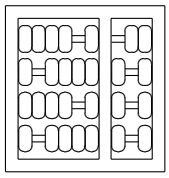

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Problema da Ordenação Ponderada por Reversões e Transposições

Eduardo Rossetti Donoso *Andre Rodrigues Oliveira*
Ulisses Dias *Zanoni Dias*

Relatório Técnico - IC-PFG-16-01 - Projeto Final de Graduação

July - 2016 - Julho

Problema da Ordenação Ponderada por Reversões e Transposições

Eduardo Rossetti Donoso* Andre Rodrigues Oliveira* Ulisses Dias† Zanoni Dias*

Resumo

Rearranjo de genomas é uma área de pesquisa que estuda a distância entre genomas contabilizando os eventos de mutação que afetam grandes porções do genoma. De modo geral, a distância entre dois genomas é calculada considerando que o custo de cada operação é unitário. Este trabalho aborda a ordenação de permutações sem sinais e considera que o custo de cada operação é o número de elementos presentes na região afetada. Realizamos a comparação dos custos médios fornecidos por algumas das heurísticas conhecidas na área, e apresentamos os resultados obtidos ao modificar algumas delas para considerar o problema com distância ponderada. Com isso, desenvolvemos uma heurística a partir da combinação daquelas que obtiveram os melhores resultados. Esta heurística apresentou custos médios menores que o das outras analisadas neste trabalho. Além disso, realizamos uma análise dos custos médios obtidos pela heurística de Grafo de Ciclos, uma heurística que obteve os melhores resultados quando os custos das operações são unitários.

1 Introdução

As alterações mais comuns que ocorrem no genoma se dão através de mutações pontuais, onde nucleotídeos podem ser substituídos, inseridos ou removidos de um genoma. Na área de rearranjo de genomas, diferentemente das mutações pontuais, o interesse está focado principalmente em mutações que afetam grandes porções do genoma. Reversões e transposições são dois exemplos de eventos comuns de rearranjo. Uma reversão inverte um trecho do genoma em relação à sua posição original, já a transposição troca de posição dois blocos adjacentes de elementos.

Um problema de considerável importância tratado em biologia computacional consiste em encontrar a menor distância evolucionária entre dois genomas, que pode ser estimada calculando a distância de rearranjo entre eles. De modo geral, a distância de rearranjo é o tamanho da menor sequência de eventos que transforma um genoma no outro. Esta motivação baseia-se no princípio da máxima parcimônia, pois a menor distância de rearranjo entre dois genomas representa uma boa estimativa de sua distância evolucionária.

*Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP.
†Faculdade de Tecnologia, Universidade Estadual de Campinas, Campinas, SP.

Outra abordagem para tratar este problema é considerar que a distância de rearranjo é a soma dos tamanhos dos trechos afetados pelos eventos de rearranjo. Esta abordagem é biologicamente justificável, pois um rearranjo de maior tamanho irá causar maior perturbação nas características do organismo, tornando menos provável sua sobrevivência e, portanto, a transmissão desta mutação [2].

Uma operação de rearranjo que afeta um trecho de tamanho ℓ tem custo dado pela função $f(\ell) = \ell^\alpha$, para $\alpha \geq 0$. Muitos estudos na área de rearranjo de genomas consideram $\alpha = 0$, ou seja, que o custo independe do tamanho do trecho afetado pela operação de rearranjo. Este projeto considera o problema onde $\alpha = 1$, e operações de reversão e transposição são permitidas. Este problema é conhecido como Problema da Ordenação Ponderada por Reversões e Transposições.

2 Definições

Um genoma pode ser representado através de uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, onde cada elemento π_i da permutação π é considerado um bloco conservado comum aos dois genomas. Um bloco conservado representa um ou mais genes.

Dada uma permutação π com sinais, temos $\pi_i \in \{-n, -(n-1), \ldots, -1, 1, \ldots, n-1, n\}$ para $1 \leq i \leq n$ e $|\pi_i| \neq |\pi_j|$ para todo $i \neq j$. Neste caso, o sinal indica a orientação do gene. Caso se trate de uma permutação sem sinais, temos que $\pi_i \in \{1, 2, \ldots, n-1, n\}$ para $1 \leq i \leq n$ e $\pi_i \neq \pi_j$ para todo $i \neq j$. Este trabalho analisou o resultado de algoritmos que ordenam apenas permutações sem sinais.

A permutação identidade ι é a permutação ordenada com n elementos, tal que $\iota = (1\ 2\ \ldots\ n)$. A permutação estendida é definida como a permutação obtida a partir de π adicionando dois elementos $\pi_0 = 0$ e $\pi_{n+1} = n + 1$. A permutação estendida também é representada por π.

2.1 Operações de rearranjo

Uma reversão com sinais $\rho(i, j)$, para $1 \leq i \leq j \leq n$, irá inverter a ordem dos elementos e os sinais do bloco $\pi[i \ldots j]$ da permutação com sinais π. Ou seja, esta reversão transforma $\pi = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \ldots\ \pi_j}\ \pi_{j+1}\ \ldots\ \pi_n)$ em

$$\pi \cdot \rho(i, j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{-\pi_j\ \ldots\ -\pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$$

Uma reversão sem sinais $\rho(i, j)$, para $1 \leq i < j \leq n$, irá inverter a ordem dos elementos do bloco $\pi[i \ldots j]$ da permutação sem sinais π. Desta forma, esta reversão transforma $\pi = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \ldots\ \pi_j}\ \pi_{j+1}\ \ldots\ \pi_n)$ em

$$\pi \cdot \rho(i, j) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \ldots\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$$

