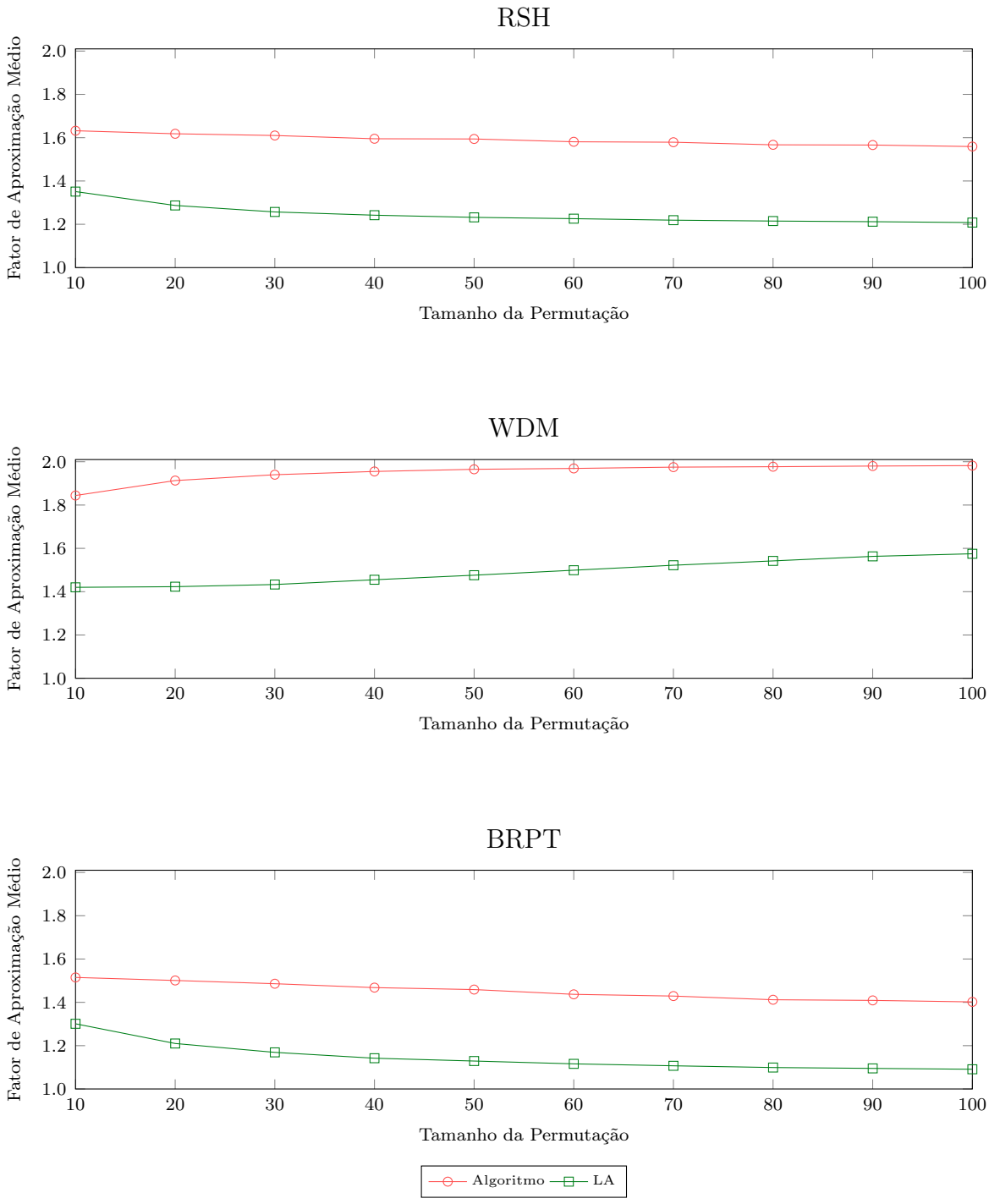

Figura 2.8: Fator de aproximação médio da heurística *Look Ahead* aplicada nos algoritmos RSH, WDM e BRPT.

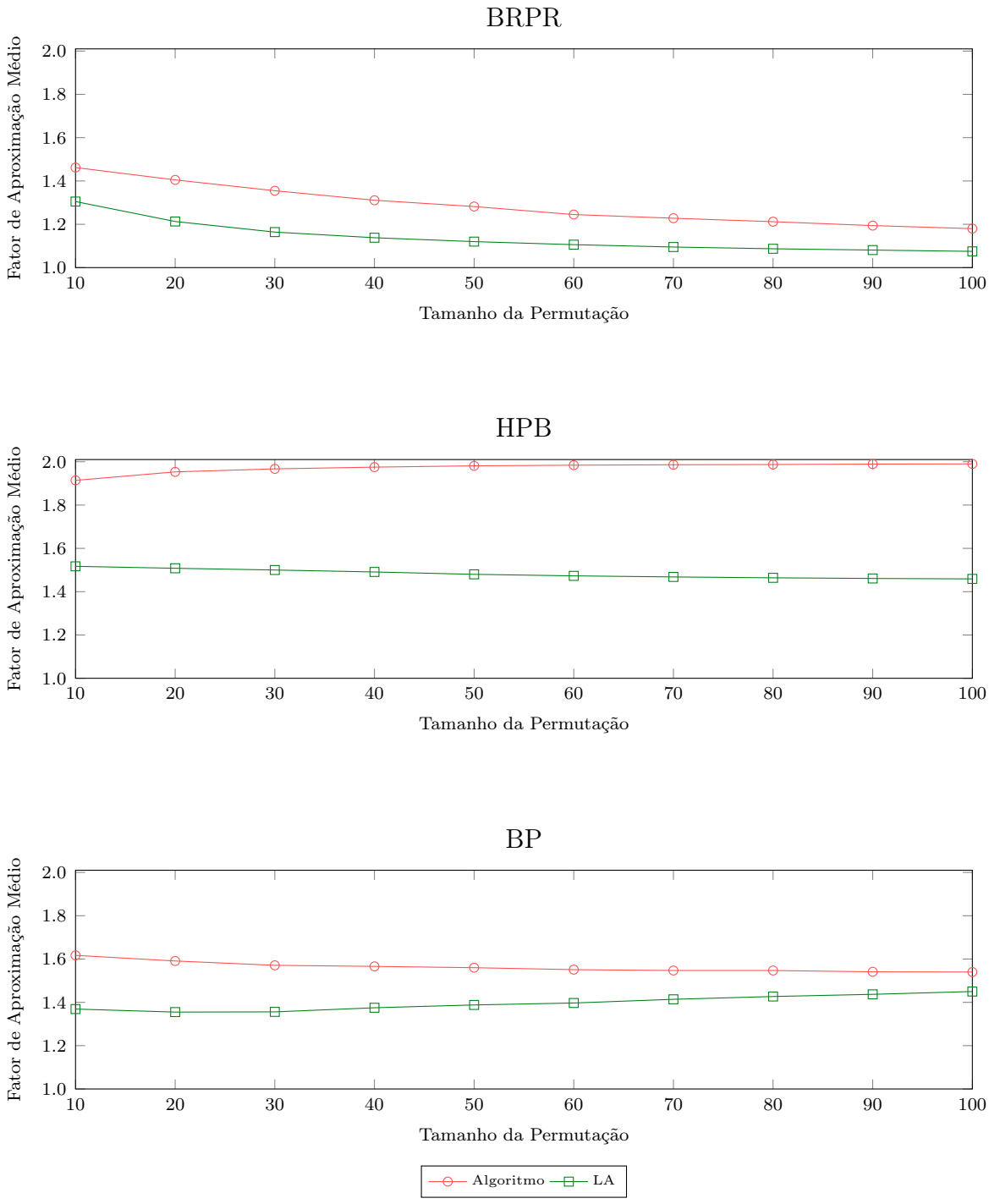

Figura 2.9: Fator de aproximação médio da heurística *Look Ahead* aplicada nos algoritmos BRPR, HPB e BP.

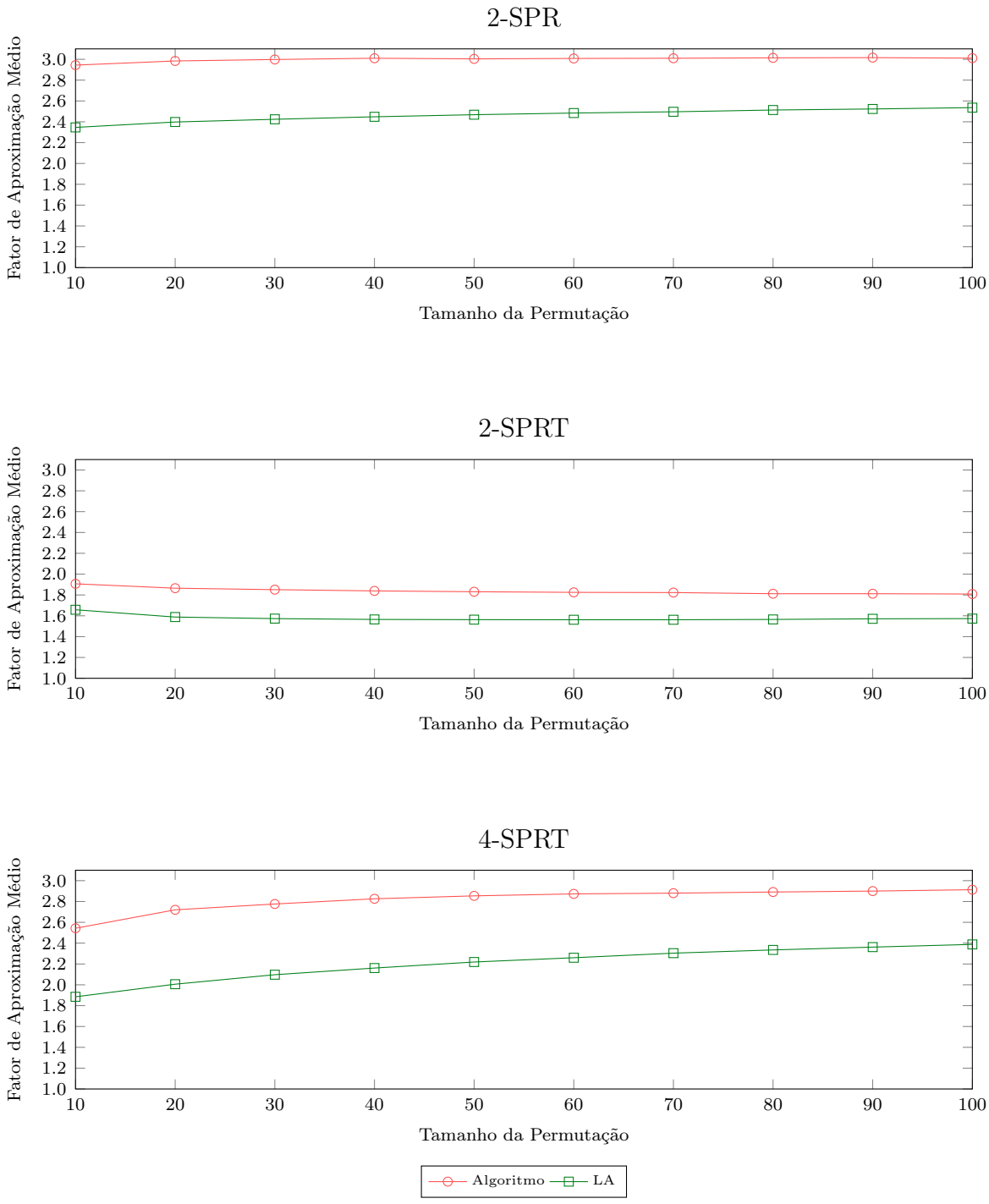

Figura 2.10: Fator de aproximação médio da heurística *Look Ahead* com restrição de operações de prefixo.

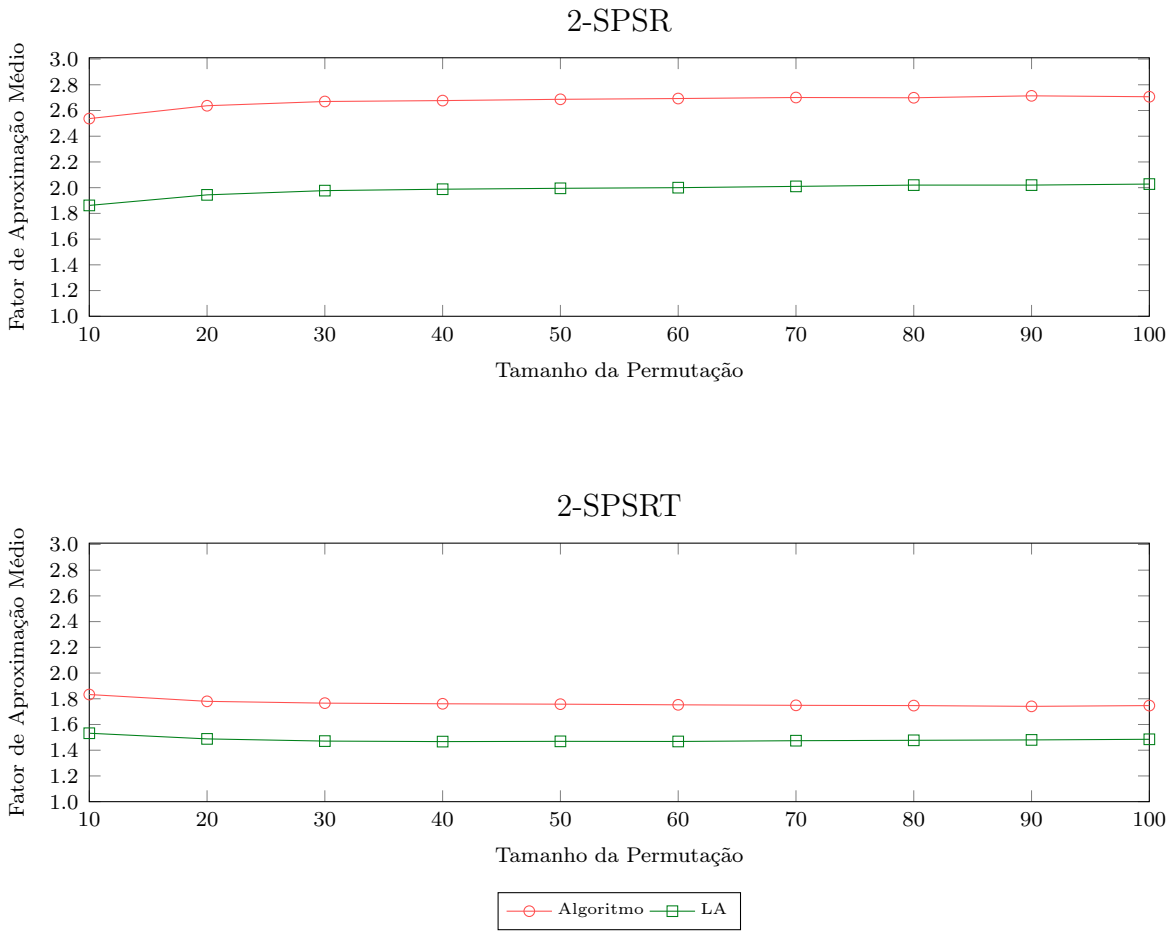

Figura 2.11: Fator de aproximação médio da heurística *Look Ahead* com restrição de operações de prefixo ou sufixo.

A heurística LA apresentou melhorias relevantes em todos os algoritmos, sem exceções. A Figura 2.12 mostra a porcentagem de soluções que foram melhoradas utilizando a heurística LA.

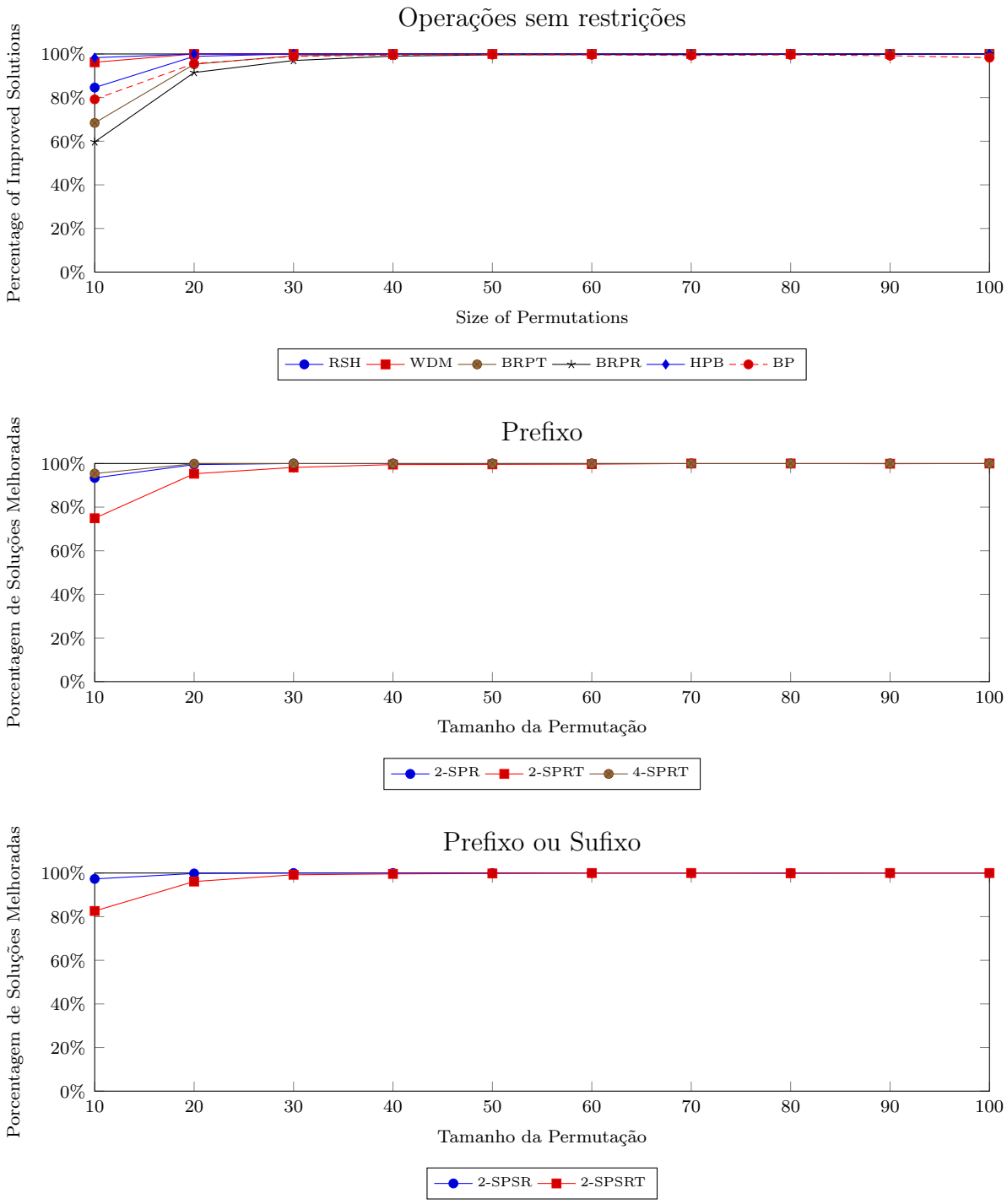

Figura 2.12: Porcentagem de soluções melhoradas utilizando a heurística *Look Ahead*.

Note pela Figura 2.12 que a heurística LA conseguiu melhorar a solução inicial fornecida pelos algoritmos em mais de 90% dos casos para permutações com tamanho maior que 10, em todas as versões. A Tabela 2.6 mostra o tempo médio, em segundos, gasto em cada permutação pela heurística LA.

Tabela 2.6: Tempo de execução médio, em segundos, da heurística *Look Ahead*.

Operações sem Restrições						
	Algoritmo					
Tamanho	RSH	WDM	BRPT	BRPR	HPB	BP
10	0.040s	0.040s	0.040s	0.008s	7.362s	0.030s
20	1.601s	1.670s	0.240s	0.176s	14.658s	0.540s
30	14.185s	15.550s	2.176s	1.505s	21.942s	5.600s
40	67.352s	76.551s	10.108s	7.124s	29.206s	30.376s
50	231.282s	271.915s	34.620s	24.189s	36.308s	113.322s
60	639.051s	774.615s	94.572s	66.158s	43.467s	350.027s
70	1507.687s	1894.321s	223.619s	128.840s	50.596s	702.762s
80	3194.246s	4131.422s	472.489s	269.399s	57.690s	1539.99s
90	6249.881s	8284.711s	921.431s	645.769s	64.861s	2085.458s
100	11378.694s	15330.239s	1676.322s	1445.922s	72.029s	3941.291s

Operações de Prefixo			
	Algoritmo		
Tamanho	2-SPR	2-SPRT	4-SPRT
10	7.770s	7.959s	0.003s
20	13.743s	13.977s	0.035s
30	19.586s	19.701s	0.183s
40	25.242s	25.580s	0.607s
50	30.769s	31.327s	1.601s
60	36.173s	37.049s	3.542s
70	41.579s	42.684s	5.265s
80	46.795s	48.126s	9.913s
90	52.014s	53.814s	17.057s
100	57.044s	59.196s	28.307s

Operações de Prefixo ou Sufixo		
	Algoritmo	
Tamanho	2-SPSR	2-SPSRT
10	8.844s	8.805s
20	16.456s	16.414s
30	23.856s	23.921s
40	31.447s	31.316s
50	38.910s	38.330s
60	46.121s	45.563s
70	53.581s	52.262s
80	60.606s	59.353s
90	67.797s	65.822s
100	75.498s	72.773s

Pela Tabela 2.6, podemos ver que a heurística LA consome um tempo de execução maior quando comparada com a heurística SW, mas vale ressaltar que a mesma necessita apenas de um estimador de distância para seu funcionamento. Quando é possível obter um estimador para a distância de ordenação com uma complexidade menor do que um algoritmo que fornece uma solução válida para o problema, a heurística LA pode executar rapidamente, sendo esse o caso do algoritmo HPB. A heurística LA, quando aplicado em algoritmos com baixa complexidade, foi capaz de fornecer uma solução, na média, em poucos segundos (HPB e versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo).

As tabelas 2.7, 2.8 e 2.9 mostram, respectivamente, o fator de aproximação mínimo, médio e máximo da heurística LA em comparação com os algoritmos sem nenhuma heurística aplicada.

Tabela 2.7: Fator de aproximação mínimo da heurística *Look Ahead*.

Operações sem Restrições						
	Algoritmo					
Tamanho	RSH/LA	WDM/LA	BRPT/LA	BRPR/LA	HPB/LA	BP/LA
10	1.200/1.000	1.200/1.000	1.000/1.000	1.000/1.000	1.750/1.000	1.000/1.000
20	1.200/1.100	1.400/1.100	1.100/1.100	1.100/1.100	1.875/1.200	1.111/1.100
30	1.267/1.067	1.533/1.200	1.067/1.067	1.067/1.067	1.923/1.267	1.267/1.133
40	1.300/1.100	1.750/1.200	1.100/1.050	1.100/1.050	1.944/1.300	1.300/1.105
50	1.304/1.120	1.750/1.200	1.120/1.080	1.120/1.080	1.957/1.320	1.280/1.120
60	1.276/1.133	1.800/1.200	1.100/1.033	1.100/1.033	1.964/1.333	1.310/1.200
70	1.353/1.143	1.853/1.286	1.176/1.057	1.086/1.029	1.970/1.314	1.314/1.229
80	1.359/1.150	1.895/1.275	1.125/1.050	1.100/1.025	1.974/1.325	1.325/1.250
90	1.318/1.133	1.889/1.289	1.136/1.044	1.067/1.044	1.977/1.318	1.311/1.267
100	1.306/1.140	1.900/1.300	1.122/1.040	1.080/1.040	1.979/1.340	1.380/1.300

Operações de Prefixo			
	Algoritmo		
Tamanho	2-SPR/LA	2-SPRT/LA	4-SPRT/LA
10	2.000/1.600	1.200/1.200	1.600/1.400
20	2.300/1.800	1.400/1.300	1.900/1.400
30	2.533/2.000	1.533/1.333	2.000/1.733
40	2.600/2.200	1.500/1.350	2.200/1.750
50	2.600/2.160	1.600/1.400	2.400/1.880
60	2.600/2.267	1.567/1.400	2.433/2.000
70	2.600/2.286	1.600/1.400	2.486/2.000
80	2.675/2.325	1.600/1.450	2.525/2.000
90	2.622/2.356	1.600/1.444	2.489/2.133
100	2.680/2.340	1.620/1.480	2.580/2.160

Operações de Prefixo ou Sufixo		
	Algoritmo	
Tamanho	2-SPSR/LA	2-SPSRT/LA
10	1.800/1.200	1.200/1.200
20	2.000/1.400	1.400/1.300
30	2.067/1.600	1.467/1.267
40	2.150/1.700	1.450/1.300
50	2.240/1.720	1.480/1.280
60	2.267/1.733	1.500/1.333
70	2.286/1.743	1.486/1.343
80	2.300/1.800	1.425/1.350
90	2.333/1.756	1.511/1.378
100	2.360/1.800	1.560/1.380

A heurística LA conseguiu reduzir o fator de aproximação mínimo em todos os algoritmos, sem exceção. Note, pela Tabela 2.7, que para a versão sem restrição nas operações, em todos os algoritmos, a heurística LA encontrou pelo menos uma solução ótima para permutações de tamanho 10, nesses casos o valor fornecido pela heurística LA foi igual ao limitante inferior para o problema.

Tabela 2.8: Fator de aproximação médio da heurística *Look Ahead*.

Operações sem Restrições						
Algoritmo						
Tamanho	RSH/LA	WDM/LA	BRPT/LA	BRPR/LA	HPB/LA	BP/LA
10	1.632/1.351	1.844/1.420	1.515/1.301	1.462/1.305	1.914/1.517	1.617/1.369
20	1.618/1.287	1.913/1.423	1.501/1.210	1.405/1.213	1.953/1.508	1.591/1.355
30	1.610/1.257	1.940/1.433	1.486/1.169	1.355/1.164	1.967/1.500	1.571/1.356
40	1.595/1.242	1.955/1.455	1.468/1.142	1.311/1.138	1.975/1.491	1.566/1.375
50	1.594/1.232	1.965/1.476	1.459/1.129	1.282/1.120	1.981/1.480	1.560/1.388
60	1.581/1.226	1.969/1.499	1.437/1.116	1.245/1.106	1.984/1.473	1.551/1.397
70	1.579/1.219	1.975/1.522	1.429/1.107	1.228/1.095	1.986/1.468	1.547/1.414
80	1.567/1.215	1.977/1.542	1.412/1.099	1.212/1.087	1.987/1.464	1.547/1.427
90	1.566/1.212	1.980/1.563	1.409/1.095	1.194/1.081	1.989/1.461	1.541/1.437
100	1.559/1.208	1.982/1.575	1.402/1.091	1.180/1.075	1.990/1.459	1.540/1.450

Operações de Prefixo			
Algoritmo			
Tamanho	2-SPR/LA	2-SPRT/LA	4-SPRT/LA
10	2.944/2.346	1.907/1.658	2.543/1.885
20	2.984/2.398	1.865/1.588	2.720/2.006
30	2.998/2.424	1.851/1.573	2.776/2.097
40	3.010/2.448	1.839/1.565	2.826/2.161
50	3.004/2.468	1.831/1.563	2.854/2.219
60	3.008/2.484	1.825/1.562	2.873/2.260
70	3.010/2.496	1.823/1.562	2.880/2.304
80	3.014/2.513	1.812/1.565	2.891/2.335
90	3.016/2.523	1.812/1.571	2.900/2.362
100	3.011/2.536	1.809/1.573	2.913/2.388

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	2-SPSR/LA	2-SPSRT/LA
10	2.537/1.862	1.833/1.532
20	2.637/1.944	1.780/1.488
30	2.670/1.977	1.766/1.471
40	2.677/1.988	1.761/1.467
50	2.687/1.995	1.758/1.469
60	2.693/2.000	1.753/1.468
70	2.701/2.010	1.749/1.474
80	2.699/2.020	1.747/1.477
90	2.714/2.020	1.741/1.480
100	2.707/2.028	1.747/1.485

Pela Tabela 2.8, podemos perceber a melhoria obtida no fator de aproximação médio de todos os algoritmos ao aplicar a heurística LA em comparação aos resultados fornecidos pelos algoritmos sem nenhuma heurística aplicada.

Tabela 2.9: Fator de aproximação máximo da heurística *Look Ahead*.

Operações sem Restrições						
Algoritmo						
Tamanho	RSH/LA	WDM/LA	BRPT/LA	BRPR/LA	HPB/LA	BP/LA
10	2.000/1.750	2.000/2.000	2.500/1.750	2.000/1.750	2.250/2.000	2.750/2.000
20	2.000/1.556	2.000/1.778	2.111/1.444	1.800/1.500	2.111/1.875	2.125/1.778
30	1.933/1.429	2.000/1.714	1.929/1.357	1.733/1.308	2.000/1.786	1.923/1.714
40	1.947/1.368	2.000/1.700	1.889/1.278	1.550/1.278	2.000/1.700	1.895/1.632
50	1.880/1.375	2.000/1.760	1.875/1.292	1.522/1.217	2.000/1.667	1.913/1.625
60	1.833/1.321	2.000/1.793	1.857/1.241	1.483/1.185	2.000/1.655	1.793/1.621
70	1.829/1.303	2.000/1.788	1.771/1.206	1.424/1.188	2.000/1.636	1.788/1.667
80	1.875/1.300	2.000/1.763	1.725/1.205	1.410/1.162	2.000/1.605	1.744/1.615
90	1.822/1.279	2.000/1.778	1.698/1.182	1.364/1.140	2.000/1.591	1.767/1.651
100	1.800/1.271	2.000/1.776	1.653/1.184	1.306/1.128	2.000/1.592	1.714/1.646

Operações de Prefixo			
Algoritmo			
Tamanho	2-SPR/LA	2-SPRT/LA	4-SPRT/LA
10	4.000/3.000	2.400/2.000	3.400/2.400
20	3.600/2.800	2.200/1.800	3.500/2.400
30	3.733/2.667	2.133/1.800	3.267/2.400
40	3.550/2.700	2.100/1.700	3.450/2.500
50	3.440/2.680	2.040/1.680	3.320/2.480
60	3.400/2.667	2.067/1.667	3.267/2.533
70	3.371/2.686	2.029/1.686	3.314/2.543
80	3.375/2.675	2.000/1.650	3.250/2.525
90	3.378/2.689	2.000/1.667	3.289/2.578
100	3.280/2.680	1.960/1.660	3.180/2.580

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	2-SPSR/LA	2-SPSRT/LA
10	3.600/2.400	2.200/1.800
20	3.500/2.300	2.100/1.700
30	3.267/2.267	2.067/1.667
40	3.250/2.250	2.000/1.600
50	3.240/2.240	2.000/1.600
60	3.200/2.233	1.967/1.600
70	3.286/2.257	1.943/1.571
80	3.200/2.250	1.950/1.575
90	3.156/2.244	1.911/1.556
100	3.180/2.220	1.960/1.580

Pela Tabela 2.9, nota-se que a heurística LA conseguiu reduzir o fator de aproximação máximo de todos algoritmos em quase todos os conjuntos de permutações, com exceção apenas do conjunto de permutações de tamanho 10 e utilizando o algoritmo WDM.

2.5 Look Ahead + Sliding Window

As heurísticas *Look Ahead* e *Sliding Window* apresentaram bons resultados quando aplicadas separadamente. Entretanto, existe a possibilidade de combiná-las para obtenção de resultados ainda melhores. Nesse trabalho a combinação dessas duas heurísticas será referenciada pela sigla LASW.

A heurística *Sliding Window* receberá como algoritmo de parâmetro a heurística *Look Ahead* já previamente inicializada com algum algoritmo. Dessa forma, a sequência inicial

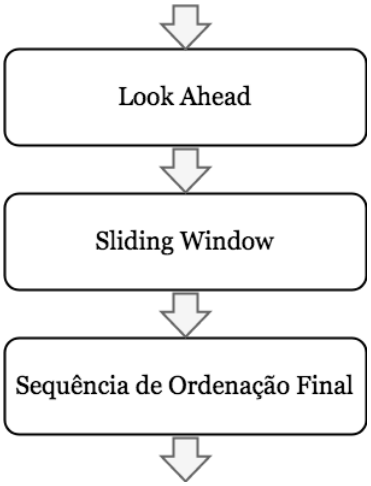

Figura 2.13: Esquema das heurísticas *Look Ahead* e *Sliding Window* sendo combinadas.

de eventos necessária para o *Sliding Window* construir a sequência de permutações será fornecida pelo *Look Ahead*. A Figura 2.13 mostra um esquema de execução quando combinamos essas duas heurísticas.

Combinando essas duas heurísticas podemos obter um resultado ainda melhor em relação à qualidade da solução.

2.5.1 Resultados

A combinação das heurísticas LA e SW foi executada com todos os algoritmos descritos na Tabela 2.1 e com os conjuntos de permutações com identificadores de 10 até 100. Os conjuntos de permutações com identificadores de 150 até 500 não foram utilizados pelo fato do tempo de execução gasto pela heurística LA ser mais elevado. A Figura 2.14 mostra a porcentagem de soluções que foram melhoradas pela combinação das heurísticas LA e SW em comparação com as soluções obtidas pela heurística LA isoladamente.

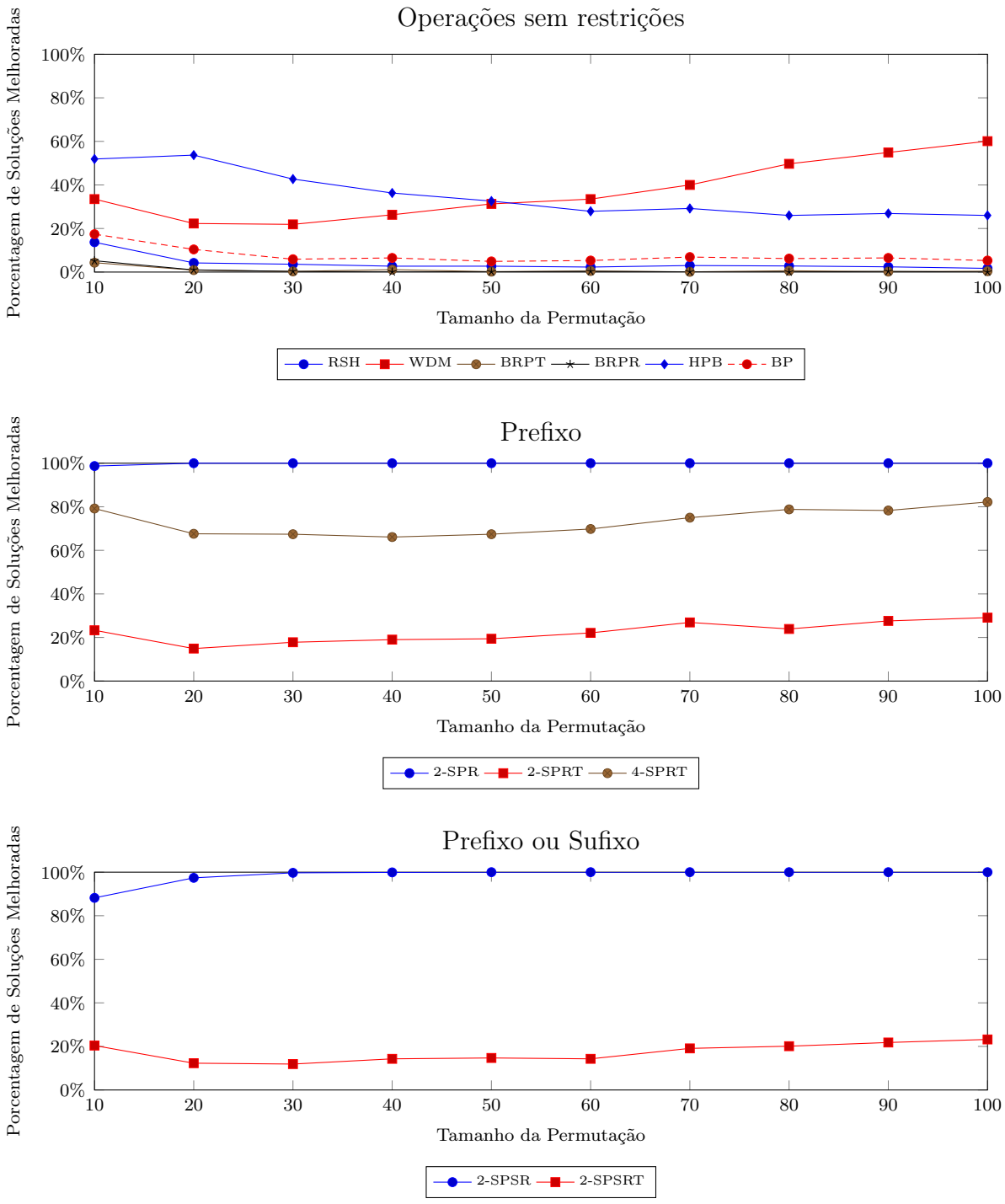

Figura 2.14: Porcentagem de soluções melhoradas utilizando a aplicação conjunta das heurísticas *Look Ahead* e *Sliding Window* em comparação com as soluções da heurística *Look Ahead* isoladamente.

É interessante observar que mesmo com a ótima qualidade das soluções fornecidas pela heurística LA, foi possível melhorar uma quantidade significativa de soluções combinando as heurísticas LA e SW. As tabelas 2.10, 2.11 e 2.12 mostram, respectivamente, o fator de aproximação mínimo, médio e máximo da combinação das heurísticas LA e SW em comparação com a heurística LA.

Tabela 2.10: Fator de aproximação mínimo das heurísticas *Look Ahead* e *Sliding Window* combinadas comparada com o *Look Ahead* isoladamente.

Operações sem Restrições						
Algoritmo						
Tamanho	LA(RSH)/LASW	LA(WDM)/LASW	LA(BRPT)/LASW	LA(BRPR)/LASW	LA(HPB)/LASW	LA(BP)/LASW
10	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
20	1.100/1.100	1.100/1.100	1.100/1.100	1.100/1.100	1.200/1.100	1.100/1.100
30	1.067/1.067	1.200/1.200	1.067/1.067	1.067/1.067	1.267/1.133	1.133/1.133
40	1.100/1.100	1.200/1.200	1.050/1.050	1.050/1.050	1.300/1.250	1.105/1.105
50	1.120/1.120	1.200/1.200	1.080/1.080	1.080/1.040	1.320/1.250	1.120/1.120
60	1.133/1.133	1.200/1.200	1.033/1.033	1.033/1.033	1.333/1.310	1.200/1.200
70	1.143/1.143	1.286/1.286	1.057/1.057	1.029/1.029	1.314/1.286	1.229/1.229
80	1.150/1.150	1.275/1.275	1.050/1.050	1.025/1.025	1.325/1.300	1.250/1.250
90	1.133/1.133	1.289/1.289	1.044/1.044	1.044/1.044	1.318/1.311	1.267/1.267
100	1.140/1.140	1.300/1.300	1.040/1.040	1.040/1.040	1.340/1.340	1.300/1.300

Operações de Prefixo			
Algoritmo			
Tamanho	LA(2-SPR)/LASW	LA(2-SPRT)/LASW	LA(4-SPRT)/SWLA
10	1.600/1.400	1.200/1.200	1.400/1.200
20	1.800/1.500	1.300/1.300	1.400/1.300
30	2.000/1.600	1.333/1.333	1.733/1.667
40	2.200/1.650	1.350/1.350	1.750/1.700
50	2.160/1.640	1.400/1.400	1.880/1.840
60	2.267/1.733	1.400/1.400	2.000/1.933
70	2.286/1.714	1.400/1.400	2.000/1.971
80	2.325/1.775	1.450/1.450	2.000/1.975
90	2.356/1.800	1.444/1.444	2.133/2.111
100	2.340/1.820	1.480/1.480	2.160/2.120

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	LA(2-SPSR)/LASW	LA(2-SPSRT)/LASW
10	1.200/1.200	1.200/1.000
20	1.400/1.300	1.300/1.200
30	1.600/1.467	1.267/1.267
40	1.700/1.400	1.300/1.300
50	1.720/1.520	1.280/1.280
60	1.733/1.567	1.333/1.333
70	1.743/1.543	1.343/1.343
80	1.800/1.575	1.350/1.350
90	1.756/1.600	1.378/1.356
100	1.800/1.600	1.380/1.380

Pela Tabela 2.10, podemos perceber que houve pouca mudança no fator de aproximação mínimo ao aplicar as heurísticas LA e SW em conjunto para a versão sem restrições nas operações. Percebemos mudanças mais relevantes nas versões restrita a operações de prefixo e restrita a operações de prefixo ou sufixo.

Tabela 2.11: Fator de aproximação médio das heurísticas *Look Ahead* e *Sliding Window* combinadas comparada com o *Look Ahead* isoladamente.