Uma transposição $\rho(i, j, k)$ para $1 \leq i < j < k \leq n + 1$ troca os blocos consecutivos $\pi[i \ldots j-1]$ e $\pi[j \ldots k-1]$ de lugar. Ou seja, transforma a permutação $\pi = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}}\ \underline{\pi_j\ \pi_{j+1}\ \ldots\ \pi_{k-1}}\ \pi_k\ \ldots\ \pi_n)$ em

$$\pi \cdot \rho(i,j,k) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \pi_{j+1} \ \ldots \ \pi_{k-1}} \ \underline{\pi_i \ \pi_{i+1} \ \ldots \ \pi_{j-1}} \ \pi_k \ \ldots \ \pi_n)$$

Por exemplo, dada a permutação $\pi = (0\ 2\ 3\ 1\ 5\ 4\ 6)$ e uma reversão sem sinal $\rho(1,3)$, temos $\pi \cdot \rho(1,3) = (0\ 1\ 3\ 2\ 5\ 4\ 6)$. Caso seja aplicada uma transposição $\rho(1,3,4)$, temos $\pi \cdot \rho(1,3,4) = (0\ 1\ 2\ 3\ 5\ 4\ 6)$. Considerando a permutação com sinais $\pi = (+0\ +2\ +3\ -1\ +5\ +4\ +6)$, então uma reversão com sinais $\rho(1,3)$ resulta em $\pi \cdot \rho(1,3) = (+0\ +1\ -3\ -2\ +5\ +4\ +6)$.

Dadas duas permutações π e σ, a Distância de Reversão $d_r(\pi,\sigma)$ é o número mínimo de reversões que transformam π em σ. Ou seja, dada uma sequência mínima de reversões tal que $\pi \cdot \rho_1 \cdot \rho_2 \ldots \rho_r = \sigma$, então $d_r(\pi,\sigma) = r$. A Distância de Transposição $d_t(\pi,\sigma)$ é o número mínimo de transposições que transformam π em σ. De forma semelhante, a Distância de Reversão e Transposição $d_{rt}(\pi,\sigma)$ é o número mínimo de reversões e transposições que transformam π em σ.

Considerando o problema de ordenação com distância ponderada, o custo de uma reversão $\rho(i,j)$ é dado por $custo_{lwR}(\rho(i,j)) = j - i + 1$, enquanto que o custo de uma transposição $\rho(i,j,k)$ é dado por $custo_{lwT}(\rho(i,j,k)) = k - i$. Dada uma permutação π, o problema da ordenação ponderada consiste em encontrar uma sequência de operações que ordene π, tal que a somatória dos seus custos seja a menor possível.

2.2 Breakpoints

Para uma permutação estendida π sem sinais, um par de elementos π_i e π_{i+1}, para $0 \leq i \leq n$, é um breakpoint se $|\pi_{i+1} - \pi_i| \neq 1$. Caso contrário, é dito ser uma adjacência. Para uma permutação com sinais, um par de elementos π_i e π_{i+1} com $0 \leq i \leq n$ é um breakpoint se $\pi_{i+1} - \pi_i \neq 1$. Um breakpoint pode ser indicado na permutação utilizando o símbolo "$\bullet$" entre os elementos π_i e π_{i+1}.

Dada uma permutação π, o número de breakpoints em π é dado por $b(\pi)$. Note que a permutação identidade ι é a única onde $b(\iota) = 0$. Além disso, para uma operação qualquer ρ, denotamos por $\Delta_{bp}(\pi,\rho) = b(\pi \cdot \rho) - b(\pi)$ a variação no número de breakpoints em razão da aplicação de ρ em π.

Por exemplo, para uma permutação sem sinais $\pi = (0\ 2\ 3\ 1\ 5\ 4\ 6)$, existem breakpoints entre os pares de elementos (0 2), (3 1), (1 5) e (4 6). Utilizando o símbolo de breakpoints, a permutação de exemplo pode ser representada como $\pi = (0 \bullet 2\ 3 \bullet 1 \bullet 5\ 4 \bullet 6)$. Neste caso, temos $b(\pi) = 4$.

2.3 Strips

Uma *strip* em π é um trecho maximal de elementos $\pi[i \ldots j]$, tal que todos os pares de elementos π_k e π_{k+1}, para $i \leq k < j$, sejam adjacências. Se a sequência $\pi_i \ldots \pi_j$ é crescente, a strip é dita crescente. Da mesma forma, se a sequência $\pi_i \ldots \pi_j$ é decrescente, a strip é dita decrescente. Uma strip unitária é decrescente por definição, com exceção dos elementos π_0 e π_{n+1} que são sempre crescentes. Por exemplo, para $\pi = (0 \bullet 2\ 3\ 4 \bullet 1 \bullet 6\ 5 \bullet 7\ 8)$, as strips (2 3 4) e (7 8) são crescentes, enquanto que (6 5) é decrescente. A strip (0) é unitária e crescente e a strip (1) é unitária e decrescente.

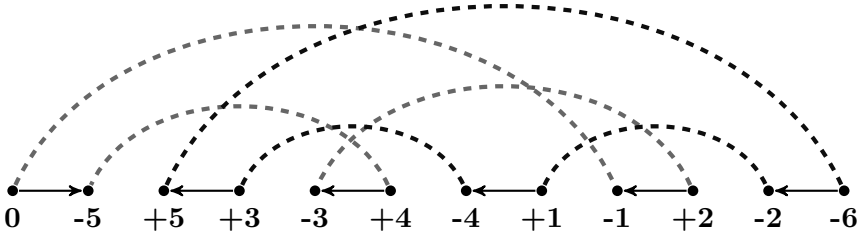

Figura 1: Grafo de ciclos para $\pi = (+5\ -3\ -4\ -1\ -2)$.