Operações sem Restrições						
Algoritmo						
Tamanho	LA(RSH)/LASW	LA(WDM)/LASW	LA(BRPT)/LASW	LA(BRPR)/LASW	LA(HPB)/LASW	LA(BP)/LASW
10	1.351/1.322	1.420/1.348	1.301/1.292	1.305/1.295	1.517/1.403	1.369/1.332
20	1.287/1.283	1.423/1.398	1.210/1.209	1.213/1.212	1.508/1.443	1.355/1.343
30	1.257/1.254	1.433/1.416	1.169/1.169	1.164/1.164	1.500/1.466	1.356/1.352
40	1.242/1.240	1.455/1.440	1.142/1.142	1.138/1.138	1.491/1.470	1.375/1.372
50	1.232/1.231	1.476/1.460	1.129/1.128	1.120/1.120	1.480/1.464	1.388/1.386
60	1.226/1.226	1.499/1.485	1.116/1.116	1.106/1.105	1.473/1.463	1.397/1.395
70	1.219/1.218	1.522/1.506	1.107/1.107	1.095/1.095	1.468/1.458	1.414/1.412
80	1.215/1.215	1.542/1.524	1.099/1.099	1.087/1.087	1.464/1.457	1.427/1.425
90	1.212/1.211	1.563/1.545	1.095/1.095	1.081/1.080	1.461/1.454	1.437/1.436
100	1.208/1.208	1.575/1.555	1.091/1.091	1.075/1.075	1.459/1.453	1.450/1.449

Operações de Prefixo			
Algoritmo			
Tamanho	LA(2-SPR)/LASW	LA(2-SPRT)/LASW	LA(4-SPRT)/LASW
10	2.346/1.862	1.658/1.611	1.885/1.631
20	2.398/1.911	1.588/1.573	2.006/1.899
30	2.424/1.938	1.573/1.561	2.097/2.027
40	2.448/1.952	1.565/1.555	2.161/2.108
50	2.468/1.964	1.563/1.555	2.219/2.174
60	2.484/1.971	1.562/1.554	2.260/2.219
70	2.496/1.972	1.562/1.554	2.304/2.264
80	2.513/1.979	1.565/1.558	2.335/2.297
90	2.523/1.985	1.571/1.564	2.362/2.327
100	2.536/1.986	1.573/1.566	2.388/2.352

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	LA(2-SPSR)/LASW	LA(2-SPSRT)/LASW
10	1.862/1.594	1.532/1.491
20	1.944/1.682	1.488/1.476
30	1.977/1.720	1.471/1.463
40	1.988/1.737	1.467/1.460
50	1.995/1.746	1.469/1.463
60	2.000/1.752	1.468/1.463
70	2.010/1.756	1.474/1.468
80	2.020/1.766	1.477/1.471
90	2.020/1.766	1.480/1.474
100	2.028/1.771	1.485/1.480

Pela Tabela 2.11, nota-se que houve pequenas melhorias no fator de aproximação médio de todos os algoritmos. Em alguns casos, essas melhorias são mais perceptíveis, como por exemplo nos algoritmos 2-SPR e 2-SPSR.

Tabela 2.12: Fator de aproximação máximo das heurísticas *Look Ahead* e *Sliding Window* combinadas comparada com o *Look Ahead* isoladamente.

Operações sem Restrições						
Algoritmo						
Tamanho	LA(RSH)/LASW	LA(WDM)/LASW	LA(BRPT)/LASW	LA(BRPR)/LASW	LA(HPB)/LASW	LA(BP)/LASW
10	1.750/1.750	2.000/2.000	1.750/1.750	1.750/1.750	2.000/2.000	2.000/2.000
20	1.556/1.556	1.778/1.778	1.444/1.444	1.500/1.500	1.875/1.875	1.778/1.778
30	1.429/1.429	1.714/1.714	1.357/1.357	1.308/1.308	1.786/1.786	1.714/1.714
40	1.368/1.368	1.700/1.700	1.278/1.278	1.278/1.278	1.700/1.684	1.632/1.632
50	1.375/1.375	1.760/1.750	1.292/1.292	1.217/1.217	1.667/1.667	1.625/1.625
60	1.321/1.321	1.793/1.759	1.241/1.241	1.185/1.185	1.655/1.655	1.621/1.621
70	1.303/1.303	1.788/1.727	1.206/1.206	1.188/1.188	1.636/1.636	1.667/1.667
80	1.300/1.300	1.763/1.737	1.205/1.205	1.162/1.162	1.605/1.605	1.615/1.615
90	1.279/1.279	1.778/1.756	1.182/1.182	1.140/1.140	1.591/1.591	1.651/1.651
100	1.271/1.271	1.776/1.750	1.184/1.184	1.128/1.128	1.592/1.583	1.646/1.646

Operações de Prefixo			
Algoritmo			
Tamanho	LA(2-SPR)/LASW	LA(2-SPRT)/LASW	LA(4-SPRT)/LASW
10	3.000/2.400	2.000/2.000	2.400/2.200
20	2.800/2.400	1.800/1.800	2.400/2.300
30	2.667/2.333	1.800/1.800	2.400/2.400
40	2.700/2.250	1.700/1.700	2.500/2.400
50	2.680/2.240	1.680/1.680	2.480/2.480
60	2.667/2.200	1.667/1.667	2.533/2.467
70	2.686/2.171	1.686/1.657	2.543/2.543
80	2.675/2.200	1.650/1.650	2.525/2.525
90	2.689/2.222	1.667/1.667	2.578/2.533
100	2.680/2.220	1.660/1.660	2.580/2.560

Operações de Prefixo ou Sufixo		
Algoritmo		
Tamanho	LA(2-SPSR)/LASW	LA(2-SPSRT)/LASW
10	2.400/2.000	1.800/1.800
20	2.300/2.100	1.700/1.700
30	2.267/2.133	1.667/1.667
40	2.250/2.000	1.600/1.600
50	2.240/2.000	1.600/1.600
60	2.233/2.000	1.600/1.567
70	2.257/1.971	1.571/1.571
80	2.250/2.000	1.575/1.575
90	2.244/2.000	1.556/1.556
100	2.220/1.960	1.580/1.580

Pela Tabela 2.12, podemos perceber que não foi possível reduzir o fator de aproximação máximo da heurística LA em grande parte dos casos, com exceção dos algoritmos 2-SPR e 2-SPSR.

2.6 Iterative Sliding Window

A heurística *Iterative Sliding Window* (ISW) é de certa forma semelhante a heurística *Sliding Window*. Um dos pontos que as diferencia é que nessa nova heurística removemos a dependência de uma base de dados de sequências de ordenação e a limitação de um tamanho máximo para a janela. Para isso, abrimos mão da otimalidade que a base de dados proporcionava e a substituímos pela heurística *Look Ahead*, que apresentou excelentes

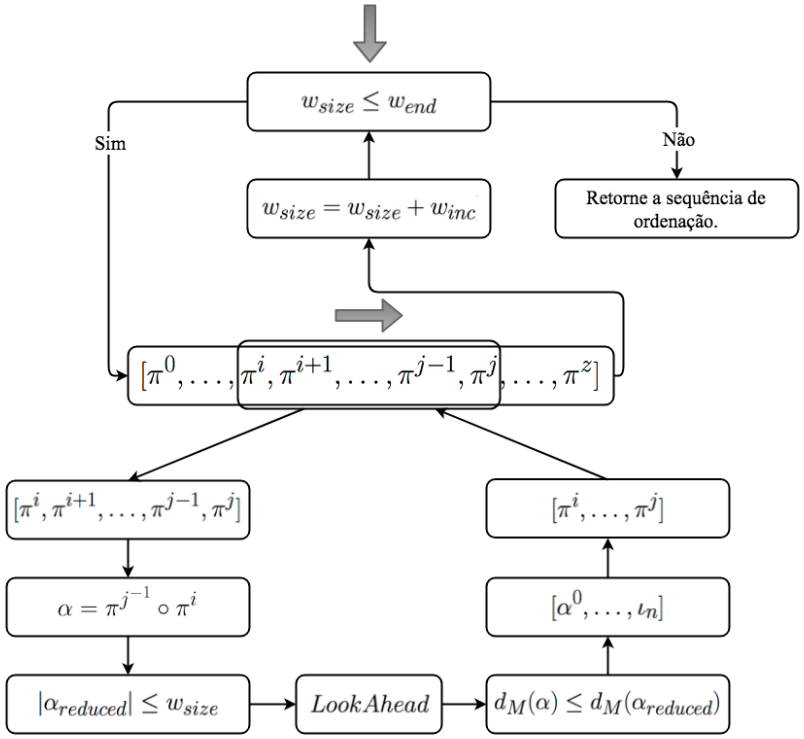

Figura 2.15: Esquema da heurística *Iterative Sliding Window*.

resultados. Outro ponto importante é que a heurística é executada iterativamente com um tamanho de janela reajustado a partir da execução anterior.

A heurística recebe como parâmetros um algoritmo alg, um valor inicial para o tamanho da janela w_{size}, um valor incremental para o tamanho da janela w_{inc} e um valor máximo para o tamanho da janela w_{end}. Semelhante ao *Sliding Window*, a heurística utiliza o algoritmo alg para obter uma sequência inicial de eventos para construir uma sequência de permutações. A janela é deslizada pela sequência de permutações diferenciando da heurística SW apenas no momento de obter uma ordenação para a permutação $\alpha_{reduced}$ onde a heurística *Look Ahead* é utilizada, em vez de consultar a base de dados. Após a janela ser deslizada por toda a sequência de permutações, o valor de w_{size} é incrementado ($w_{size} = w_{size} + w_{inc}$) e o processo é repetido. A heurística finaliza sua execução quando o valor de w_{size} for maior que w_{end}. A Figura 2.15 mostra um esquema de execução dessa heurística.

Os algoritmos 3 e 4 mostram o pseudocódigo da heurística *Iterative Sliding Window*.

Algoritmo 3: ModifiedSlidingWindow(S,w_{size},alg)

Entrada:
S: Sequência de permutações;
w_{size}: Valor máximo para o tamanho da janela;
alg: Algoritmo utilizado pelo *Look Ahead*;
Saída:
Sequência de permutações;

```
1  início
2      i ← 0;
3      j ← 2;
4      enquanto j ≤ Tamanho(S) faça
5          S^w ← ObtenhaSubsequencia(S, i, j);
6          α ← ComputeAlpha(S^w);
7          α_reduced ← ReduzaAlpha(α);
8          se Tamanho(α_reduced) ≤ w_size então
9              S^w' ← LookAhead(alg,α_reduced);
10             se Tamanho(S^w') < Tamanho(S^w) então
11                 S^w' ← AtualizePermutacoes(S^w',π^j);
12                 S^w ← S^w';
13                 i ← i + 1;
14                 j ← i + Tamanho(S^w');
15             senão
16                 j ← j + 1;
17         senão
18             i ← i + 1;
19     retorna S;
```

Algoritmo 4: IterativeSlidingWindow(alg,π)

Entrada:
alg: Algoritmo utilizado para construir a sequência de permutações e pelo *Look Ahead*;
w_{size}: Valor inicial para o tamanho máximo da janela;
w_{inc}: Valor incremental para o tamanho da janela;
w_{end}: Valor final para o tamanho máximo da janela;
π: Permutação para ser ordenada;
Saída:
Sequência de ordenação de π;

```
1  início
2      S ← ConstruaSequenciaPermutacoes(alg, π);
3      enquanto w_size ≤ w_end faça
4          S ← ModifiedSlidingWindow(S, w_size, alg);
5          w_size = w_size + w_inc
6      retorna SequenciaOrdenacao(S);
```

2.6.1 Determinando os Parâmetros da Heurística

Nesse trabalho, a heurística ISW é a única que apresenta parâmetros que permitem efetuar ajustes em seu comportamento. Determinar o tamanho máximo da janela e o quanto será incrementado a cada iteração possibilita que a heurística se adeque melhor ao problema e ao algoritmo utilizado. Para encontrar os melhores parâmetros para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições, aplicamos uma série de testes. Por esse motivo, os algoritmos utilizados por essa heurística foram específicos para os problemas sem restrições nas operações (RSH, WDM, BRPT e BRPR), restrito a operações de prefixo (2-SPRT) e restrito a operações de prefixo ou sufixo (2-SPSRT).

Utilizamos um conjunto com 1000 permutações de tamanho 100 para realização dos testes. Primeiramente queríamos determinar um tamanho mínimo (w_{size}) e máximo (w_{end}) para o tamanho da janela onde houvesse um bom custo-benefício entre o tempo de execução e a qualidade da solução.

Iniciamos os testes com os parâmetros $w_{size} = 5$, $w_{inc} = 5$ e $w_{end} = 30$, onde percebemos que aumentar o tamanho máximo para a janela tornava a heurística mais demorada e a melhoria da solução não era tão expressiva. A Tabela 2.13 mostra o fator de aproximação médio enquanto a Tabela 2.14 mostra o tempo de execução médio, em segundos, para o teste realizado.

Tabela 2.13: Aproximação média obtida pela heurística ISW com os parâmetros w_{size}, w_{inc} e w_{end}.

$w_{size} = 5, w_{inc} = 5$						
Algoritmos	$w_{end} = 5$	$w_{end} = 10$	$w_{end} = 15$	$w_{end} = 20$	$w_{end} = 25$	$w_{end} = 30$
RSH	1.432	1.364	1.325	1.299	1.279	1.266
WDM	1.528	1.418	1.365	1.338	1.323	1.314
BRPT	1.387	1.363	1.339	1.316	1.295	1.275
BRPR	1.167	1.147	1.132	1.121	1.113	1.106
2-SPRT	1.783	1.723	1.647	1.611	1.597	1.574
2-SPSRT	1.724	1.663	1.590	1.542	1.519	1.493

Tabela 2.14: Tempo de execução médio da heurística ISW, em segundos, parâmetros $w_{size} = 5$, $w_{inc} = 5$ e $w_{end} = 30$.

$w_{size} = 5, w_{inc} = 5$						
Algoritmos	$w_{end} = 5$	$w_{end} = 10$	$w_{end} = 15$	$w_{end} = 20$	$w_{end} = 25$	$w_{end} = 30$
RSH	0.032s	1.207s	9.869s	47.268s	166.949s	478.788s
WDM	0.064s	1.381s	11.222s	54.445s	194.752s	569.155s
BRPT	0.010s	0.163s	2.9400s	6.123s	25.095s	77.001s
BRPR	0.032s	0.189s	1.323s	6.477s	16.828s	69.298s
2-SPRT	0.038s	0.954s	1.202s	2.514s	5.113s	14.491s
2-SPSRT	0.046s	1.023s	2.433s	4.714s	12.435s	46.338s

Tabela 2.15: Aproximação média da heurística ISW com decremento do tamanho da janela.

$w_{size} = 30, w_{inc} = -5$						
Algoritmos	$w_{end} = 30$	$w_{end} = 25$	$w_{end} = 20$	$w_{end} = 15$	$w_{end} = 10$	$w_{end} = 5$
RSH	1.326	1.316	1.315	1.314	1.314	1.313
WDM	1.488	1.455	1.446	1.442	1.441	1.441
BRPT	1.303	1.296	1.295	1.295	1.295	1.295
BRPR	1.122	1.120	1.119	1.119	1.119	1.119
2-SPRT	1.780	1.753	1.723	1.661	1.632	1.604
2-SPSRT	1.726	1.682	1.658	1.632	1.595	1.564

O resultado dos primeiros testes foram promissores, mas ainda queríamos saber se incrementar ou decrementar a cada iteração o tamanho da janela produziria resultados melhores. A Tabela 2.15 mostra o fator de aproximação médio dos testes decrementando o tamanho máximo da janela.

Pelas tabelas 2.13 e 2.15, podemos observar que houve uma mudança considerável na qualidade das soluções quando incrementamos ou decrementados o valor do tamanho da janela. Para os algoritmos utilizados percebemos que as soluções incrementando tal valor apresentaram melhores resultados.

Por fim, obtivemos um valor adequado para o parâmetro w_{inc}. Para isso, realizamos um novo teste com os parâmetros $w_{size} = 5$, $w_{inc} = 1$ e $w_{end} = 30$. O objetivo desse teste foi verificar se o fator de aproximação médio sofreria uma grande mudança adotando tais valores. A Tabela 2.16 mostra um comparativo utilizando o valor de w_{inc} como 1 e 5. Pelos resultados apresentados na Tabela 2.16, podemos constatar que a variação no fator de aproximação médio foi pequena e que adotar $w_{inc} = 5$ é aceitável.

Tabela 2.16: Aproximação média da heurística ISW com valores distintos para o parâmetro w_{inc}.

$w_{size} = 5, w_{inc} = 1$						
Algoritmos	$w_{end} = 5$	$w_{end} = 10$	$w_{end} = 15$	$w_{end} = 20$	$w_{end} = 25$	$w_{end} = 30$
RSH	1.432	1.362	1.320	1.292	1.272	1.257
WDM	1.528	1.406	1.346	1.315	1.299	1.289
BRPT	1.387	1.361	1.336	1.311	1.289	1.269
BRPR	1.167	1.146	1.130	1.119	1.111	1.104
2-SPRT	1.783	1.709	1.629	1.602	1.589	1.567
2-SPSRT	1.724	1.650	1.581	1.534	1.510	1.487

$w_{size} = 5, w_{inc} = 5$						
Algoritmos	$w_{end} = 5$	$w_{end} = 10$	$w_{end} = 15$	$w_{end} = 20$	$w_{end} = 25$	$w_{end} = 30$
RSH	1.432	1.364	1.325	1.299	1.279	1.266
WDM	1.528	1.418	1.365	1.338	1.323	1.314
BRPT	1.387	1.363	1.339	1.316	1.295	1.275
BRPR	1.167	1.147	1.132	1.121	1.113	1.106
2-SPRT	1.783	1.723	1.647	1.611	1.597	1.574
2-SPSRT	1.724	1.663	1.590	1.542	1.519	1.493

2.6.2 Resultados

A heurística ISW foi aplicada em todos os algoritmos referentes ao problema de Ordenação de Permutações com Sinais por Reversões e Transposições, tanto na forma clássica (RSH, WDM, BRPT e BRPR) quanto restrita a operações de prefixo (2-SPRT) e restrita a operações de prefixo ou sufixo (2-SPSRT). A heurística foi executada com todos os conjuntos de permutações. As figuras 2.16, 2.17, 2.18 e 2.19 mostram o fator de aproximação médio dos algoritmos em comparação com o obtido pela heurística ISW.

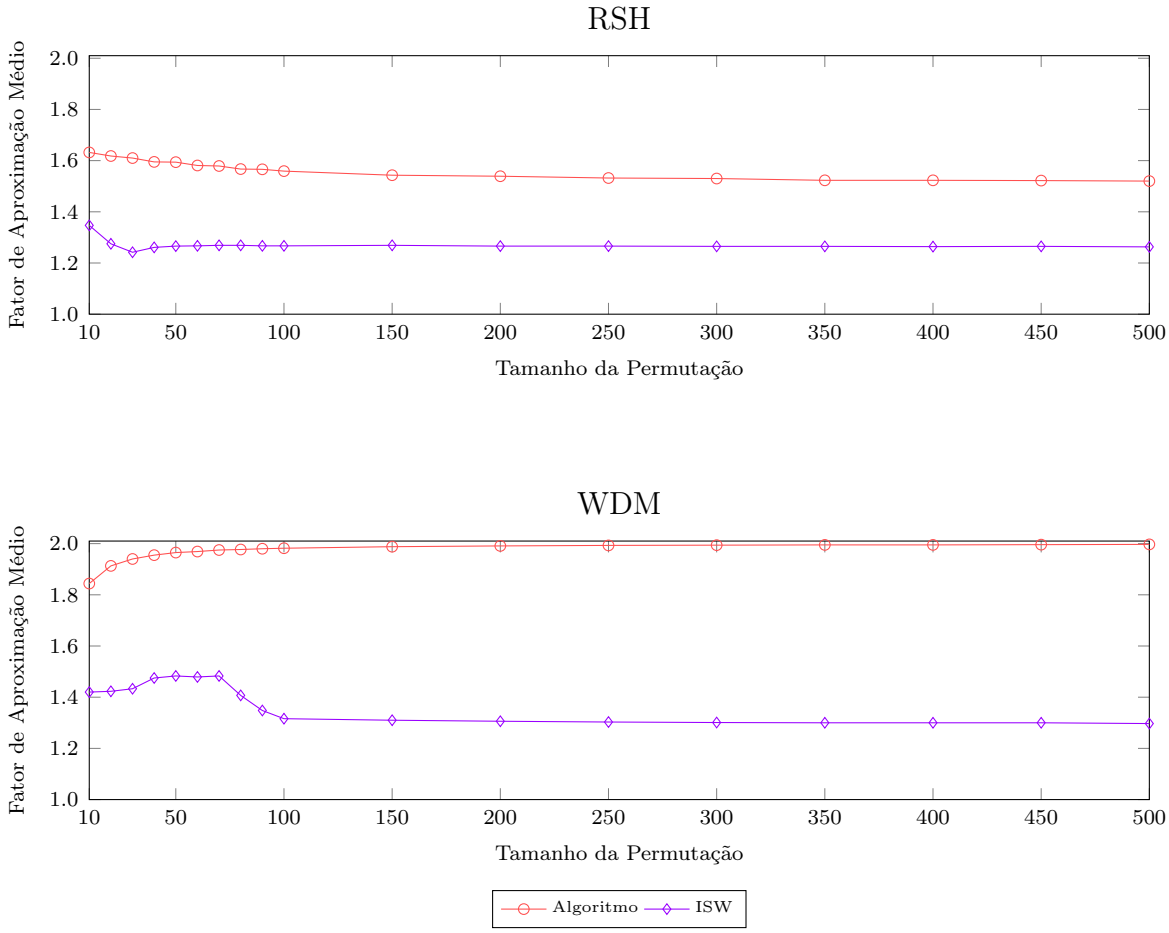

Figura 2.16: Fator de aproximação médio da heurística *Iterative Sliding Window* aplicada nos algoritmos RSH e WDM.

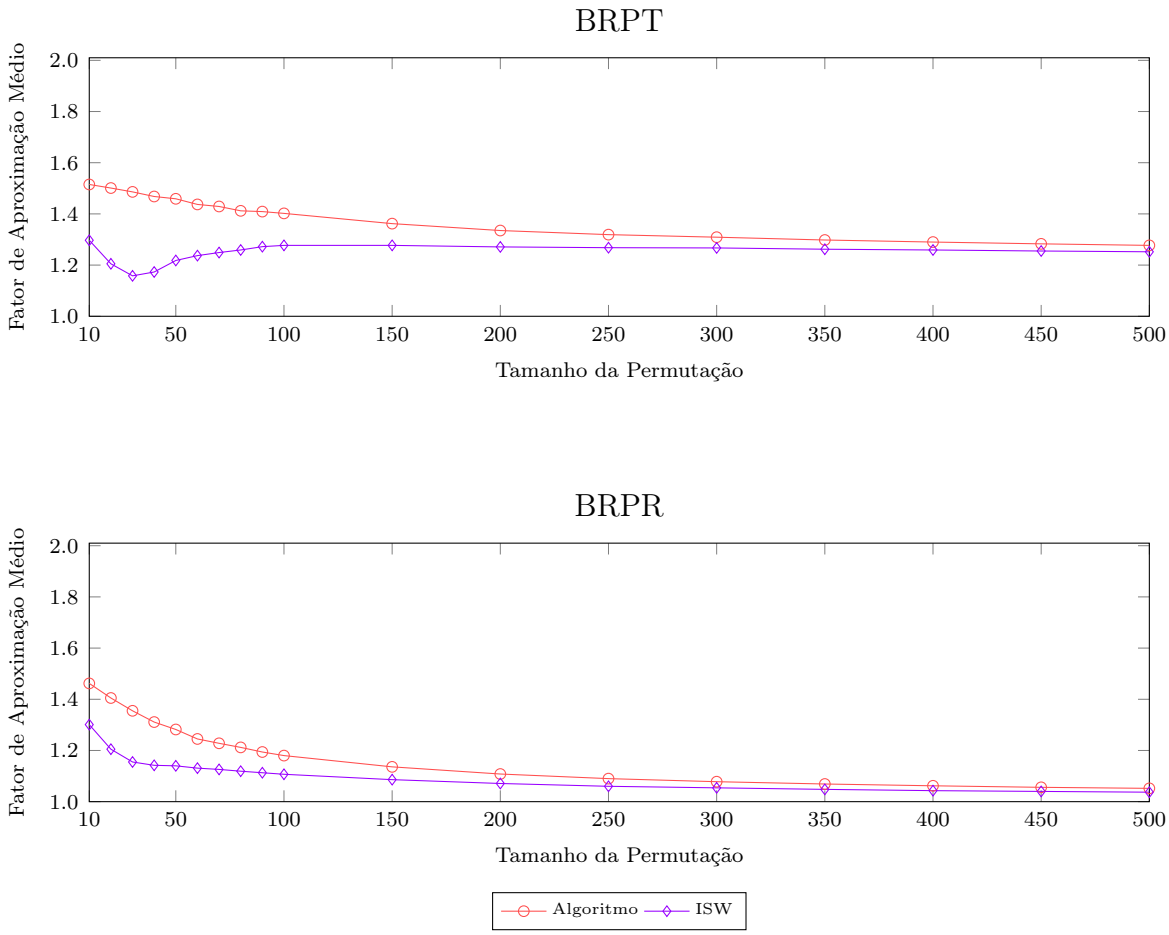

Figura 2.17: Fator de aproximação médio da heurística *Iterative Sliding Window* aplicada nos algoritmos BRPT e BRPR.

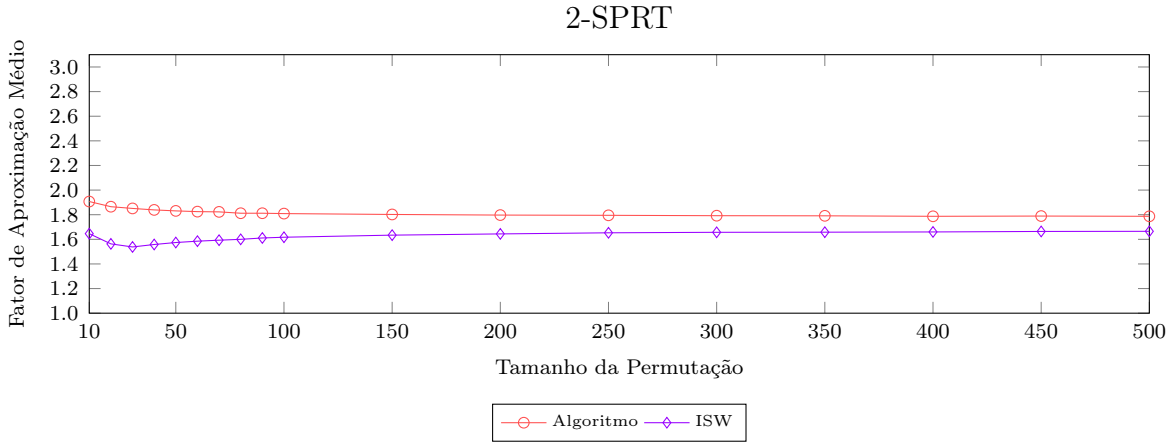

Figura 2.18: Fator de aproximação médio da heurística *Iterative Sliding Window* aplicado no algoritmo 2-SPRT.

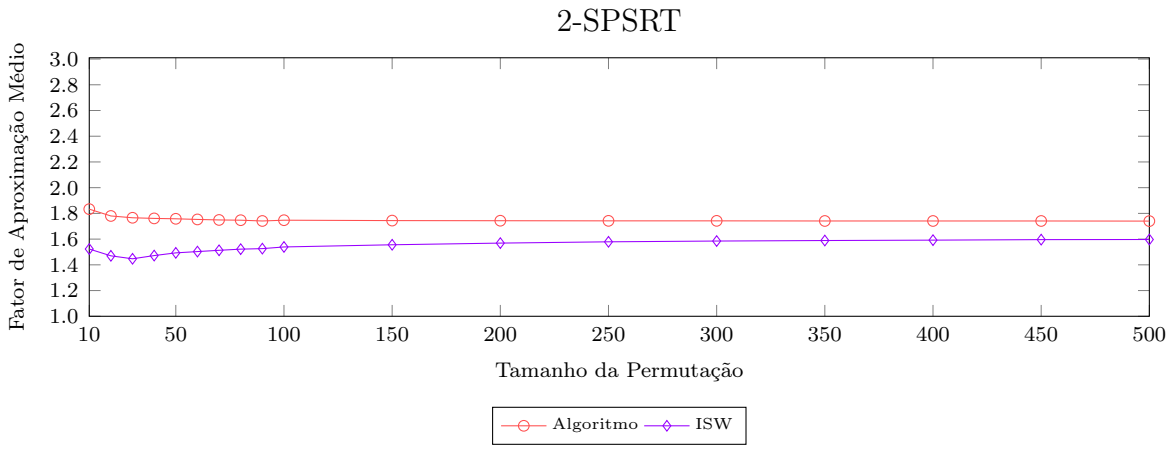

Figura 2.19: Fator de aproximação médio da heurística *Iterative Sliding Window* aplicado no algoritmo 2-SPSRT.

Ressaltamos o bom desempenho obtido pela heurística ISW em relação aos algoritmos sem nenhuma heurística aplicada. O fator de aproximação médio dos algoritmos foi reduzido de maneira considerável ao aplicar a heurística, inclusive nos algoritmos BRPT e BRPR, onde os resultados dos algoritmos foram naturalmente muito bons. A Tabela 2.17 mostra o tempo médio, em segundos, gasto em cada permutação pela heurística ISW.

Pela Tabela 2.17, podemos perceber que a heurística ISW para permutações de tamanho 500 fornece uma solução, na média, em poucos minutos. Para os algoritmos RSH e WDM, que apresentam uma complexidade um pouco mais elevada, o tempo de execução foi um pouco maior. A Figura 2.20 mostra a porcentagem de soluções que foram melhoradas utilizando a heurística ISW.

Tabela 2.17: Tempo de execução médio, em segundos, gasto em cada permutação pela heurística *Iterative Sliding Window*.

	Operações sem Restrições				Prefixo	Prefixo e Sufixo
Tamanho	RSH	WDM	BRPT	BRPR	2-SPRT	2-SPSRT
50	174.683s	244.000s	41.661s	35.677s	5.391s	17.135s
100	478.788s	569.155s	78.180s	69.298s	14.491s	46.338s
150	779.858s	941.730s	102.712s	92.188s	23.249s	74.301s
200	1077.222s	1319.121s	124.501s	111.657s	31.889s	102.356s
250	1370.217s	1685.888s	143.024s	128.396s	40.271s	130.166s
300	1668.735s	2072.016s	161.663s	145.635s	48.815s	157.683s
350	1963.604s	2445.818s	180.396s	162.340s	57.396s	185.459s
400	2264.583s	2830.796s	198.332s	178.331s	65.765s	213.411s
450	2563.567s	3203.581s	215.413s	193.762s	74.429s	240.675s
500	2869.587s	3584.094s	231.904s	209.160s	82.639s	267.607s

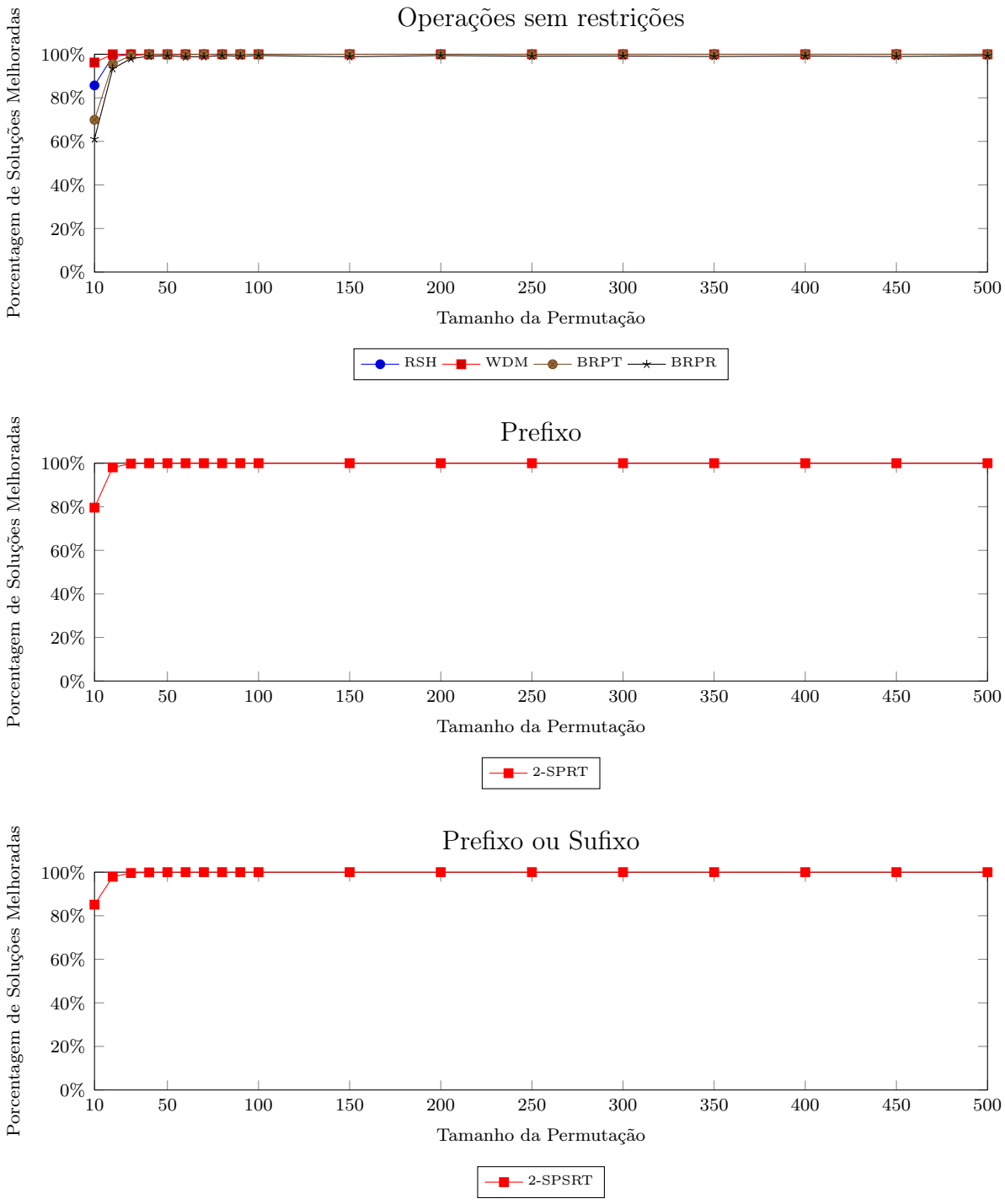

Figura 2.20: Porcentagem de soluções melhoradas utilizando a heurística *Iterative Sliding Window*.

Pela Figura 2.20, nota-se que a heurística ISW foi capaz de melhorar as soluções iniciais dos algoritmos em mais de 90% dos casos para permutações com tamanho maior que 10. A Tabela 2.18 mostra os fatores de aproximação mínimo, médio e máximo da heurística ISW em comparação com os algoritmos sem nenhuma heurística aplicada.

Pela Tabela 2.18 nota-se que a heurística ISW reduziu os fatores de aproximação mínimo, médio e máximo para todos os algoritmos, sem exceções.

Tabela 2.18: Fator de aproximação mínimo, médio e máximo da heurística *Iterative Sliding Window*.

Fator de Aproximação Mínimo						
	Operações sem Restrições				Prefixo	Prefixo e Sufixo
Tamanho	RSH/ISW	WDM/ISW	BRPT/ISW	BRPR/ISW	2-SPRT/ISW	2-SPSRT/ISW
50	1.304/1.120	1.750/1.200	1.120/1.040	1.120/1.040	1.600/1.360	1.480/1.320
100	1.306/1.160	1.900/1.204	1.122/1.061	1.080/1.040	1.620/1.460	1.560/1.400
150	1.284/1.187	1.933/1.187	1.123/1.054	1.067/1.040	1.653/1.520	1.587/1.440
200	1.313/1.200	1.950/1.200	1.120/1.061	1.060/1.040	1.670/1.530	1.610/1.460
250	1.303/1.200	1.960/1.224	1.122/1.081	1.048/1.032	1.680/1.528	1.616/1.464
300	1.311/1.213	1.973/1.240	1.134/1.094	1.040/1.027	1.700/1.560	1.627/1.487
350	1.275/1.217	1.977/1.246	1.086/1.075	1.040/1.029	1.686/1.549	1.623/1.486
400	1.330/1.220	1.975/1.240	1.131/1.101	1.035/1.025	1.705/1.555	1.645/1.505
450	1.300/1.218	1.978/1.241	1.143/1.121	1.031/1.022	1.707/1.587	1.618/1.516
500	1.294/1.208	1.984/1.244	1.081/1.065	1.028/1.024	1.712/1.584	1.648/1.508

Fator de Aproximação Médio						
	Operações sem Restrições				Prefixo	Prefixo e Sufixo
Tamanho	RSH/ISW	WDM/ISW	BRPT/ISW	BRPR/ISW	2-SPRT/ISW	2-SPSRT/ISW
50	1.594/1.266	1.965/1.483	1.459/1.218	1.282/1.140	1.831/1.574	1.758/1.493
100	1.559/1.267	1.982/1.316	1.402/1.277	1.180/1.107	1.809/1.617	1.747/1.539
150	1.543/1.269	1.988/1.310	1.362/1.277	1.136/1.086	1.802/1.634	1.744/1.556
200	1.539/1.266	1.991/1.306	1.335/1.271	1.108/1.071	1.797/1.644	1.743/1.569
250	1.532/1.266	1.993/1.303	1.319/1.268	1.090/1.060	1.795/1.653	1.742/1.579
300	1.530/1.265	1.994/1.301	1.309/1.267	1.078/1.054	1.792/1.657	1.742/1.585
350	1.523/1.265	1.995/1.300	1.298/1.262	1.069/1.048	1.791/1.658	1.741/1.589
400	1.523/1.264	1.995/1.300	1.290/1.259	1.062/1.043	1.787/1.660	1.741/1.592
450	1.522/1.265	1.996/1.300	1.283/1.255	1.056/1.040	1.789/1.664	1.741/1.596
500	1.520/1.263	1.997/1.297	1.277/1.252	1.052/1.037	1.787/1.665	1.740/1.597

Fator de Aproximação Máximo						
	Operações sem Restrições				Prefixo	Prefixo e Sufixo
Tamanho	RSH/ISW	WDM/ISW	BRPT/ISW	BRPR/ISW	2-SPRT/ISW	2-SPSRT/ISW
50	1.880/1.435	2.000/1.792	1.875/1.500	1.522/1.261	2.040/1.760	2.000/1.680
100	1.800/1.375	2.000/1.500	1.653/1.531	1.306/1.188	1.960/1.800	1.960/1.700
150	1.733/1.351	2.000/1.425	1.608/1.500	1.222/1.137	1.933/1.813	1.933/1.707
200	1.740/1.343	2.000/1.408	1.535/1.485	1.192/1.122	1.940/1.750	1.890/1.690
250	1.744/1.339	2.000/1.382	1.492/1.411	1.163/1.106	1.904/1.768	1.872/1.688
300	1.725/1.338	2.000/1.385	1.487/1.433	1.134/1.094	1.887/1.747	1.853/1.687
350	1.703/1.322	2.000/1.395	1.460/1.397	1.110/1.075	1.880/1.731	1.851/1.680
400	1.685/1.318	2.000/1.365	1.437/1.386	1.102/1.066	1.865/1.750	1.840/1.695
450	1.700/1.309	2.000/1.372	1.411/1.379	1.103/1.058	1.862/1.738	1.849/1.684
500	1.692/1.325	2.000/1.355	1.378/1.347	1.085/1.060	1.856/1.736	1.824/1.692

2.7 Comparando as Heurísticas

Nessa seção iremos realizar um comparativo em relação a qualidade das soluções fornecidas por nossas heurísticas. Para isso, iremos considerar o fator de aproximação médio fornecido por cada uma delas. As figuras 2.21 e 2.22 mostram um comparativo entre o fator de aproximação médio das heurísticas para a versão do problema sem restrição nas operações. As figuras 2.23 e 2.24 apresentam um comparativo entre o fator de aproximação médio das heurísticas para as versões do problema restrita a operações de prefixo e restrita a operações de prefixo ou sufixo, respectivamente.