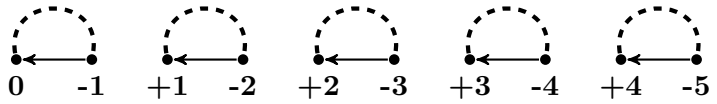

Figura 2: Grafo de ciclos para a permutação identidade $\iota = (1\ 2\ 3\ 4)$, de tamanho $n = 4$. Note que existem n+1 ciclos alternantes ímpares no grafo de ciclos.

2.4 Grafo de Ciclos

O grafo de ciclos de uma permutação π é definido como $G(\pi) = (V, E_{black} \cup E_{gray})$. O conjunto de vértices V, o conjunto de arestas pretas E_{black} e o conjunto de arestas cinzas E_{gray} são definidos da seguinte forma:

- $V = \{0, -\pi_1, +\pi_1, -\pi_2, +\pi_2, \ldots, -\pi_n, +\pi_n, -(n+1)\}$.
- $E_{black} = \{(-\pi_i, +\pi_{i-1}),$ para todo $1 \leq i \leq n+1\}$.
- $E_{gray} = \{(+(i-1), -i),$ para todo $1 \leq i \leq n+1\}$.

Para uma permutação π com sinais, a decomposição em ciclos é única. Por exemplo, para a permutação $\pi = (+5\ -3\ -4\ -1\ -2)$, o grafo de ciclos correspondente é apresentado na Figura 1. Note nesta figura que os vértices $-\pi_2 = +3$ e $+\pi_1 = +5$ estão conectados por uma aresta preta (referente a $i = 2$), assim como os vértices $-\pi_3 = +4$ e $+\pi_2 = -3$ (referente a $i = 3$). Por outro lado, note que os elementos $+5$ e -6 estão conectados por uma aresta cinza (referente a $i = 6$), assim como os elementos $+3$ e -4 (referente a $i = 4$).

Um ciclo alternante é dito *ímpar* se o número de arestas pretas for ímpar. De modo semelhante, um ciclo alternante é dito *par* se o número de arestas pretas for par. Seja $c(\pi)$ o número de ciclos alternantes de $G(\pi)$. A permutação identidade é a única que possui n+1 ciclos alternados ímpares. A Figura 2 apresenta o grafo de ciclos para a permutação identidade ι de tamanho $n = 4$.

Para uma permutação π sem sinais, pode-se associar sinais aos elementos de forma arbitrária, de modo a obter um grafo de ciclos. Neste caso, encontrar a decomposição máxima no número de ciclos alternantes é um problema NP-Difícil [9]. O melhor fator de aproximação conhecido é de $1.4167 + \epsilon$, para $\epsilon > 0$, apresentado por Chen [19]. Por

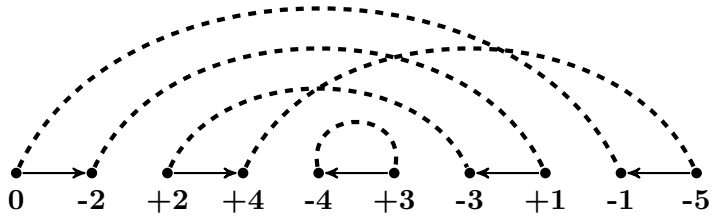

Figura 3: Decomposição em ciclos que não é máxima, onde a atribuição de sinais para $\pi = (2\ \ 4\ \ 3\ \ 1)$ é $\hat{\pi} = (+2\ \ -4\ \ -3\ \ -1)$. Neste caso, temos $c(\hat{\pi}) = 2$.

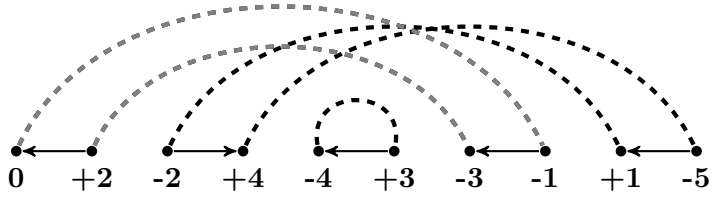

Figura 4: Decomposição máxima no número de ciclos, onde a atribuição de sinais para $\pi = (2\ \ 4\ \ 3\ \ 1)$ é $\hat{\pi} = (-2\ \ -4\ \ -3\ \ +1)$. Neste caso, temos $c(\hat{\pi}) = 3$.

exemplo, para uma permutação $\pi = (2\ 4\ 3\ 1)$, a Figura 3 mostra uma decomposição em ciclos alternantes que não é máxima, onde a atribuição de sinais em π é $\hat{\pi} = (+2\ \ -4\ \ -3\ \ -1)$, com $c(\hat{\pi}) = 2$. Por outro lado, a Figura 4 mostra uma decomposição máxima no número de ciclos, onde a atribuição de sinais em π é $\hat{\pi} = (-2\ \ -4\ \ -3\ \ +1)$, resultando em $c(\hat{\pi}) = 3$.

2.5 Trabalhos anteriores

O primeiro algoritmo polinomial para o Problema de Ordenação por Reversões com Sinais foi apresentado por Hannenhalli e Pevzner [4], com complexidade de tempo de $\mathcal{O}(n^4)$. Trabalhos posteriores simplificaram essa solução e aprimoraram sua eficiência [5]. Atualmente, o algoritmo mais eficiente possui complexidade de $\mathcal{O}(n^{3/2}\sqrt{\lg n})$ [6]. Quando não é necessário listar as reversões utilizadas na ordenação, há um algoritmo com complexidade $\mathcal{O}(n)$ que apenas calcula a distância de reversão [7].

Quando a orientação dos genes não é conhecida, temos o Problema de Ordenação por Reversões Sem Sinais. Neste caso, Caprara [8] provou que obter a distância de reversão entre dois genomas é um problema NP-Difícil. O melhor algoritmo de aproximação até o momento, possui fator de aproximação de 1.375, apresentado por Berman, Hannenhalli e Karpinski em 2002 [10].