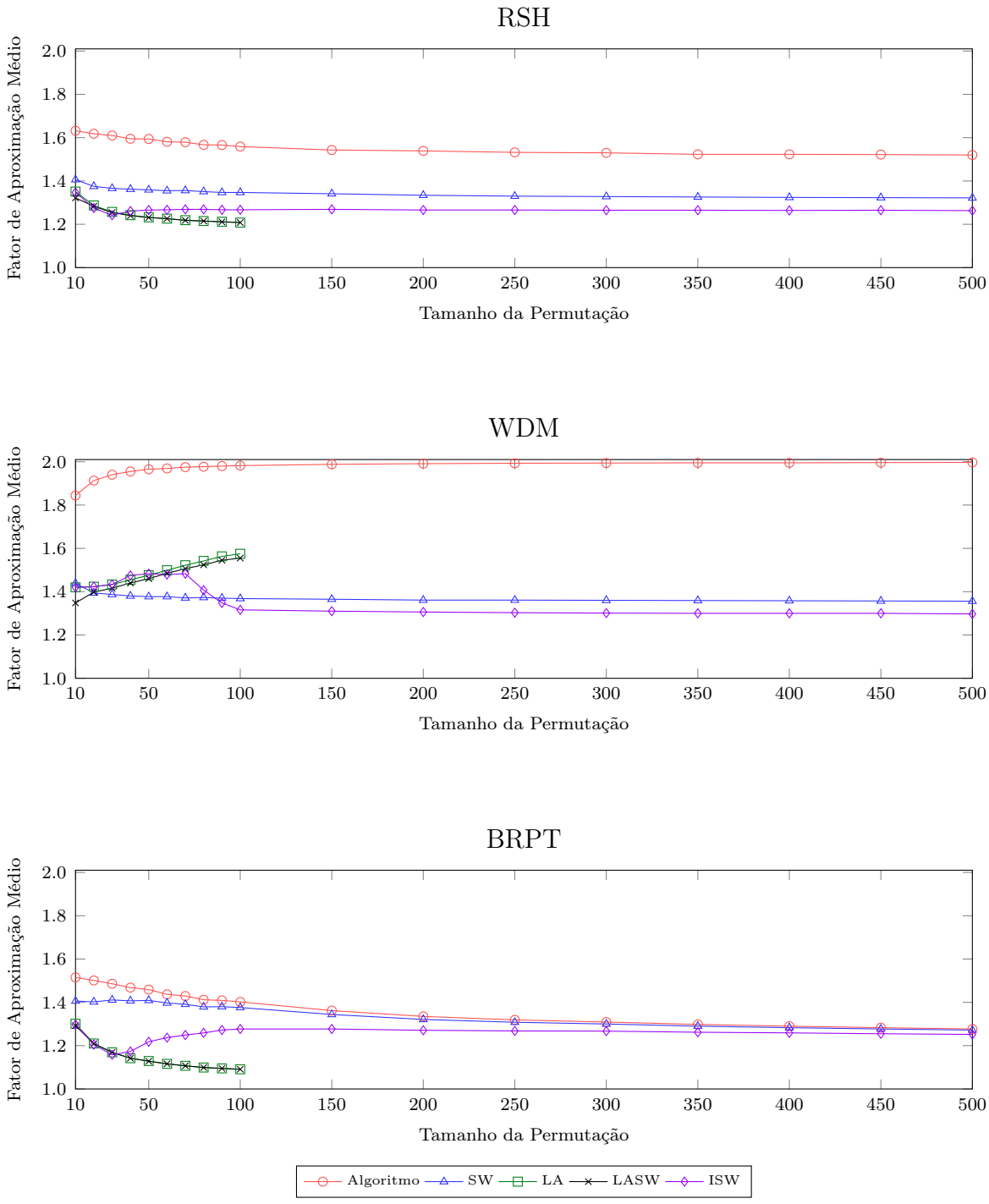

Figura 2.21: Comparativo entre o fator de aproximação médio das heurísticas aplicadas nos algoritmos RSH, WDM e BRPT.

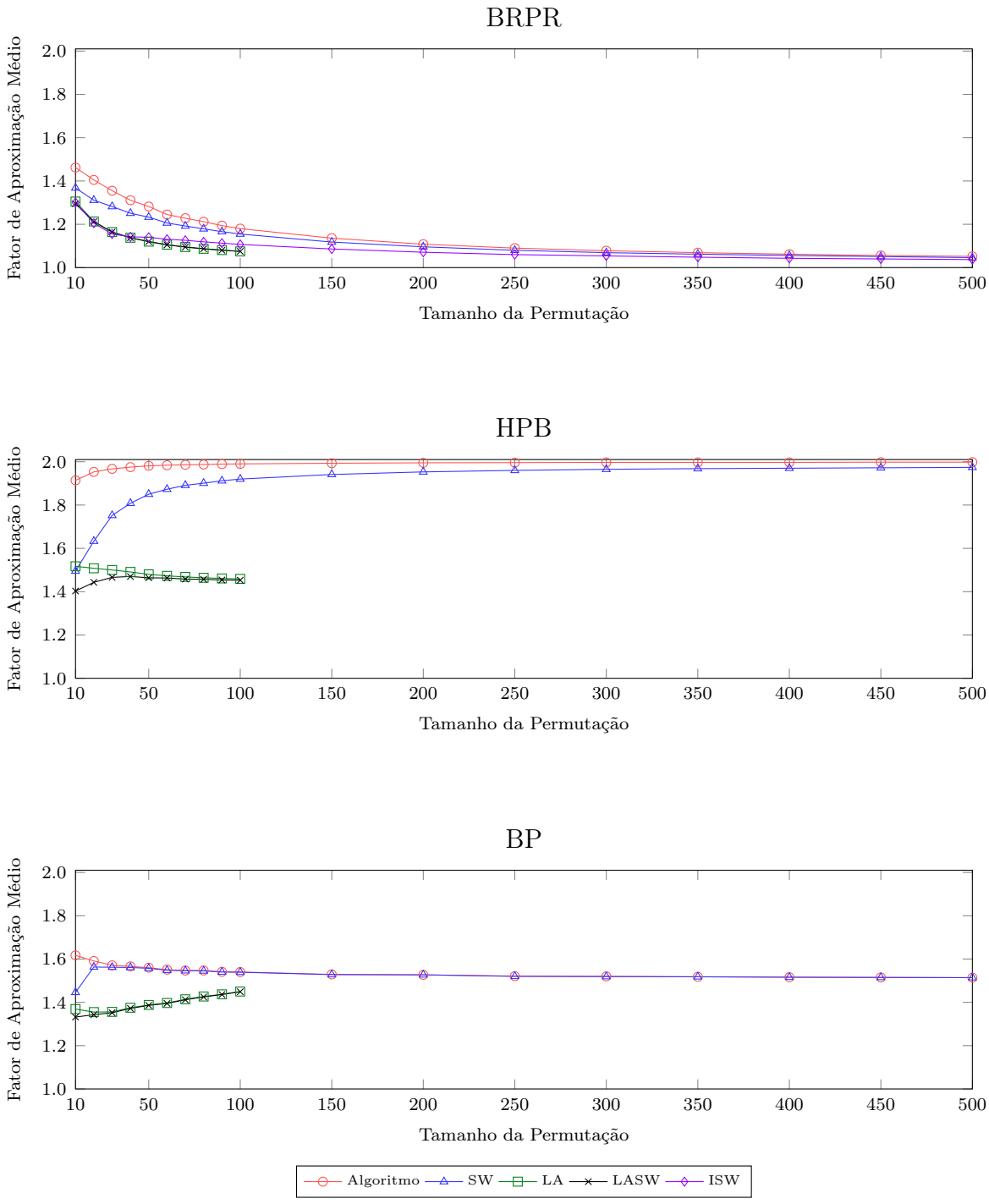

Figura 2.22: Comparativo entre o fator de aproximação médio das heurísticas aplicadas nos algoritmos BRPR, HPB e BP. A heurística ISW não foi aplicada nos algoritmos HPB e BP.

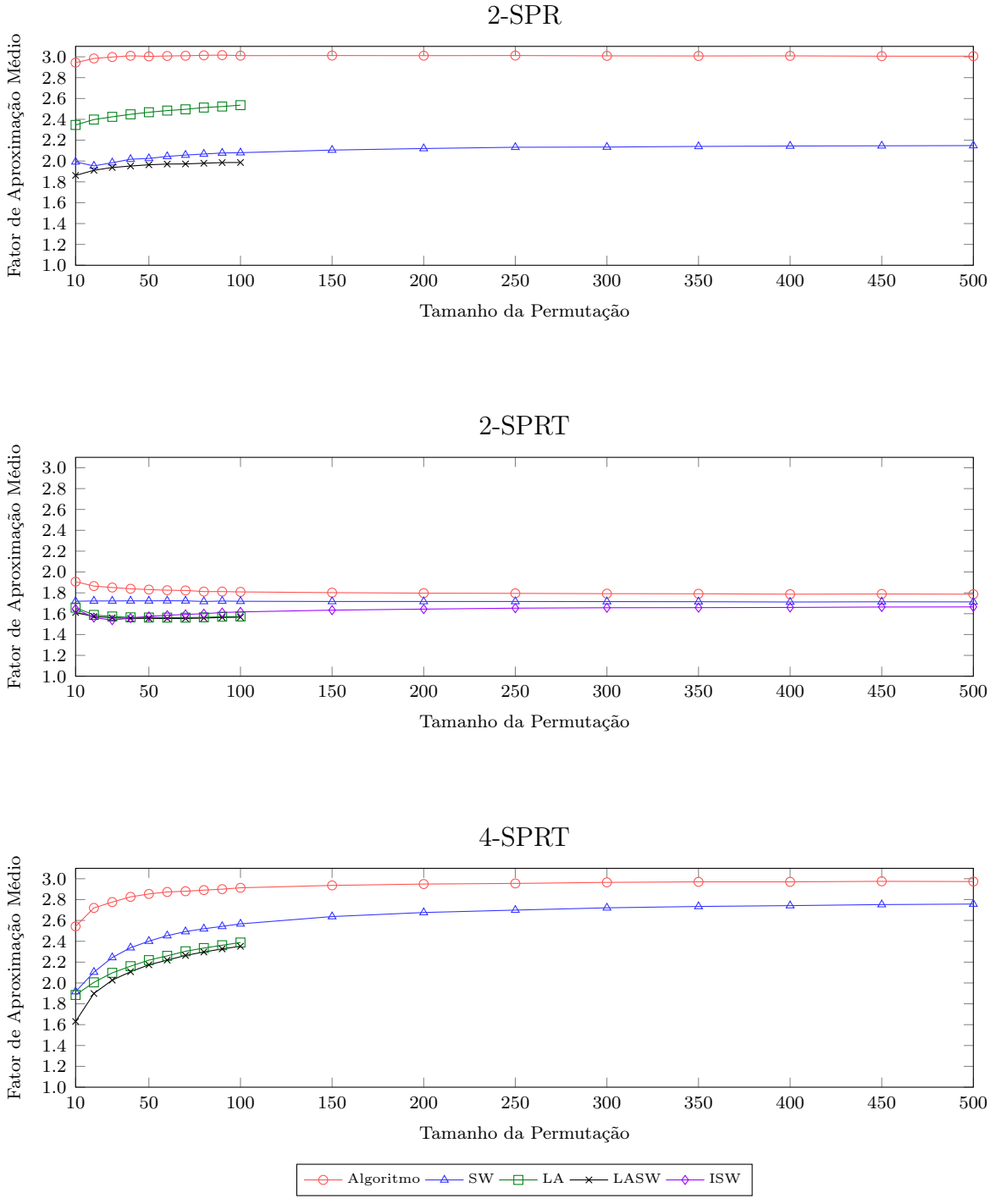

Figura 2.23: Comparativo entre o fator de aproximação médio das heurísticas com restrições de operações de prefixo. A heurística ISW não foi aplicada nos algoritmos 2-SPR e 4-SPRT.

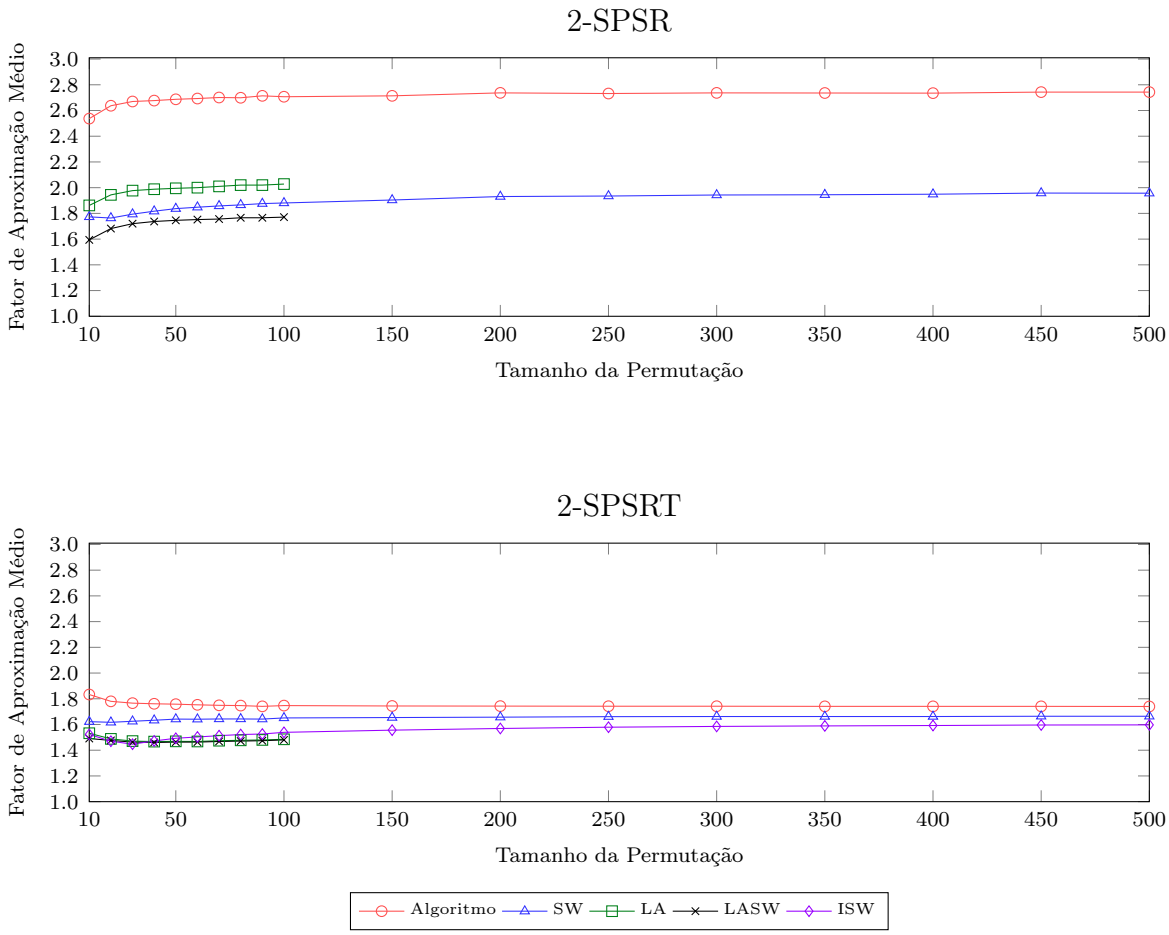

Figura 2.24: Comparativo entre o fator de aproximação médio das heurísticas com restrições de operações de prefixo ou sufixo. A heurística ISW não foi aplicada no algoritmo 2-SPSR.

A Tabela 2.19 mostra os valores da razão de aproximação média das heurísticas para os conjuntos de permutações com tamanhos 100 e 500. Escolhemos esses conjuntos com base nos maiores tamanhos de permutações que foram utilizadas como entrada por nossas heurísticas, sendo 100 para a heurística LA, e 500 para as heurísticas SW e ISW.

Tabela 2.19: Comparação do fator de aproximação médio das heurísticas.

Tamanho = 100					
Operações sem Restrições					
Algoritmo	ALG	SW	LA	LASW	ISW
RSH	1.559	1.347	1.208	1.208	1.267
WDM	1.982	1.368	1.575	1.555	1.316
BRPT	1.402	1.376	1.091	1.091	1.277
BRPR	1.180	1.155	1.075	1.075	1.107
HPB	1.990	1.920	1.459	1.453	-
BP	1.540	1.539	1.450	1.449	-
Operações de Prefixo					
Algoritmo	ALG	SW	LA	LASW	ISW
2-SPR	3.011	2.080	2.536	1.986	-
2-SPRT	1.809	1.719	1.573	1.566	1.617
4-SPRT	2.913	2.567	2.388	2.352	-
Operações de Prefixo ou Sufixo					
Algoritmo	ALG	SW	LA	LASW	ISW
2-SPSR	2.707	1.881	2.028	1.771	-
2-SPSRT	1.747	1.651	1.485	1.480	1.539

Tamanho = 500			
Operações sem Restrições			
Algoritmo	ALG	SW	ISW
RSH	1.520	1.322	1.263
WDM	1.997	1.356	1.297
BRPT	1.277	1.272	1.252
BRPR	1.052	1.046	1.037
HPB	1.998	1.974	-
BP	1.514	1.514	-
Operações de Prefixo			
Algoritmo	ALG	SW	ISW
2-SPR	3.006	2.148	-
2-SPRT	1.787	1.713	1.665
4-SPRT	2.973	2.757	-
Operações de Prefixo ou Sufixo			
Algoritmo	ALG	SW	ISW
2-SPSR	2.743	1.957	-
2-SPSRT	1.740	1.664	1.597

A heurística SW executa de maneira extremamente rápida e fornece soluções de boa qualidade. Por isso, recomendamos o uso dessa heurística em cenários onde o tempo de execução é um fator prioritário.

A heurística LA, por sua vez, apresenta um tempo de execução mais elevado, mas em contrapartida consegue obter as melhores soluções. Logo, seu uso é mais indicado quando o foco é qualidade da solução.

A heurística ISW é a que fornece o melhor custo-benefício entre qualidade da solução e tempo de execução. Com os parâmetros que foram adotados foi possível obter soluções melhores que a heurística SW, com um tempo significativamente menor do que a heurística LA.

Capítulo 3

Aproximações para Ordenação de Permutação com Sinais por Reversões e Transposições Ponderadas

Existem variações de problemas nos quais se consideram pesos distintos para diferentes eventos de rearranjo. Isso é bastante útil quando sabemos que determinados eventos de rearranjo apresentam maior probabilidade de ocorrer do que outros [5, 33]. Continuamos querendo transformar um genoma em outro utilizando os eventos permitidos por um modelo de rearranjo. Antes, queríamos encontrar uma sequência de eventos com tamanho mínimo para realizar essa tarefa. Agora, queremos encontrar uma sequência de eventos com custo mínimo, sendo o custo dado pelo somatório dos pesos dos eventos utilizados.

Em 1999, Gu e coautores [23] apresentaram um evento de rearranjo chamado de *transreversão*. Esse evento é similar ao evento de transposição, diferenciando-se pelo fato de aplicar um evento de reversão em um dos blocos afetado pela transposição. Ainda no trabalho, os autores apresentaram um algoritmo 2-aproximado para o problema de Ordenação de Permutações com Sinais por Reversões, Transposições e Transreversões.

Em 2002, Eriksen [17] apresentou um algoritmo com fator de aproximação $\frac{7}{6}$ e um *polynomial-time approximation scheme* (PTAS) para o problema de Ordenação de Permutações com Sinais por Reversões, Transposições e Transreversões Ponderadas, adotando os pesos 1 para reversões e 2 para transposições e transreversões.

Em 2007, Bader e Ohlebusch [2] apresentaram um algoritmo com fator de aproximação 1.5 para o problema de Ordenação de Permutações com Sinais por Reversões, Transposições e Transreversões Ponderadas, adotando os pesos 1 para reversões e qualquer valor em um intervalo de 1 até 2 para transposições e transreversões.

Neste capítulo, apresentaremos limitantes inferiores e algoritmos de aproximação para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas. O evento de Transreversão não será abordado nesse trabalho, mas o mesmo pode ser simulado adotando o custo dos eventos de transposição e reversão. Adotaremos os pesos 2 e 3 para reversões e transposições, respectivamente. Os pesos que adotamos são baseados nos experimentos realizados por Eriksen [16], que otimizou os pesos utilizados para cada evento e concluiu que os melhores valores são 2 para reversão e 3 para transposição. Nos experimentos realizados por Eriksen [16], também foi considerado o evento de

transreversão, mas concluiu-se que não era uma boa escolha utilizar pesos distintos para os eventos de transposição e transreversão.

O capítulo tem início com a explicação dos algoritmos (Seção 3.1). Em seguida, são apresentados os resultados práticos (Seção 3.2) e, por fim, estendemos nossa investigação para considerar diferentes pesos para os eventos de reversão e transposição (Seção 3.3).

3.1 Algoritmos

Os algoritmos que serão apresentados foram desenvolvidos para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas, sendo que os pesos adotados para os eventos de reversão e transposição foram 2 e 3, respectivamente. Inicialmente vamos apresentar conceitos e definições utilizados durante este capítulo.

Definição 1. w_ρ representa o custo de um evento de reversão.

Definição 2. w_τ representa o custo de um evento de transposição.

Definição 3. $w_{RT}(\pi, w_\rho, w_\tau)$ representa o custo da solução para ordenar a permutação π adotando w_ρ e w_τ como custos para reversão e transposição, respectivamente.

Lema 1. *Dada uma permutação π e um evento de reversão ρ, temos que $b(\pi) - 2 \leq b(\pi \circ \rho) \leq b(\pi) + 2$.*

Demonstração. Como um evento de reversão $\rho(i, j)$ afeta dois pontos de uma permutação, então é possível adicionar até dois novos breakpoints, remover até dois breakpoints ou manter o número de breakpoints inalterado. □

Lema 2. *Dada uma permutação π e um evento de transposição τ, temos que $b(\pi) - 3 \leq b(\pi \circ \tau) \leq b(\pi) + 3$.*

Demonstração. Como um evento de transposição $\tau(i, j, k)$ afeta três pontos de permutação, então é possível adicionar até três novos breakpoints, remover até três breakpoints ou manter o número de breakpoints inalterado. □

Pelos Lemas 1 e 2 é possível obter o seguinte limitante inferior:

Lema 3. *Dada uma permutação π, $w_{RT}(\pi, w_\rho = 2, w_\tau = 3) \geq b(\pi)$.*

3.1.1 Algoritmo 3-aproximado

Iremos apresentar o algoritmo 3-WRT para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas. Esse algoritmo é baseado em *breakpoints*. Agindo de forma gulosa, o algoritmo tenta remover o maior número de *breakpoints* a cada iteração. Enquanto a permutação não estiver ordenada, o algoritmo 3-WRT aplicará um dos cinco passos na ordem de prioridade a seguir:

(i) Aplique uma transposição que remove três *breakpoints*.

(ii) Aplique uma reversão que remove dois *breakpoints*.

(iii) Aplique uma transposição que remove dois *breakpoints*.

(iv) Aplique uma reversão que remove um *breakpoint*.

(v) Aplique uma transposição que remove um *breakpoint*.

O seguinte lema foi dado por Walter e coautores [32].

Lema 4. *Para qualquer permutação $\pi \neq \iota$ existe um evento de reversão ou um evento de transposição que remove pelo menos um breakpoint [32].*

Teorema 1. *O 3-WRT é um algoritmo 3-aproximado para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas.*

Demonstração. Pelos lemas 1, 2 e 4, temos a garantia de que para qualquer permutação π um dos passos do algoritmo será executado. Como o algoritmo remove pelo menos um *breakpoint* a cada iteração, então serão aplicadas no máximo $b(\pi)$ operações para ordenar a permutação π. Logo, $w_{RT}(\pi, w_\rho, w_\tau) \leq \max\{w_\rho, w_\tau\} \times b(\pi)$ implicando que $w_{RT}(\pi, w_\rho, w_\tau) \leq 3 \times b(\pi)$. Pelo Lema 3 temos que $w_{RT}(\pi, w_\rho, w_\tau) \geq b(\pi)$, logo o algoritmo é 3-aproximado. □

3.1.2 Algoritmo 2-aproximado

Apresentaremos o algoritmo 2-WRT para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas. Esse algoritmo também é baseado em *breakpoints*. Agindo de forma gulosa, o algoritmo tenta remover o maior número de *breakpoints* e, sempre que possível, mantém a permutação com *strips* negativas. Enquanto a permutação não estiver ordenada, o algoritmo 2-WRT aplicará um dos cinco passos na ordem de prioridade a seguir:

(i) Aplique uma transposição que remove três *breakpoints*.

(ii) Aplique uma reversão que remove dois *breakpoints*.

(iii) Aplique uma transposição que remove dois *breakpoints*.

(iv) Aplique uma reversão que remove um *breakpoint* e mantém a permutação com pelo menos uma *strip* negativa.

(v) Aplique uma reversão nos dois primeiros *breakpoints* da permutação.

Os dois lemas seguintes foram dados por Kececioglu e Sankoff [26].

Lema 5. *É possível aplicar uma reversão que remove um breakpoint em qualquer permutação com strips negativas [26].*

Lema 6. *Seja π uma permutação com uma strip negativa. Se toda reversão que remove um breakpoint de π torna a permutação sem strips negativas, então existe uma reversão que se aplicada em π remove dois breakpoints [26].*

Teorema 2. *O 2-WRT é um algoritmo 2-aproximado para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas.*

Demonstração. Pelos lemas 1 e 2, temos a garantia de que para qualquer permutação π um dos passos do algoritmo será executado. O passo (i) remove três *breakpoints*. Os passos (ii) e (iii) removem dois *breakpoints*. O passo (iv) remove um *breakpoint* com uma reversão desde que a permutação resultante possua pelo menos uma *strip* negativa. O passo (v) aplica um evento de reversão que não remove nenhum *breakpoint* mas, em contrapartida, gera uma *strip negativa* na permutação. Analisando o fator de aproximação de cada passo, observamos que os passos (i) e (ii) apresentam fator de aproximação 1, o passo (iii) apresenta fator de aproximação $\frac{3}{2}$ e o passo (iv) apresenta um fator de aproximação 2. O passo (v), se analisado separadamente, não gera um fator de aproximação pois não remove nenhum *breakpoint*, mas, pelos lemas 5 e 6, sabemos que se o passo (v) ocorrer, em algum momento antes da permutação ficar ordenada uma reversão que remove dois *breakpoints* será aplicada pelo algoritmo, mantendo o fator de aproximação desse passo em 2. Note que não é possível aplicar eventos de transposição em uma permutação com *strips* negativas e obter como resultado uma permutação sem *strips* negativas, pois um evento de transposição não altera a orientação dos elementos de uma permutação. □

3.1.3 Algoritmo 5/3-aproximado

Apresentaremos agora o algoritmo 5/3-WRT para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas. Esse algoritmo utiliza a estrutura de grafo de ciclos para ordenar a permutação. A ideia consiste em obter um limitante inferior baseado no número de ciclos e ciclos ímpares do grafo de ciclos e analisar a contribuição que sequências de operações específicas trazem quando aplicadas em π.

Sejam w_c e w_o os pesos para cada ciclo e ciclo ímpar de $G(\pi)$, respectivamente. Definimos como $\Delta c(\pi, \gamma) = c(\pi \circ \gamma) - c(\pi)$ e $\Delta c_{odd}(\pi, \gamma) = c_{odd}(\pi \circ \gamma) - c_{odd}(\pi)$ como a variação de ciclos e ciclos ímpares ao se aplicar uma operação γ em π, respectivamente.

Dada um sequência de operações $\mathcal{S}$ aplicadas na permutação π, temos seu custo denotado por $w_{\mathcal{S}}(\pi, w_\rho, w_\tau)$. Além disso, $G(\pi \circ \mathcal{S})$ tem uma variação de $\Delta c(\pi, \mathcal{S})$ ciclos e $\Delta c_{odd}(\pi, \mathcal{S})$ ciclos ímpares em comparação com $G(\pi)$. A função objetivo que descreve o ganho em termos de ciclos e ciclos ímpares em relação ao custo das operações que foram aplicadas é dada por:

$$R_{\mathcal{S}} = \frac{w_c \Delta c(\mathcal{S}) + w_o \Delta c_{odd}(\mathcal{S})}{w_{\mathcal{S}}(\pi, w_\rho, w_\tau)}$$

Note que $\Delta c(\pi, \rho) \in \{-1, 0, 1\}$ e $\Delta c_{odd}(\pi, \rho) \in \{-2, 0, 2\}$ [12,24]. Dessa forma, temos que $R_\rho = \frac{w_c + 2w_o}{2}$ é a melhor função objetivo para uma reversão. De maneira similar, $\Delta c(\pi, \tau) \in \{-2, -1, 0, 1, 2\}$ e $\Delta c_{odd}(\pi, \tau) \in \{-2, 0, 2\}$ [3,12,32], logo $R_\tau = \frac{2w_c + 2w_o}{3}$ é a melhor função objetivo para uma transposição. Com isso, temos o seguinte limitante inferior em relação ao número de ciclos e ciclos ímpares.

Lema 7. *Dada uma permutação π temos que*

$$w_{RT}(\pi, w_\rho, w_\tau) \geq \frac{(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))}{\max\{R_\rho, R_\tau\}} = L_B(\pi)$$

Demonstração. Note que para qualquer reversão $\rho(i,j)$ temos que $R_{\rho(i,j)} \leq R_\rho$, e para qualquer transposição $\tau(i,j,k)$ temos que $R_{\tau(i,j,k)} \leq R_\tau$. Como a permutação identidade ι tem $n+1$ ciclos ímpares, podemos chegar a um limitante inferior dividindo o ganho que precisa ser obtido em termos de ciclos e ciclos ímpares para alcançarmos a permutação identidade $(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))$ pela função que apresente o melhor ganho entre uma reversão ou uma transposição ($\max\{R_\rho, R_\tau\}$). □

O funcionamento do algoritmo 5/3-WRT é baseado na transformação em permutação simples. Note que o processo que transforma uma permutação π em uma permutação simples $\hat{\pi}$ não garante que $w_{RT}(\hat{\pi}, w_\rho, w_\tau) = w_{RT}(\pi, w_\rho, w_\tau)$, mas pelo Lema 8 temos a garantia de que π e $\hat{\pi}$ possuem o mesmo limitante inferior apresentado pelo Lema 7.

Lema 8. *Dada uma permutação π e uma permutação simples $\hat{\pi}$, temos que:*

$$L_B(\pi) = L_B(\hat{\pi}).$$

Demonstração. Note que, a cada ciclo longo que é quebrado, o número de elementos e ciclos ímpares de $\hat{\pi}$ aumenta em uma unidade. Logo, temos que $c(\hat{\pi}) = c(\pi) + 1$ e $c_{odd}(\hat{\pi}) = c(\pi) + 1$. Como adicionamos mais um elemento na permutação $\hat{\pi}$, a permutação identidade terá $(n+2)$ ciclos, mantendo assim a igualdade entre os limitantes inferiores de π e $\hat{\pi}$. □

Enquanto a permutação não estiver ordenada, o algoritmo 5/3-WRT aplicará um dos oito passos na ordem de prioridade a seguir:

(i) Se existir um ciclo orientado c, aplique uma transposição em c (Figura 3.1(i)).

(ii) Se existir um ciclo divergente c de tamanho dois, aplique uma reversão em c (Figura 3.1(ii)).

(iii) Se existir um ciclo divergente c de tamanho três, aplique uma reversão em duas arestas divergentes de c (Figura 3.1(iii)).

(iv) Se existirem dois ciclos não orientados c_1 e c_2 de tamanho dois que estejam cruzados, então aplique uma sequência de duas transposições em c_1 e c_2 (Figura 3.1(iv)).

(v) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com dois ciclos não orientados c_2 e c_3, ambos de tamanho dois, aplique uma sequência de duas transposições nesses ciclos (Figura 3.1(v)).

(vi) Se existir um ciclo não orientado c_1 de tamanho três, entrelaçado com um ciclo não orientado c_2 também de tamanho três, aplique uma sequência de três transposições nesses ciclos (Figura 3.1(vi)).

(vii) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com dois ciclos não orientados c_2 e c_3, ambos de tamanho três, aplique uma sequência de três transposições nesses ciclos (Figura 3.1(vii)).

(viii) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com um ciclo não orientado c_2 de tamanho dois e com um ciclo não orientado c_3 de tamanho três, aplique uma sequência de três transposições nesses ciclos (Figura 3.1(viii)).

A Figura 3.1 mostra todas as operações de cada etapa do algoritmo sendo aplicadas e os ciclos resultantes. O passo (iii) resulta em um ciclo trivial e outro ciclo de tamanho dois, sendo que esse pode ser divergente ou não, dependendo da configuração do ciclo neste passo. O passo (v) resulta em quatro ciclos triviais e outro ciclo de tamanho três, sendo que esse pode ser divergente ou não, dependendo da configuração dos dois ciclos nesse passo. O passo (vii) resulta em seis ciclos triviais e mantém um dos ciclos não orientados de tamanho três. O passo (viii) resulta em seis ciclos triviais e mantém um dos ciclos não orientados de tamanho dois. Os demais passos resultam em ciclos triviais. Note que em todos os passos do algoritmo os novos ciclos gerados possuem tamanho menor ou igual a três. Sendo assim, a permutação continua sendo uma permutação simples durante todo o processo de ordenação.

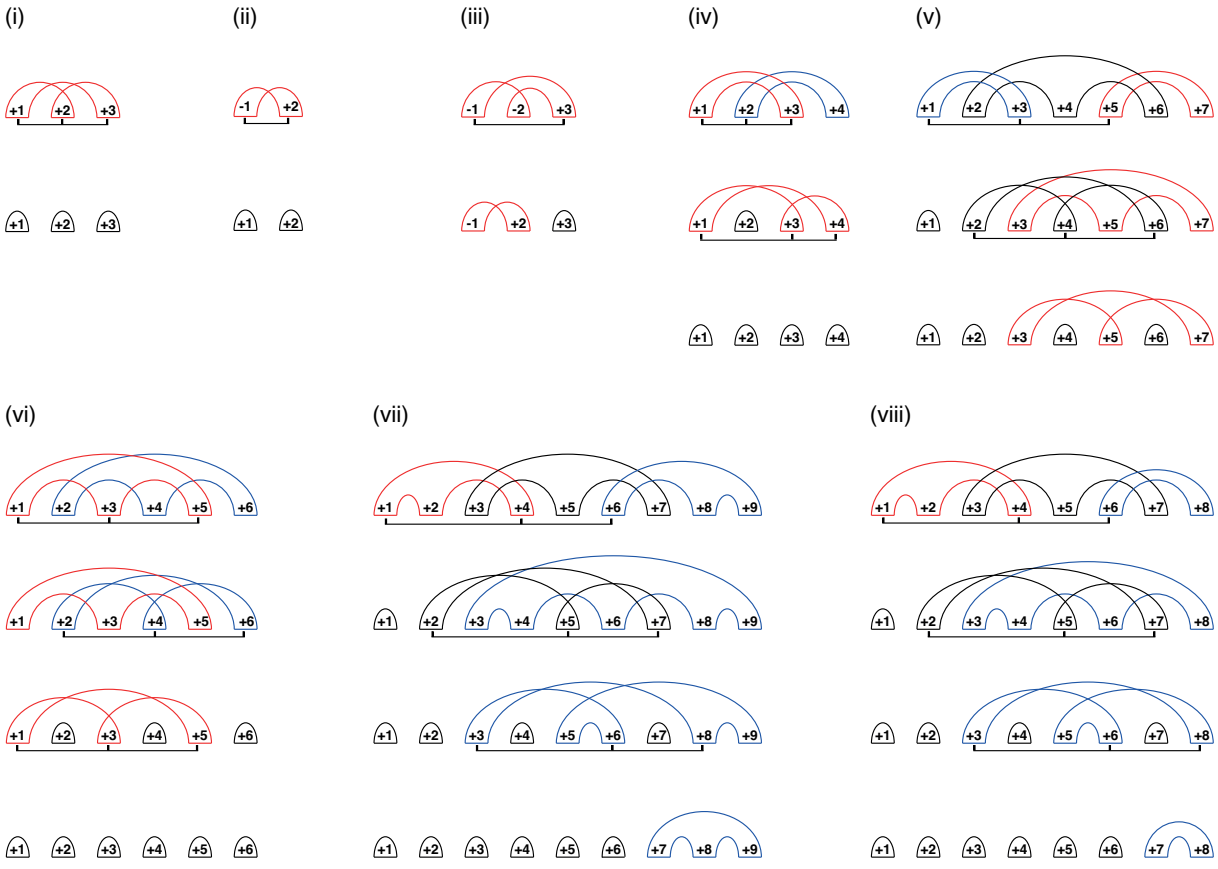

Figura 3.1: Operações aplicadas pelo algoritmo 5/3-WRT e representação de ciclos em cada uma das etapas.

Tabela 3.1: Informações sobre a variação no número de ciclos, a sequência de operações, custo e função objetivo de cada passo do algoritmo 5/3-WRT.

Passo	$\mathcal{S}$	$\Delta c(\pi, \mathcal{S})$	$\Delta c_{odd}(\pi, \mathcal{S})$	$w_{\mathcal{S}}$	$R_{\mathcal{S}}$
(i)	τ	2	2	3	$\frac{2w_c+2w_o}{3}$
(ii)	ρ	1	2	2	$\frac{1w_c+2w_o}{2}$
(iii)	ρ	1	0	2	$\frac{1w_c}{2}$
(iv)	τ, τ	2	4	6	$\frac{2w_c+4w_o}{6}$
(v)	τ, τ	2	4	6	$\frac{2w_c+4w_o}{6}$
(vi)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$
(vii)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$
(viii)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$

Lema 9. *Seja $\hat{\pi}$ uma permutação simples. Se $\hat{\pi} \neq \iota$, então é sempre possível aplicar pelo menos um dos passos do algoritmo 5/3-WRT.*

Demonstração. Como $\hat{\pi} \neq \iota$, então deve existir pelo menos um ciclo não trivial. Se $G(\hat{\pi})$ possui um ciclo orientado ou divergente, então um dos passos (i), (ii) ou (iii) pode ser aplicado. Se nenhum dos passos (i), (ii) ou (iii) foi aplicado, isso implica que $G(\hat{\pi})$ possui apenas ciclos não orientados. Bafna e Pevzner [3] mostraram que cada ciclo não orientado de tamanho dois deve cruzar pelo menos outro ciclo. Além disso, cada ciclo não orientado de tamanho três deve cruzar pelo menos dois ciclos não orientados ou entrelaçar-se com outro ciclo.

Se $G(\hat{\pi})$ tem um ciclo não orientado de tamanho três, então esse ciclo deve estar entrelaçado com outro ciclo não orientado (passo (vi) pode ser aplicado) ou então esse ciclo deve estar cruzado com outros dois ciclos. Esses dois ciclos podem ser ambos de tamanho dois (passo (v) pode ser aplicado), ambos de tamanho três (passo (vii) pode ser aplicado) ou de tamanhos diferentes (passo (viii) pode ser aplicado). Caso $G(\hat{\pi})$ não possua um ciclo não orientado de tamanho 3, então $G(\hat{\pi})$ possui um ciclo não orientado de tamanho 2 que deve estar cruzado com outro ciclo não orientado de tamanho dois (passo (iv) pode ser aplicado). □

A Tabela 3.1 mostra, para cada passo do algoritmo, a sequência de operações aplicada ($\mathcal{S}$), a variação no número de ciclos ($\Delta c(\pi, \mathcal{S})$), a variação de ciclos ímpares ($\Delta c_{odd}(\pi, \mathcal{S})$), o custo da sequência de operações ($w_{\mathcal{S}}$) e a função objetivo ($R_{\mathcal{S}}$).

O melhor fator de aproximação é obtido quando adotamos os valores $w_c = 4$ e $w_o = 1$, resultando nos seguintes valores para as funções objetivos de cada passo do algoritmo: $R_{(i)} = \frac{10}{3}$, $R_{(ii)} = \frac{6}{2} = 3$, $R_{(iii)} = \frac{4}{2} = 2$, $R_{(iv)} = R_{(v)} = \frac{12}{6} = 2$ e $R_{(vi)} = R_{(vii)} = R_{(viii)} = \frac{20}{9}$.

Teorema 3. *O 5/3-WRT é um algoritmo 5/3-aproximado para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas.*

Demonstração. Pelo Lema 9, e analisando a função objetivo de cada passo do algoritmo, temos que a menor razão obtida foi 2. Logo, o custo da solução do algoritmo é:

$$\frac{(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))}{2}$$

Note que adotando os valores $w_c = 4$ e $w_o = 1$ temos que $\max\{R_\rho, R_\tau\} = \frac{10}{3}$. Então, o limitante inferior para esse caso é:

$$\frac{(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))}{\frac{10}{3}}$$

Substituindo $(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))$ por x, e dividindo o custo da solução do algoritmo pelo limitante inferior, chegamos na seguinte razão de aproximação:

$$\frac{\frac{x}{2}}{\frac{3x}{10}} = \frac{10x}{6x} = \frac{5}{3} \approx 1.667$$

□

3.1.4 Algoritmo 3/2-aproximado

Apresentaremos agora o algoritmo 3/2-WRT para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas. O processo de ordenação desse algoritmo também é baseado no grafo de ciclos de uma permutação simples. Dessa forma, utilizaremos o Lema 8 juntamente com o limitante inferior apresentado no Lema 7 para provar o fator de aproximação $\frac{3}{2}$.

Enquanto a permutação não estiver ordenada, o algoritmo 3/2-WRT aplicará um dos doze passos na ordem de prioridade a seguir:

(i) Se existir um ciclo orientado c, aplique uma transposição em c (Figura 3.3(i)).