Quando apenas transposições são permitidas, temos o Problema de Ordenação por Transposições. Neste caso, obter a distância de transposição foi provado ser um problema NP-Difícil por Bulteau, Fertin e Rusu em 2011 [11]. O primeiro algoritmo para este problema foi desenvolvido por Bafna e Pevzner, possuindo um fator de aproximação 1.5 e

complexidade $\mathcal{O}(n^2)$ [12]. Implementações posteriores ofereceram alternativas simplificadas para essa solução [13, 14]. Posteriormente, Elias e Hartman [15] implementaram uma solução com fator de aproximação de 1.375 e complexidade $\mathcal{O}(n^2)$.

Quando tanto a operação de reversão quanto a operação de transposição são permitidas, temos o Problema de Ordenação por Reversões e Transposições. A complexidade deste problema permanece desconhecida para ambas as versões deste problema, com sinais e sem sinais. Walter, Dias e Meidanis [16] apresentaram um algoritmo com fator de aproximação 2 quando a orientação dos genes é conhecida, e um algoritmo com fator de aproximação 3 quando a orientação dos genes não é conhecida. Para permutações sem sinais, Rahman, Shatabda e Hasan [18] apresentaram um algoritmo $2k$-aproximado, sendo k o fator de aproximação do algoritmo para o problema de Máxima Decomposição em Ciclos. Atualmente, o melhor valor de k é $1.4167 + \epsilon$ [19], portanto, existe um algoritmo com fator de aproximação $2.8334 + \epsilon$, para $\epsilon > 0$.

Quando apenas reversões são permitidas, e seu custo depende do tamanho da região afetada, temos o Problema da Ordenação Ponderada por Reversões. Neste caso, o custo da reversão é dada por $f(\ell) = \ell^\alpha$, para $\alpha \geq 0$, onde ℓ é o tamanho da região afetada. Quando a orientação dos genes não é conhecida, Pinter e Skiena [2] desenvolveram um algoritmo com fator de aproximação de $\mathcal{O}(\lg^2 n)$ para $\alpha = 1$. Bender e coautores [3] propuseram diversos algoritmos de aproximação para vários valores de $\alpha \geq 0$. Em específico, quando $\alpha = 1$, o algoritmo garante um fator de aproximação de $\mathcal{O}(\lg n)$. Quando a orientação dos genes é conhecida, também para $\alpha = 1$, Swidan e coautores [17] garantem fator de aproximação de $\mathcal{O}(\lg n)$.

O problema onde transposições ponderadas são permitidas, com ou sem reversões ponderadas, não possui resultados conhecidos.

3 Heurísticas

Nesta seção serão definidas algumas das heurísticas analisadas neste projeto para o problema da ordenação por reversões e transposições não ponderadas. Na próxima seção, algumas dessas heurísticas serão adaptadas para lidar com operações ponderadas.

As heurísticas aqui apresentadas utilizam métricas clássicas em problemas de rearranjo, sendo que cada uma utiliza um método diferente para calcular a pontuação de uma permutação π. O propósito de uma pontuação é fornecer um critério de escolha de operações. Ou seja, a cada passo da ordenação, queremos aplicar a operação que transforma a permutação atual em uma permutação com a melhor pontuação possível. O objetivo de uma métrica é obter a menor sequência de operações possível que ordena uma permutação. Definições mais detalhadas das heurísticas apresentadas abaixo podem ser encontradas na dissertação de mestrado de Andre Rodrigues Oliveira [1].

3.1 Código Esquerdo e Direito

Esta heurística é baseada em uma métrica desenvolvida no trabalho de Gagné e Hamel [20], que propuseram um algoritmo de 3-aproximação para o problema de ordenação por transposições. Dada uma permutação π, esta métrica computa dois códigos para cada elemento

desta permutação. O *código esquerdo* de um elemento π_i, denotado por $lc(\pi_i)$, indica quantos elementos à esquerda de π_i são maiores que ele. De forma semelhante, o *código direito* de um elemento π_i, denotado por $rc(\pi_i)$, indica quantos elementos à direita de π_i são menores que ele. Também define-se os códigos esquerdo e direito para uma dada permutação. O código esquerdo de uma permutação π é formada pela sequência dos códigos esquerdos de cada um de seus elementos. O mesmo ocorre para o código direito da permutação.

Por exemplo, dada uma permutação $\alpha = (6\ 3\ 2\ 1\ 4\ 5)$, o código esquerdo do quinto elemento $\alpha_5 = 4$ vale $lc(\alpha_5) = 1$, uma vez que apenas um elemento, $\alpha_1 = 6$, está à sua esquerda e é maior que 4. De maneira análoga, o código direito do segundo elemento $\alpha_2 = 3$ vale $rc(\alpha_2) = 2$, uma vez que dois elementos, $\alpha_4 = 1$ e $\alpha_3 = 2$, estão à sua direita e são menores que 3. Calculando o código esquerdo e direito para todos os elementos da permutação α, obtemos $lc(\alpha)$ e $rc(\alpha)$, conforme apresentado abaixo:

$$\begin{aligned} \alpha &= (6\ 3\ 2\ 1\ 4\ 5) \\ lc(\alpha) &= (0\ 1\ 2\ 3\ 1\ 1) \\ rc(\alpha) &= (5\ 2\ 1\ 0\ 0\ 0) \end{aligned}$$

A partir destas definições, define-se um *platô* como a sequência maximal de elementos de um código, tal que tenham mesmo valor e sejam não-nulos. O número de platôs de um código é dado por $pl(lc(\pi))$ ou $pl(rc(\pi))$, dependendo do código utilizado. Finalmente, o platô de uma permutação, denotado por $pl(\pi)$, é mínimo entre $pl(lc(\pi))$ e $pl(rc(\pi))$. A pontuação de uma permutação π é dada pelo seu platô, $pl(\pi)$.

Por exemplo, utilizando novamente a permutação α, temos $pl(lc(\alpha)) = 4$, uma vez que este código possui quatro platôs, (1), (2), (3) e (1, 1). De forma semelhante, $pl(rc(\alpha)) = 3$, uma vez que este código possui três platôs, (5), (2) e (1). Assim, $pl(\alpha) = 3$, uma vez que 3 é o mínimo entre $pl(lc(\alpha))$ e $pl(rc(\alpha))$.

O objetivo desta heurística é diminuir, a cada passo, o número de platôs de uma permutação qualquer π, uma vez que somente a permutação identidade ι possui $pl(\iota) = 0$. É possível demonstrar que sempre existe uma transposição que remove um platô, sem criar outros, se $\pi \neq \iota$. Desta forma, esta métrica permite ordenar uma permutação até que $\pi = \iota$. A complexidade de tempo total desta heurística é $\mathcal{O}(n^6)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n^2)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.2 Breakpoints

Esta heurística utiliza o conceito de breakpoints para ordenar uma permutação. Dada uma permutação, as operações aplicadas visam diminuir o número de breakpoints, visto que a permutação identidade ι é a única sem breakpoints ($b(\iota) = 0$). A pontuação de uma permutação π é dada pelo seu número de breakpoints $b(\pi)$. A cada passo, a heurística de Breakpoints inicialmente procura uma transposição que remova 3 breakpoints. Se esta operação não existir, procura uma operação qualquer que remova 2 breakpoints. Caso não exista uma operação que remova 2 ou 3 breakpoints, então uma operação que remova 1 breakpoint será aplicada. É possível demonstrar que sempre existe uma operação que

remove pelo menos um breakpoint se $\pi \neq \iota$. Assim sendo, a heurística garante que sempre encontrará uma sequência de operações que ordene uma permutação π qualquer.

Por exemplo, para uma permutação sem sinais $\pi = (0\ 1 \bullet 3 \bullet 5 \bullet 2 \bullet 4 \bullet 6)$, com $b(\pi) = 5$, a heurística procura uma operação que remova 3 breakpoints. Neste caso, a transposição $\rho(2, 4, 5)$ remove esta quantidade de breakpoints, visto que $\pi \cdot \rho(2, 4, 5) = (0\ 1\ 2\ 3\ \bullet 5\ 4\ \bullet 6)$ e $b(\pi \cdot \rho(2, 4, 5)) = 2$. Temos aqui que $\Delta_{bp}(\pi, \rho(2, 4, 5)) = 5 - 2 = 3$. Na próxima iteração, com $\pi \leftarrow \pi \cdot \rho(2, 4, 5)$, não é possível encontrar uma operação que remova 3 breakpoints. Em seguida, a heurística procura uma operação que remova 2 breakpoints, encontrando a reversão $\rho(4, 5)$. Esta reversão completa a ordenação, pois $\pi \cdot \rho(4, 5) = \iota = (0\ 1\ 2\ 3\ 4\ 5\ 6)$.

Esta heurística é uma 3-aproximação, pois é possível remover, a cada passo, até 3 breakpoints com uma transposição. A complexidade de tempo é $\mathcal{O}(n^2)$, pois são $\mathcal{O}(n)$ passos para ordenar a permutação, enquanto que encontrar a operação leva $\mathcal{O}(n)$ com o auxílio de um vetor que armazena as posições dos elementos da permutação.

3.3 LIS

O Problema da Subsequência Crescente Máxima (*LIS*, ou *Longest Increasing Subsequence*), busca encontrar um subsequência crescente de elementos tal que seu tamanho seja o maior possível. A subsequência crescente máxima de uma permutação π é representada por $LIS(\pi)$, e seu tamanho, por $|LIS(\pi)|$. A heurística LIS utilizada para o problema de rearranjo de genomas aplica reversões e transposições na permutação tal que sua subsequência crescente máxima sempre aumente de tamanho, uma vez que a permutação identidade possui a maior subsequência crescente possível. A pontuação de uma permutação π é dada pelo tamanho de sua LIS. Pode-se provar que sempre existe uma operação que aumenta sua pontuação, $|LIS(\pi)|$, em uma unidade, logo, sempre é possível encontrar uma sequência de operações que ordene uma permutação π qualquer.

Por exemplo, para uma permutação $\pi = (1\ 3\ 6\ 2\ 4\ 5)$, sua subsequência crescente máxima é $LIS(\pi) = (1\ 2\ 4\ 5)$, com $|LIS(\pi)| = 4$. Se aplicarmos a operação $\rho(2, 3, 5)$ em π, teremos $\pi \cdot \rho(2, 3, 5) = (1\ 6\ 2\ 3\ 4\ 5)$ aumentando sua subsequência crescente máxima para $LIS(\pi \cdot \rho(2, 3, 5)) = (1\ 2\ 3\ 4\ 5)$, com $|LIS(\pi)| = 5$.

Esta heurística possui complexidade de tempo $\mathcal{O}(n^5 \log n)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n \log n)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.4 LIS+LDS

Esta heurística combina LIS com a Subsequência Decrescente Máxima (*LDS* ou *Longest Decreasing Subsequence*). Por exemplo, para uma sequência $A = (8\ 3\ 7\ 4\ 2\ 6\ 5\ 1)$, sua subsequência decrescente máxima, ou seja, a de maior tamanho, é $B = (8\ 7\ 4\ 2\ 1)$. A motivação para considerar a métrica LDS é que, caso $LDS(\pi)$ seja maior que $LIS(\pi)$, é mais interessante aumentar $LDS(\pi)$ e posteriormente transformá-la em uma LIS aplicando uma reversão. A pontuação de uma permutação π é dada pelo maior valor dentre $|LIS(\pi)|$ e $|LDS(\pi)|$.