(ii) Se existir um ciclo divergente c de tamanho dois, aplique uma reversão em c (Figura 3.3(ii)).

(iii) Se existir um ciclo divergente c de tamanho três, tal que toda aresta cinza do ciclo é cruzada por pelo menos uma aresta cinza do mesmo ciclo, aplique uma sequência de duas reversões (Figura 3.3(iii)). Os ciclos com essa característica são exibidos na Figura 3.2.

(iv) Se existirem dois ciclos não orientados c_1 e c_2 cruzados, ambos de tamanho dois, aplique uma sequência de duas transposições em c_1 e c_2 (Figura 3.3(iv)).

(v) Se existir um ciclo não orientado c_1 de tamanho dois, cruzado com um ciclo divergente c_2 de tamanho três, aplique uma sequência de duas reversões nesses ciclos (Figura 3.3(v)).

(vi) Se existirem dois ciclos divergentes c_1 e c_2 de tamanho três, que estejam cruzados, aplique um sequência de três reversões nesses ciclos (Figura 3.3(vi)).

(vii) Se existir um ciclo não orientado c_1 de tamanho três, entrelaçado com um ciclo não orientado c_2, também de tamanho três, aplique uma sequência de três transposições nesses ciclos (Figura 3.3(vii)).

(viii) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com dois ciclos não orientados c_2 e c_3, ambos de tamanho dois, aplique uma sequência de duas transposições nesses ciclos (Figura 3.3(viii)).

(ix) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com dois ciclos não orientados c_2 e c_3, ambos de tamanho três, aplique uma sequência de transposições nesses ciclos (Figura 3.3(ix)).

(x) Se existir um ciclo não orientado c_1 de tamanho três, cruzado por um ciclo não orientado c_2 de tamanho dois e por um ciclo não orientado c_3 de tamanho três, aplique uma sequência de três transposições nesses ciclos (Figura 3.3(x)).

(xi) Se existir um ciclo não orientado c_1 de tamanho três, entrelaçado com um ciclo divergente c_2 de tamanho três, aplique uma sequência de quatro reversões nesses ciclos (Figura 3.3(xi)).

(xii) Se existir um ciclo não orientado c_1 de tamanho três, cruzado com um ciclo divergente c_2 de tamanho três, aplique uma sequência de três reversões nesses ciclos (Figura 3.3(xii)).

A Figura 3.3 mostra todas as operações de cada etapa do algoritmo sendo aplicadas e os ciclos resultantes. O passo (v) resulta em três ciclos triviais e outro ciclo de tamanho dois. Os passos (vi) e (xii) resultam em quatro ciclos triviais e outro ciclo de tamanho dois. O passo (viii) resulta em quatro ciclos triviais e outro ciclo de tamanho três, sendo que esse pode ser divergente ou não, dependendo da configuração dos dois ciclos nesse passo. O passo (ix) resulta em seis ciclos triviais e mantém um dos ciclos não orientados de tamanho três. O passo (x) resulta em seis ciclos triviais e mantém um dos ciclos não orientados de tamanho dois. Os demais passos resultam em ciclos triviais.

Note que em todos os passos do algoritmo os novos ciclos gerados possuem tamanho menor ou igual a três. Sendo assim, a permutação continua sendo uma permutação simples durante todo o processo de ordenação.

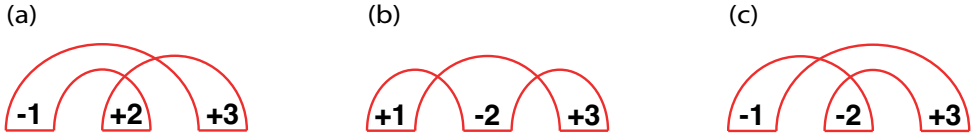

Figura 3.2: Ciclos divergentes onde as arestas cinzas do próprio ciclo fecham os possíveis open gates. (a) $c_1 = (+3, -1, +2)$, (b) $c_2 = (+3, +1, -2)$ e (c) $c_3 = (+3, -2, -1)$.

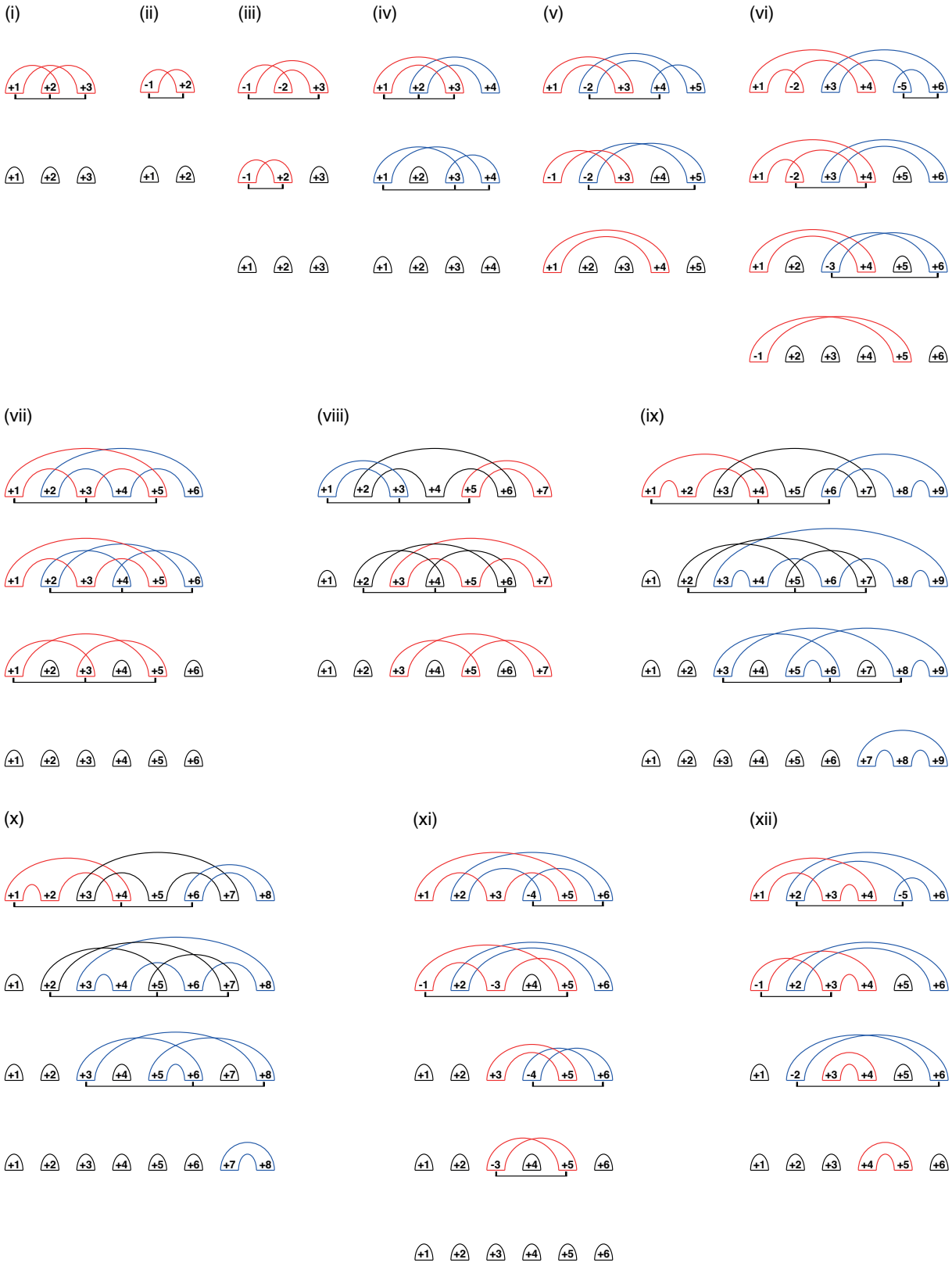

Figura 3.3: Operações aplicadas pelo algoritmo 3/2-WRT e representação de ciclos em cada uma das etapas.

Lema 10. *Seja $\hat{\pi}$ uma permutação simples. Se $\hat{\pi} \neq \iota$, então é sempre possível aplicar pelo menos um dos passos do algoritmo 3/2-WRT.*

Demonstração. Se $\hat{\pi} \neq \iota$, então deve existir pelo menos um ciclo não trivial em $G(\hat{\pi})$. Se em $G(\hat{\pi})$ existir um ciclo orientado ou um ciclo divergente de tamanho dois então os passos (i) ou (ii) podem ser aplicados.

Perceba que nesse ponto só podem existir ciclos não orientados de tamanho dois e três e ciclos divergentes de tamanho três.

O passo (iii) trata um caso específico de ciclos divergentes de tamanho três em que as próprias arestas cinzas do ciclo fecham os seus possíveis open gates. Perceba que isso é importante para o algoritmo, como o grafo de ciclos é construído a partir de uma permutação, então não devem existir open gates em $G(\hat{\pi})$. Logo, os ciclos restantes devem estar cruzados com outro(s) ciclo(s) para fechar seus possíveis open gates pois, caso contrário, teríamos pelo menos um open gate em $G(\hat{\pi})$ e não seria possível gerar uma permutação.

Supondo que existem apenas ciclos do tipo não orientado de tamanho dois e divergente de tamanho três, e sabendo que eles devem estar cruzados para fechar seus possíveis open gates, temos as seguinte possibilidades:

- Ciclo não orientado de tamanho dois cruzado com ciclo não orientado de tamanho dois (passo (iv) pode ser aplicado).

- Ciclo não orientado de tamanho dois cruzado com ciclo divergente de tamanho três (passo (v) pode ser aplicado).

- Ciclo divergente de tamanho três cruzado com ciclo não orientado de tamanho dois (passo (v) pode ser aplicado).

- Ciclo divergente de tamanho três cruzado (ou entrelaçado) com ciclo divergente de tamanho três (passo (vi) pode ser aplicado).

Perceba que passos (iv), (v) e (vi) do algoritmo cobrem todos os possível cruzamentos entre os ciclos não orientados de tamanho dois e divergentes de tamanho três.

Se a permutação não está ordenada e o algoritmo chegou até esse ponto sem aplicar nenhuma sequência de operações, então temos pelo menos um ciclo não orientado de tamanho três em $G(\hat{\pi})$ pois, caso contrário, algum dos passos anteriores já teria sido aplicado.

Nesse ponto o algoritmo tratará todos os possíveis cruzamentos que um ciclo não orientado de tamanho três pode ter. Sabemos que ele deve estar entrelaçado com outro ciclo de tamanho três ou cruzado com pelo menos dois ciclos para fechar seus possíveis open gates. Dessa forma, podemos encontrar as seguintes configurações de ciclos:

- Ciclo não orientado de tamanho três entrelaçado com ciclo não orientado de tamanho três (passo (vii) pode ser aplicado).

- Ciclo não orientado de tamanho três entrelaçado com ciclo divergente de tamanho três (passo (xi) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com dois ciclos não orientados, ambos de tamanho dois (passo (viii) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com dois ciclos não orientados, ambos de tamanho três (passo (ix) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com dois ciclos não orientados de tamanhos distintos (passo (x) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com um ciclo não orientado de tamanho dois e um ciclo divergente de tamanho três (passo (xii) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com um ciclo não orientado de tamanho três e um ciclo divergente de tamanho três (passo (xii) pode ser aplicado).

- Ciclo não orientado de tamanho três cruzado com dois ciclos divergentes de tamanho três (passo (xii) pode ser aplicado).

Perceba que os passos (vii), (viii), (ix), (x), (xi) e (xii) cobrem todos os possível cruzamentos que um ciclo não orientado de tamanho três pode ter nesse ponto do algoritmo. Dado que cobrimos todas as possibilidades de ciclos não triviais em um grafo de ciclos, para qualquer permutação simples $\hat{\pi} \neq \iota$ temos a garantia de que é possível aplicar pelo menos um dos passos do algoritmo 3/2-WRT.

□

A Tabela 3.2, mostra para cada passo do algoritmo, a sequência de operações aplicada ($\mathcal{S}$), a variação no número de ciclos ($\Delta c(\pi, \mathcal{S})$), a variação de ciclos ímpares ($\Delta c_{odd}(\pi, \mathcal{S})$), o custo da sequência de operações ($w_{\mathcal{S}}$) e a função objetivo ($R_{\mathcal{S}}$).

O melhor fator de aproximação é obtido quando adotamos os valores $w_c = 2$ e $w_o = 1$, resultando nos seguintes valores para as funções objetivos de cada passo do algoritmo: $R_{(i)} = R_{(ii)} = 2$, $R_{(iii)} = R_{(v)} = R_{(xi)} = \frac{3}{2}$ e $R_{(iv)} = R_{(vi)} = R_{(vii)} = R_{(viii)} = R_{(ix)} = R_{(x)} = R_{(xii)} = \frac{4}{3}$.

Tabela 3.2: Informações sobre a variação no número de ciclos, a sequência de operações, custo e função objetivo de cada passo do algoritmo 3/2-WRT.

Passo	$\mathcal{S}$	$\Delta c(\pi, \mathcal{S})$	$\Delta c_{odd}(\pi, \mathcal{S})$	$w_{\mathcal{S}}$	$R_{\mathcal{S}}$
(i)	τ	2	2	3	$\frac{2w_c+2w_o}{3}$
(ii)	ρ	1	2	2	$\frac{1w_c+2w_o}{2}$
(iii)	ρ, ρ	2	2	4	$\frac{2w_c+2w_o}{4}$
(iv)	τ, τ	2	4	6	$\frac{2w_c+4w_o}{6}$
(v)	ρ, ρ	2	2	4	$\frac{2w_c+2w_o}{4}$
(vi)	ρ, ρ, ρ	3	2	6	$\frac{3w_c+2w_o}{6}$
(vii)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$
(viii)	τ, τ	2	4	6	$\frac{2w_c+4w_o}{6}$
(ix)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$
(x)	τ, τ, τ	4	4	9	$\frac{4w_c+4w_o}{9}$
(xi)	ρ, ρ, ρ, ρ	4	4	8	$\frac{4w_c+4w_o}{8}$
(xii)	ρ, ρ, ρ	3	2	6	$\frac{3w_c+2w_o}{6}$

Teorema 4. *O 3/2-WRT é um algoritmo 3/2-aproximado para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas.*

Demonstração. Pelo Lema 10, e analisando a função objetivo de cada passo do algoritmo, temos que a menor razão obtida foi $\frac{4}{3}$. Logo, o custo da solução do algoritmo é:

$$\frac{(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))}{\frac{4}{3}}$$

Note que adotando os valores $w_c = 2$ e $w_o = 1$ temos que $\max\{R_\rho, R_\tau\} = 2$. Então, o limitante inferior para esse caso é:

$$\frac{(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))}{2}$$

Substituindo $(w_c + w_o)(n+1) - (w_c \times c(\pi) + w_o \times c_{odd}(\pi))$ por x, e dividindo o custo da solução do algoritmo pelo limitante inferior, chegamos na seguinte razão de aproximação:

$$\frac{\frac{3x}{4}}{\frac{x}{2}} = \frac{6x}{4x} = \frac{3}{2}$$

□

3.2 Resultados Experimentais

Para verificar os resultados práticos de nosso algoritmos, desenvolvemos cinco grupos de testes que foram utilizados como entrada pelos mesmos. Cada grupo apresenta diversos conjuntos de permutações com características específicas. Apresentamos a seguir as características de cada grupo e como foram construídos.

- GR01: cada conjunto desse grupo apresenta um total de 10.000 permutações com um tamanho específico. O identificador de cada conjunto é dado pelo tamanho das permutações contidas no mesmo. As permutações foram geradas de maneira aleatória mantendo o maior número de *breakpoints* possível. Para esse grupo, foram criados conjuntos com identificadores de 10 até 500, em intervalos de 5.

- GR02: cada conjunto desse grupo apresenta um total de 10.000 permutações com um tamanho específico. O identificador de cada conjunto é dado pelo tamanho das permutações contidas no mesmo. As permutações foram geradas a partir da permutação identidade aplicando uma quantidade fixa de 20 eventos de maneira aleatória, sendo que dentre os eventos aleatórios aplicados foi mantida uma relação de 50% de reversões e 50% de transposições. Para esse grupo, foram criados conjuntos com identificadores de 10 até 100, em intervalos de 5.

- GR03: cada conjunto desse grupo apresenta um total de 10.000 permutações com um tamanho específico. O identificador de cada conjunto é dado pelo tamanho das permutações contidas no mesmo. As permutações foram geradas a partir da

permutação identidade aplicando 20% do tamanho da permutação em eventos de maneira aleatória, sendo que dentre os eventos aleatórios aplicados foi mantida uma relação de 50% de reversões e 50% de transposições. Para esse grupo, foram criados conjuntos com identificadores de 10 até 100, em intervalos de 10.

- GR04: cada conjunto desse grupo apresenta um total de 10.000 permutações de tamanho 100. As permutações foram geradas a partir da permutação identidade aplicando uma porcentagem variável do tamanho da permutação em eventos de maneira aleatória, sendo que dentre os eventos aleatórios aplicados foi mantida uma relação de 50% de reversões e 50% de transposições. O identificador de cada conjunto é dado pela porcentagem de operações aleatórias que foram aplicadas em cada permutação. Para esse grupo, foram criados conjuntos com identificadores de 10 até 100, em intervalos de 10.

- GR05: cada conjunto desse grupo apresenta um total de 10.000 permutações com um tamanho específico. O identificador de cada conjunto é dado pelo tamanho das permutações contidas no mesmo. As permutações foram geradas a partir da permutação identidade aplicando 20% do tamanho da permutação em eventos de maneira aleatória, sendo que dentre os eventos aleatórios aplicados foi mantida uma relação de 60% de reversões e 40% de transposições. Para esse grupo, foram criados conjuntos com identificadores de 50 até 500, em intervalos de 50.

A elaboração desses grupos busca testar nossos algoritmos com permutações com características distintas. Dessa forma, podemos fazer um comparativo do desempenho de nossos algoritmos em diferentes cenários. Os grupos de teste estão disponíveis no endereço eletrônico:

`https://www.ic.unicamp.br/~klairton/dataset/SbWRT/`.

De maneira geral, nossos algoritmos apresentaram um desempenho satisfatório, inclusive os algoritmos 3-WRT e 2-WRT, que são os que apresentam um fator de aproximação mais elevado, mostraram bons resultados na prática.

As tabelas 3.3, 3.4, 3.5, 3.6 e 3.7 mostram os fatores de aproximação mínimo, médio e máximo de nossos algoritmos nos grupos de teste GR01, GR02, GR03, GR04 e GR05, respectivamente.

Tabela 3.3: Fatores de aproximação mínimo, médio e máximo dos algoritmos utilizando o grupo de teste GR01.

Fator de Aproximação Mínimo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.000	1.000	1.000	1.000
50	1.028	1.028	1.041	1.041
100	1.027	1.027	1.034	1.034
150	1.023	1.023	1.032	1.032
200	1.020	1.020	1.017	1.017
250	1.019	1.019	1.016	1.016
300	1.018	1.018	1.022	1.022
350	1.013	1.013	1.017	1.017
400	1.015	1.015	1.018	1.020
450	1.013	1.013	1.016	1.016
500	1.012	1.012	1.013	1.013

Fator de Aproximação Médio				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.180	1.175	1.222	1.215
50	1.118	1.117	1.145	1.143
100	1.081	1.081	1.099	1.098
150	1.061	1.061	1.075	1.074
200	1.050	1.049	1.060	1.060
250	1.042	1.042	1.051	1.051
300	1.036	1.036	1.044	1.044
350	1.032	1.032	1.039	1.039
400	1.029	1.029	1.035	1.035
450	1.026	1.026	1.032	1.032
500	1.024	1.024	1.029	1.029

Fator de Aproximação Máximo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.692	1.667	1.571	1.467
50	1.233	1.230	1.270	1.271
100	1.152	1.156	1.174	1.176
150	1.122	1.122	1.133	1.133
200	1.092	1.092	1.100	1.100
250	1.075	1.075	1.089	1.089
300	1.063	1.061	1.074	1.071
350	1.055	1.055	1.069	1.065
400	1.050	1.050	1.055	1.055
450	1.043	1.043	1.051	1.050
500	1.039	1.039	1.048	1.047

Pela Tabela 3.3, com base no fator de aproximação médio e no conjunto de permutações de tamanho 500, nota-se que as soluções fornecidas pelos nossos algoritmos em permutações maiores estão muito próximas do ótimo. Considerando o fator de aproximação máximo em permutações de tamanho 500 podemos ver que em todos os algoritmos esse valor foi estritamente menor que 1.050. Vale lembrar que o fator de aproximação foi computado em relação ao limitante inferior que é menor ou igual que o valor ótimo.