Por exemplo, dada uma permutação $\pi = (4\ 5\ 7\ 6\ 3\ 2\ 1)$, aplica-se a operação ρ tal

que a pontuação de $\pi \cdot \rho$ seja máxima. Neste caso, a operação $\rho(6, 8, 10)$ resulta na maior $LDS(\pi \cdot \rho)$ possível, enquanto que a operação $\rho(1, 7)$ resulta na maior $LIS(\pi \cdot \rho)$ possível.

$$\begin{aligned} \pi &= (7\ 9\ 6\ 8\ 5\ 2\ 1\ 4\ 3) \\ \pi \cdot \rho(1,7) &= (1\ 2\ 5\ 8\ 6\ 9\ 7\ 4\ 3) \qquad LIS(\pi \cdot \rho(1,7)) = (1\ 2\ 5\ 6\ 7) \\ \pi \cdot \rho(6,8,10) &= (7\ 9\ 6\ 8\ 5\ 4\ 3\ 2\ 1) \qquad LDS(\pi \cdot \rho(6,8,10)) = (7\ 6\ 5\ 4\ 3\ 2\ 1) \end{aligned}$$

Como $|LDS(\pi \cdot \rho(6, 8, 10))|$ é maior que $|LIS(\pi \cdot \rho(1, 7))|$, a operação escolhida será $\rho(6, 8, 10)$.

Esta heurística possui a mesma complexidade de tempo da heurística LIS, $\mathcal{O}(n^5 \log n)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n \log n)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.5 LIS+BP

A heurística LIS+BP modifica a heurística LIS de forma que as operações evitem aumentar o número de breakpoints. A pontuação de uma permutação π é dada por $|LIS(\pi)| - b(\pi)$. Por exemplo, dada a permutação estendida $\pi = (0\ 3\ 2\ 5\ 4\ 1\ 6)$, temos $LIS(\pi) = (2\ 4\ 6)$.

$$\begin{aligned} \pi &= (0\ \bullet\ 3\ 2\ \bullet\ 5\ 4\ \bullet\ 1\ \bullet\ 6) \\ \pi \cdot \rho(3,4,6) &= (0\ \bullet\ 3\ 2\ \bullet\ 4\ \bullet\ 1\ \bullet\ 5\ 6) \qquad LIS(\pi \cdot \rho(3,4,6)) = (2\ 4\ 5\ 6) \\ \pi \cdot \rho(3,5) &= (0\ \bullet\ 3\ 2\ 1\ \bullet\ 4\ 5\ 6) \qquad LIS(\pi \cdot \rho(3,5)) = (1\ 4\ 5\ 6) \end{aligned}$$

De acordo com a heurística LIS, a operação $\rho(3, 4, 6)$ possui a maior pontuação possível. Caso esta operação seja aplicada em π, sua LIS aumentará em um elemento. Por outro lado, a operação $\rho(3, 5)$ possui a maior pontuação de acordo com a heurística LIS+BP e também aumentará a LIS de π em um elemento. Porém, esta operação remove 2 breakpoints, ao contrário do que ocorre com a operação escolhida através da heurística LIS, que não remove nenhum breakpoint.

Esta heurística permanece com a mesma complexidade de tempo da heurística LIS, $\mathcal{O}(n^5 \log n)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n \log n)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.6 Entropia

Esta heurística utiliza o nível de desordem dos elementos da permutação, ou seja, a distância que cada elemento está de sua posição desejada (em relação à identidade). O cálculo da entropia para uma permutação π sem sinais é obtido da seguinte forma:

$$entro(\pi) = \sum_{i=1}^{n} |\pi_i - i|$$

Ou seja, para uma permutação π sem sinais, calculamos a distância $|\pi_i - i|$ de cada um de seus elementos e somamos estes valores para obter a entropia. Por exemplo, para uma permutação $\pi = (1\ 4\ 3\ 5\ 2\ 6)$, sua entropia pode ser calculada da seguinte forma:

$$\pi = (1\ 4\ 3\ 5\ 2\ 6)$$
$$|\pi_i - i| = (0\ 2\ 0\ 1\ 3\ 0)$$
$$entro(\pi) = 2 + 1 + 3 = 6$$

Dado que apenas a permutação identidade possui entropia nula ($entro(\iota) = 0$), o objetivo desta heurística é aplicar operações que diminuam sua entropia a cada passo. É possível demonstrar que sempre existe uma operação que, aplicada em uma permutação $\pi \neq \iota$, irá diminuir o valor de sua entropia. Portanto, é certo que esta heurística encontrará uma sequência de operações que ordene uma permutação π qualquer. A heurística de Entropia possui complexidade de tempo $\mathcal{O}(n^5)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.7 Inversões

A heurística de Inversões analisa, par a par, a posição dos elementos de uma permutação à procura dos pares que estejam com suas posições invertidas. Para uma permutação π sem sinais, e um par de elementos π_i e π_j, existe uma inversão se $\pi_i > \pi_j$ e $i < j$. Ou seja, se o elemento à esquerda possui um valor maior que o elemento à direita. Denota-se por $inv(\pi)$ a soma das inversões em uma permutação.

Por exemplo, dada a permutação $\pi = (2\ 4\ 5\ 1\ 3)$, temos o seguinte valor para o número de inversões de π:

$$\pi = (2\ 4\ 5\ 1\ 3)$$
$$inv(\pi) = (1\ 2\ 2\ 0\ 0) = 5$$

Neste exemplo, temos $inv(\pi) = 5$, resultado da soma dos números de inversões de cada elemento individualmente. Pode-se observar também, por exemplo, que $inv(2) = 1$, pois o elemento 1 está à direita do elemento 2, e $inv(4) = 2$, pois os elementos 1 e 3 estão à direita de 4.

Dada uma permutação π de tamanho n e uma operação ρ, denotamos a variação no número de inversões por $\Delta_{inv}(\pi, \rho) = inv(\pi \cdot \rho) - inv(\pi)$. A maior variação possível no número de inversões ocorre quando uma reversão é aplicada em uma permutação invertida $(n\ \ldots 2\ 1)$, ordenando-a. Neste caso, a variação no número de inversões será de $n(n-1)/2$, uma vez que para cada elemento da permutação, todos à sua direita estão na posição errada. Por exemplo, dada uma permutação $\pi = (5\ 4\ 3\ 2\ 1)$ com $n = 5$, então $inv(\pi) = n(n-1)/2 = 10$. Se aplicarmos $\rho(1, 5)$ em π, teremos $inv(\pi \cdot \rho) = inv(\iota) = 0$. Finalmente, obtemos que $\Delta_{inv}(\pi, \rho) = 0 - n(n-1)/2 = -10$

Dado que apenas a permutação identidade possui valor nulo para o número de inversões ($inv(\iota) = 0$), esta heurística tem como objetivo diminuir, a cada passo, o número de inversões de uma permutação π. Isto é possível, pois, dada uma permutação $\pi \neq \iota$, sempre existe uma operação que diminui o número de inversões em pelo menos uma unidade com uma transposição.

A demonstração é dada, resumidamente, da seguinte forma. Dada uma permutação $\pi \neq \iota$, aplica-se uma transposição para levar o menor elemento a que esteja na posição

errada para sua posição correta π_a. Sem dúvida, todos os elementos à esquerda de a que estejam na posição errada terão o número de inversões diminuído em uma unidade. Continuando com a permutação de exemplo anterior, $\pi = (2\ 4\ 5\ 1\ 3)$, o menor elemento na posição errada é $a = 1$. Aplicando a transposição $\rho(1, 4, 5)$, o elemento $a = 1$ será levado à sua posição correta. Teremos os seguintes números de inversões:

$$\pi \cdot \rho(1, 4, 5) = (1\ 2\ 4\ 5\ 3)$$
$$inv(\pi \cdot \rho(1, 4, 5)) = (0\ 0\ 1\ 1\ 0) = 2$$

Pode-se observar que $inv(\pi \cdot \rho(1, 4, 5)) - inv(\pi) = 2 - 5 = -3$, ou seja, o número de inversões caiu em 3 unidades. Portanto, sempre é possível encontrar uma sequência de operações que ordenam π através da heurística de Inversões.

Nesta heurística, pode-se encontrar o número de inversões em $\mathcal{O}(n \log n)$ utilizando o método de divisão e conquista do algoritmo MergeSort. A complexidade de tempo total é $\mathcal{O}(n^5 \log n)$. São $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, custo de $\mathcal{O}(n \log n)$ para calcular a pontuação e $\mathcal{O}(n)$ passos para ordenar a permutação.

3.8 Heurística de Grafo de Ciclos

Esta heurística ordena permutações por reversões e transposições com base no grafo de ciclos da permutação. A decomposição em ciclos de permutações com sinais é única, o que não acontece para permutações sem sinais. Ou seja, uma permutação sem sinais pode apresentar várias decomposições em ciclos. Como encontrar uma decomposição em ciclos de permutações sem sinais é um problema NP-Difícil [9], esta heurística utiliza uma decomposição aproximada com fator de 1.4193, apresentada por Lin e Jiang [21].

A ordenação ocorre em três etapas. Na primeira delas, procura-se por uma transposição que remova 3 breakpoints ou, caso não seja possível, uma operação que remova 2 breakpoints. Na segunda etapa, a heurística busca por operações que aumentem o número de ciclos ímpares do grafo. Caso isto não seja possível, a heurística tenta aumentar o número de ciclos com a garantia de que as próximas duas operações irão aumentar o número de ciclos ímpares, garantindo assim um fator de aproximação de 1.5. Em último caso, na terceira etapa, a heurística procura uma sequência de operações em quatro bancos de configurações, sendo que três foram criados para uso desta heurística e o último apresentado por Elias e Hartman [15]. Os quatro bancos em conjunto possuem garantia de aproximação máxima de 2.0.

4 Resultados

Os testes realizados utilizaram permutações de tamanho variando entre 10 e 100, em múltiplos de 5, onde cada tamanho possui 10000 permutações a ordenar. Cada uma delas foi obtida aplicando 10 reversões e 10 transposições em ordem aleatória a partir da permutação identidade. Dessa forma, todas podem ser ordenadas com no máximo 20 operações.

Primeiramente, analisamos os custos ponderados das heurísticas que não levam em conta o tamanho da operação. Estes resultados são apresentados na Figura 5. Como pode ser