Tabela 3.4: Fatores de aproximação mínimo, médio e máximo dos algoritmos utilizando o grupo de teste GR02.

Fator de Aproximação Mínimo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.000	1.000	1.000	1.000
20	1.000	1.000	1.000	1.000
30	1.000	1.000	1.000	1.000
40	1.000	1.000	1.000	1.000
50	1.020	1.020	1.000	1.000
60	1.000	1.000	1.000	1.000
70	1.000	1.000	1.000	1.000
80	1.000	1.000	1.000	1.000
90	1.000	1.000	1.000	1.000
100	1.000	1.000	1.000	1.000

Fator de Aproximação Médio				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.173	1.169	1.217	1.210
20	1.153	1.150	1.194	1.189
30	1.140	1.138	1.173	1.169
40	1.132	1.130	1.157	1.154
50	1.128	1.125	1.143	1.140
60	1.124	1.122	1.131	1.128
70	1.122	1.120	1.121	1.117
80	1.121	1.118	1.114	1.110
90	1.120	1.117	1.106	1.102
100	1.118	1.116	1.100	1.097

Fator de Aproximação Máximo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.667	1.667	1.636	1.500
20	1.636	1.435	1.458	1.417
30	1.375	1.375	1.375	1.333
40	1.405	1.368	1.371	1.317
50	1.341	1.325	1.350	1.308
60	1.419	1.359	1.311	1.311
70	1.350	1.362	1.289	1.279
80	1.348	1.340	1.319	1.279
90	1.364	1.295	1.311	1.255
100	1.370	1.326	1.261	1.277

Pela Tabela 3.4 nota-se que para todos os algoritmos os fatores de aproximação médio e máximo foram menores que 1.175 e 1.667, respectivamente.

Tabela 3.5: Fatores de aproximação mínimo, médio e máximo dos algoritmos utilizando o grupo de teste GR03.

Fator de Aproximação Mínimo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.000	1.000	1.000	1.000
20	1.000	1.000	1.000	1.000
30	1.000	1.000	1.000	1.000
40	1.000	1.000	1.000	1.000
50	1.000	1.000	1.000	1.000
60	1.000	1.000	1.000	1.000
70	1.000	1.000	1.000	1.000
80	1.000	1.000	1.000	1.000
90	1.000	1.000	1.000	1.000
100	1.000	1.000	1.000	1.000

Fator de Aproximação Médio				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.066	1.071	1.069	1.075
20	1.097	1.099	1.100	1.100
30	1.110	1.109	1.109	1.106
40	1.118	1.116	1.112	1.107
50	1.120	1.117	1.112	1.108
60	1.122	1.119	1.111	1.106
70	1.121	1.118	1.110	1.105
80	1.121	1.118	1.106	1.102
90	1.120	1.117	1.103	1.099
100	1.117	1.115	1.100	1.097

Fator de Aproximação Máximo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10	1.333	1.333	1.333	1.333
20	1.778	1.778	1.667	1.444
30	1.571	1.583	1.571	1.500
40	1.625	1.500	1.533	1.375
50	1.600	1.450	1.429	1.400
60	1.448	1.417	1.414	1.333
70	1.394	1.375	1.387	1.333
80	1.395	1.324	1.333	1.290
90	1.375	1.341	1.310	1.286
100	1.354	1.304	1.295	1.267

Pela Tabela 3.5 podemos perceber que todos os algoritmos encontraram pelo menos uma solução ótima para cada conjunto de permutações. Notamos ainda que a diferença no fator de aproximação fornecido pelos algoritmos para esses conjuntos foi mínima.

Tabela 3.6: Fatores de aproximação mínimo, médio e máximo dos algoritmos utilizando o grupo de teste GR04.

Fator de Aproximação Mínimo				
	Algoritmos			
Operações Aplicadas	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10%	1.000	1.000	1.000	1.000
20%	1.000	1.000	1.000	1.000
30%	1.015	1.015	1.000	1.000
40%	1.023	1.023	1.022	1.022
50%	1.020	1.020	1.010	1.010
60%	1.027	1.027	1.018	1.018
70%	1.033	1.033	1.033	1.033
80%	1.031	1.031	1.023	1.023
90%	1.023	1.023	1.038	1.038
100%	1.031	1.031	1.028	1.028

Fator de Aproximação Médio				
	Algoritmos			
Operações Aplicadas	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10%	1.110	1.108	1.075	1.072
20%	1.118	1.116	1.100	1.097
30%	1.111	1.109	1.104	1.102
40%	1.104	1.102	1.106	1.105
50%	1.098	1.097	1.106	1.105
60%	1.092	1.092	1.105	1.104
70%	1.090	1.089	1.105	1.104
80%	1.087	1.087	1.104	1.103
90%	1.086	1.085	1.103	1.102
100%	1.085	1.084	1.102	1.101

Fator de Aproximação Máximo				
	Algoritmos			
Operações Aplicadas	3-WRT	2-WRT	5/3-WRT	3/2-WRT
10%	1.720	1.450	1.375	1.320
20%	1.340	1.312	1.304	1.261
30%	1.286	1.258	1.246	1.224
40%	1.222	1.200	1.229	1.224
50%	1.208	1.229	1.208	1.208
60%	1.194	1.179	1.196	1.202
70%	1.179	1.175	1.196	1.195
80%	1.175	1.175	1.203	1.197
90%	1.161	1.161	1.190	1.185
100%	1.167	1.167	1.178	1.173

Pela Tabela 3.6, com base no fator de aproximação médio, podemos perceber que com até 30% de operações aplicadas os algoritmos 5/3-WRT e 3/2-WRT obtiveram uma pequena vantagem sobre os algoritmos 3-WRT e 2-WRT. À medida que essa porcentagem aumenta, os algoritmos 3-WRT e 2-WRT passam a impor uma leve vantagem sobre os algoritmos 5/3-WRT e 3/2-WRT.

Tabela 3.7: Fatores de aproximação mínimo, médio e máximo dos algoritmos utilizando o grupo de teste GR05.

Fator de Aproximação Mínimo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
50	1.000	1.000	1.000	1.000
100	1.000	1.000	1.000	1.000
150	1.000	1.000	1.000	1.000
200	1.011	1.011	1.000	1.000
250	1.009	1.009	1.000	1.000
300	1.015	1.015	1.000	1.000
350	1.019	1.018	1.000	1.000
400	1.011	1.011	1.000	1.000
450	1.019	1.019	1.000	1.000
500	1.017	1.017	1.000	1.000

Fator de Aproximação Médio				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
50	1.120	1.117	1.115	1.110
100	1.115	1.113	1.098	1.094
150	1.107	1.105	1.081	1.078
200	1.097	1.096	1.067	1.065
250	1.088	1.087	1.055	1.053
300	1.081	1.080	1.046	1.045
350	1.074	1.074	1.039	1.038
400	1.069	1.068	1.033	1.032
450	1.064	1.064	1.028	1.027
500	1.060	1.060	1.025	1.024

Fator de Aproximação Máximo				
	Algoritmos			
Tamanho	3-WRT	2-WRT	5/3-WRT	3/2-WRT
50	1.591	1.500	1.421	1.368
100	1.349	1.326	1.302	1.262
150	1.288	1.246	1.224	1.209
200	1.227	1.230	1.195	1.172
250	1.214	1.191	1.142	1.144
300	1.164	1.178	1.124	1.117
350	1.154	1.155	1.111	1.111
400	1.140	1.138	1.123	1.112
450	1.121	1.131	1.090	1.085
500	1.124	1.118	1.081	1.081

Pela Tabela 3.7 nota-se que os algoritmos 5/3-WRT e 3/2-WRT apresentaram resultados bastante similares, o mesmo sendo verdade para os algoritmos 3-WRT e 2-WRT. O fator de aproximação médio de todos os algoritmos foi menor ou igual a 1.12.

3.3 Investigando Diferentes Ponderações

Nesta seção, apresentaremos uma investigação com diferentes pesos para w_ρ e w_τ. O problema de Ordenação de Permutações com Sinais por Reversões e Transposições Ponderadas é afetado pelos pesos atribuídos a cada evento. Isso implica que, dependendo da ponderação utilizada, um evento pode ser mais priorizado do que outro.

Tendo em vista que os algoritmos de aproximação apresentados anteriormente conse-

guem garantir um fator de aproximação apenas para um cenário específico onde $\frac{w_\tau}{w_\rho} = 1.5$, o objetivo dessa investigação é apresentar um algoritmo que possa garantir um fator de aproximação para diferentes relações de pesos entre os eventos de reversão e transposição.

Note que uma transposição $\tau(i, j, k)$ pode ser simulada por três reversões $\rho(i, k-1) \circ \rho(i, i+k-j-1) \circ \rho(i+k-j, k-1)$. Dessa forma, limitamos nossa investigação para valores onde $w_\rho \leq w_\tau \leq 3 \times w_\rho$.

Utilizamos como base o algoritmo 3/2-WRT e adicionamos sequências de operações alternativas em alguns passos para obtermos um algoritmo de ponderações genéricas que chamaremos de GWRT. Essas sequências de operações alternativas resultam na mesma configuração de ciclos quando comparadas com a sequência de operações original. Dependendo dos valores utilizados para w_ρ e w_τ, uma sequência alternativa pode tornar-se mais vantajosa do que a sequência original. Nesse caso, o algoritmo deve escolher a sequência de operações com melhor custo benefício. A Tabela 3.8 mostra as sequências de operações alternativas para cada passo do algoritmo GWRT.

Para cada sequência original de operações do algoritmo, aplicamos a técnica de *branch and bound* para encontrarmos sequências alternativas de operações que resultam na mesma configuração de ciclos. Em seguida, encontramos os valores para w_o e w_c que apresentam o melhor fator de aproximação com os pesos adotados para w_τ e w_ρ. A Tabela 3.9 mostra os melhores valores que foram obtidos. A coluna $\frac{w_\tau}{w_\rho}$ representa a relação de peso entre o evento de transposição e o evento de reversão. A coluna Aprox. mostra o fator de aproximação obtido para aquela relação de peso. As colunas w_c e w_o apresentam os valores utilizados no algoritmo para ciclos e ciclos ímpares, respectivamente.

Podemos generalizar os resultados exibidos na Tabela 3.9. Seja $r = \frac{w_\tau}{w_\rho}$, adotamos $w_c = \frac{1}{r-1}$ e $w_o = 1$ quando $r \in [1, 1.5]$ e adotamos $w_c = 2$ e $w_o = 1$ quando $r \in (1.5, 3]$. Com estes pesos garantimos uma aproximação menor ou igual a $(3-r)$ no intervalo $[1, 1.5]$ e exatamente igual a r e 2, nos intervalos $(1.5, 2]$ e $(2, 3]$, respectivamente.

Tabela 3.8: Sequências de operações alternativas para o algoritmo GWRT.

Passo	Sequência Original	Sequências Alternativas
(i)	τ	
(ii)	ρ	
(iii)	ρ, ρ	
(iv)	τ, τ	ρ, ρ, ρ
(v)	ρ, ρ	ρ, τ
(vi)	ρ, ρ, ρ	ρ, ρ, τ e ρ, τ, τ
(vii)	τ, τ, τ	$\rho, \rho, \rho, \rho, \rho$
(viii)	τ, τ	ρ, ρ, ρ, ρ
(ix)	τ, τ, τ	$\rho, \rho, \rho, \rho, \rho, \rho$ e $\rho, \rho, \rho, \rho, \rho, \tau$
(x)	τ, τ, τ	$\rho, \rho, \rho, \rho, \rho$
(xi)	ρ, ρ, ρ, ρ	ρ, ρ, τ, τ e ρ, τ, τ, τ
(xii)	ρ, ρ, ρ	ρ, τ, τ

Tabela 3.9: Fatores de aproximação obtidos adotando diferentes pesos para w_τ e w_ρ.

$\frac{w_\tau}{w_\rho}$	Aprox.	w_c	w_o	$\frac{w_\tau}{w_\rho}$	Aprox.	w_c	w_o	$\frac{w_\tau}{w_\rho}$	Aprox.	w_c	w_o	$\frac{w_\tau}{w_\rho}$	Aprox.	w_c	w_o
1.00	2.00	∞	1	1.55	1.55	2	1	2.05	2.00	2	1	2.55	2.00	2	1
1.05	1.93	25	1	1.60	1.60	2	1	2.10	2.00	2	1	2.60	2.00	2	1
1.10	1.86	25	2	1.65	1.65	2	1	2.15	2.00	2	1	2.65	2.00	2	1
1.15	1.80	33	4	1.70	1.70	2	1	2.20	2.00	2	1	2.70	2.00	2	1
1.20	1.75	6	1	1.75	1.75	2	1	2.25	2.00	2	1	2.75	2.00	2	1
1.25	1.70	14	3	1.80	1.80	2	1	2.30	2.00	2	1	2.80	2.00	2	1
1.30	1.65	15	4	1.85	1.85	2	1	2.35	2.00	2	1	2.85	2.00	2	1
1.35	1.61	22	7	1.90	1.90	2	1	2.40	2.00	2	1	2.90	2.00	2	1
1.40	1.57	8	3	1.95	1.95	2	1	2.45	2.00	2	1	2.95	2.00	2	1
1.45	1.54	16	7	2.00	2.00	2	1	2.50	2.00	2	1	3.00	2.00	2	1
1.50	1.50	2	1												

A Figura 3.4 exibe as melhores aproximações obtidas para $\frac{w_\tau}{w_\rho}$. Note que conseguimos um fator de aproximação 2 no intervalo onde $2 \leq \frac{w_\tau}{w_\rho} \leq 3$ e um fator de aproximação estritamente menor que 2 quando $1 < \frac{w_\tau}{w_\rho} < 2$. Dentre os diferentes pesos adotados para w_τ e w_ρ, o melhor fator de aproximação obtido foi 1.5, quando $\frac{w_\tau}{w_\rho} = 1.5$.

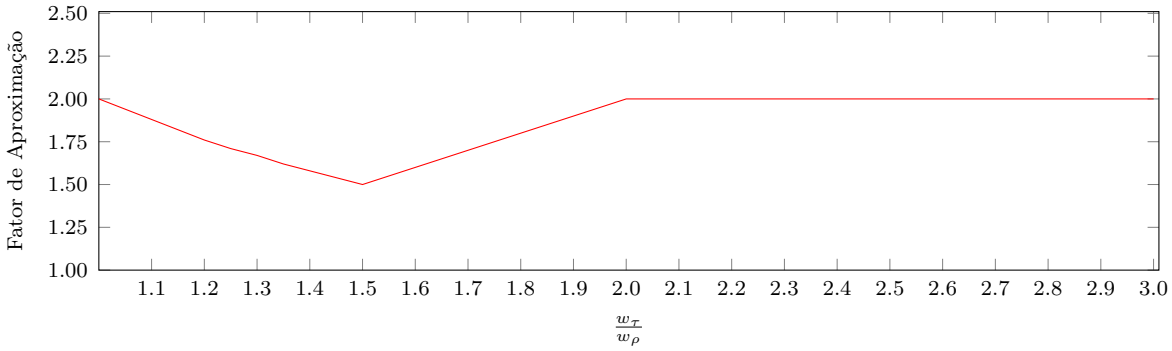

Figura 3.4: Fatores de aproximação obtidos utilizando diferentes pesos para w_τ e w_ρ.

Capítulo 4

Conclusão

Este trabalho apresentou um estudo para o problema de Ordenação de Permutações com Sinais por Reversões e Transposições, tanto na sua forma clássica quanto nas versões restritas a operações de prefixo e prefixo ou sufixo. Três heurísticas foram criadas (*Slinding Window*, *Look Ahead* e *Iterative Slinding Window*) com o intuito de melhorar as soluções de algoritmos na área de rearranjo de genomas. As heurísticas apresentadas podem adequar-se facilmente a diferentes tipos de cenários, podendo ser adaptadas para outros problemas. Mostramos ainda que é possível combinar as heurísticas *Look Ahead* e *Slinding Window* para obter resultados ainda melhores.

Além disso, apresentamos um estudo para o problema de Ordenação de Permutação com Sinais por Reversões e Transposições Ponderadas. Desenvolvemos quatro algoritmos de aproximação, sendo os mesmos com fatores de aproximação 3, 2, $\frac{5}{3}$ e $\frac{3}{2}$, considerando os pesos 2 e 3 para os eventos de reversão e transposição, respectivamente. Mostramos os resultados práticos desses algoritmos aplicados em diversos grupos de permutações com diferentes configurações.

Ampliamos nossa investigação considerando diferentes pesos para os eventos de reversão e transposição. Como resultado, apresentamos uma análise e mostramos o fator de aproximação obtido para distintas relações de peso entre as operações de transposição e reversão.

Nenhum dos problemas abordados nesse trabalho apresentam complexidade conhecida. Trabalhos futuros podem investigar essas complexidades. De maneira complementar, pode-se adicionar restrições ao problema de Ordenação de Permutações com Sinais por Operações Ponderadas resultando em novos campos de investigação. Essas restrições podem ser com base em operações de prefixo, operações de prefixo ou sufixo e ponderação em relação à quantidade de elementos afetados por cada operação.

Referências Bibliográficas

[1] David A. Bader, Bernard M. E. Moret, and Mi Yan. A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 8:483–491, 2001.

[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] 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.

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

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

[7] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Pancake Flipping is Hard. *Journal of Computer and System Sciences*, 81(8):1556–1574, 2015.

[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] David A. Christie. *Genome Rearrangement Problems*. PhD thesis, Department of Computing Science, University of Glasgow, 1998.

[11] David S. Cohen and Manuel Blum. On the Problem of Sorting Burnt Pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.

[12] Ulisses Dias, Gustavo R. Galvão, Carla N. Lintzmayer, and Zanoni Dias. A General Heuristic for Genome Rearrangement Problems. *Journal of Bioinformatics and Computational Biology*, 12(3):26, 2014.

[13] Ulisses Martins Dias. *Problemas de Comparação de Genomas*. PhD thesis, Institute of Computing, University of Campinas, 2012. In Portuguese.

[14] Zanoni Dias and João Meidanis. Sorting by Prefix Transpositions. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, volume 2476 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag Berlin Heidelberg New York, Berlin/Heidelberg, Germany, 2002.

[15] 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.

[16] Niklas Eriksen. Combinatorics of genome rearrangements and phylogeny, 2001.

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

[18] Guillaume Fertin, Loïc Jankowiak, and Géraldine Jean. Prefix and Suffix Reversals on Strings. In *String Processing and Information Retrieval*, volume 9309 of *Lecture Notes in Computer Science*, pages 165–176. Springer International Publishing, Switzerland, 2015.

[19] Guillaume Fertin, Anthony Labarre, Irena Rusu, Éric Tannier, and Stéphane Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, London, England, 2009.

[20] Johannes Fischer and Simon W. Ginzinger. A 2-Approximation Algorithm for Sorting by Prefix Reversals. In *Proceedings of the 13th Annual European Conference on Algorithms (ESA'2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 415–425, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[21] Gustavo R. Galvão and Zanoni Dias. An Audit Tool for Genome Rearrangement Algorithms. *Journal of Experimental Algorithmics*, 19:1–34, 2014.

[22] William H. Gates and Christos H. Papadimitriou. Bounds for Sorting by Prefix Reversal. *Discrete Mathematics*, 27(1):47–57, 1979.

[23] 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.

[24] 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.

[25] FanChang Hao, Zhang Melvin, and Hon Wai Leong. A 2-Approximation Scheme for Sorting Signed Permutations by Reversals, Transpositions, Transreversals, and Block-Interchanges. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, PP(99):1–1, 2017.

[26] John D. Kececioglu and David Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13:180–210, 1995.

[27] Carla Negri Lintzmayer, Guillaume Fertin, and Zanoni Dias. Sorting Permutations by Prefix and Suffix Rearrangements. *Journal of Bioinformatics and Computational Biology*, 15(1):1750002, 2017.

[28] Pavel A. Pevzner and Michael S. Waterman. Open Combinatorial Problems in Computational Molecular Biology. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'1995)*, pages 158–173, Washington, DC, USA, 1995. IEEE Computer Society.

[29] 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.

[30] Glenn Tesler. Efficient Algorithms for Multichromosomal Genome Rearrangements. *Journal of Computer and System Sciences*, 65(3):587–609, 2002.

[31] Glenn Tesler. GRIMM: Genome Rearrangements Web Server. *Bioinformatics*, 18(3):492–493, 2002.

[32] 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.

[33] Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient Sorting of Genomic Permutations by Translocation, Inversion and Block Interchange. *Bioinformatics*, 21(16):3340–3346, 2005.