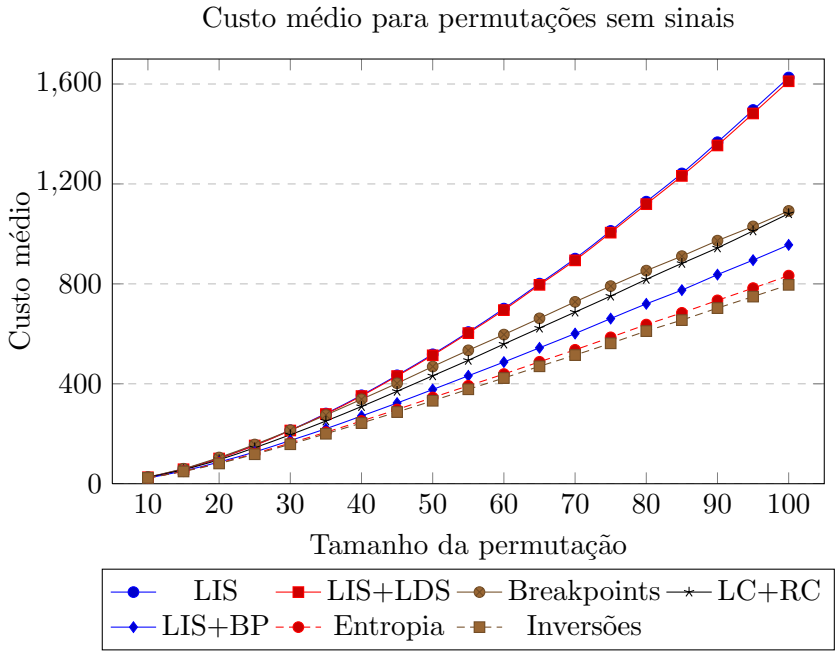

Figura 5: Custo médio das heurísticas básicas que não possuem modificações para a versão do problema com distância ponderada.

visto nos testes, as heurísticas LIS e LIS+LDS apresentaram os custos médios mais altos e que se distanciaram mais rapidamente dos custos médios das demais heurísticas conforme o tamanho da permutação aumenta. Note que, como esperado, LIS+LDS e LIS+BP possuem custos menores que a heurística LIS, já que utilizam uma métrica a mais em relação à heurística LIS. Em específico, é possível notar que LIS+BP possui um resultado significativamente melhor que LIS, apenas introduzindo em seu algoritmo a noção de *strip*, proveniente da heurística de Breakpoints. A heurística de Código Esquerdo e Direito, representada no gráficos por LC+RC, obteve um resultado próximo ao da heurística de Breakpoints. Das heurísticas apresentadas na Figura 5, Inversões obteve o melhor resultado e Entropia o segundo melhor, com custo médio próximo ao melhor.

Foram selecionadas três heurísticas e suas variantes que consideram o tamanho da operação (aqui representadas pelo sufixo W): Inversões, Breakpoints, Entropia, InversõesW, BreakpointsW e EntropiaW. Para obter a variante com sufixo W, dividimos o benefício de cada operação pelo tamanho da mesma. Por exemplo, para InversõesW, dividimos a variação do número de inversões pelo tamanho da operação e selecionamos a operação com o melhor custo-benefício.

Os custos médios dessas heurísticas são apresentados na Figura 6. A heurística de Inversões mostrou o melhor resultado tanto na sua versão original como na sua versão ponderada, em relação às heurísticas apresentadas no gráfico. Além disso, a heurística de Entropia, que havia ficado em segundo lugar, não se saiu tão bem na sua versão ponderada, EntropiaW. Neste caso, obteve um custo médio consideravelmente maior que a heurística

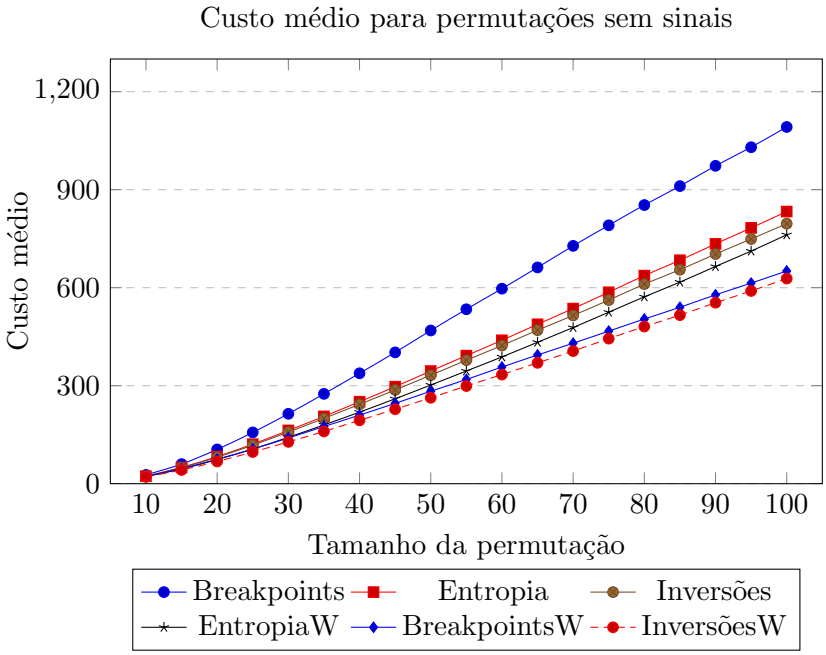

Figura 6: Comparativo das principais heurísticas básicas e suas respectivas versões ponderadas

BreakpointsW, que ficou em segundo lugar dentre as heurísticas selecionadas.

Com a intenção de obter custos médios ainda menores, combinamos as heurísticas ponderadas, InversõesW, BreakpointsW e EntropiaW. Para isto, criamos uma meta-heurística Min{Inv,BP,Ent}W que retorna o menor custo médio dentre estas três heurísticas. Isso se deve ao fato de que, apesar de InversõesW apresentar a melhor média, esta heurística não obtém o menor custo para todas as permutações. Desta forma, o menor custo que for obtido por qualquer uma das três heurísticas será o custo utilizado para compor o resultado de Min{Inv,BP,Ent}W. Este resultado é apresentado na Figura 7. Além disso, na Figura 8, podemos visualizar o quanto cada heurística contribuiu para obter o menor custo em Min{Inv,BP,Ent}W. Por exemplo, para permutações de tamanho 10, em torno de 40% dos custos obtidos por EntropiaW foram iguais ou menores que os outros dois algoritmos, enquanto que para permutações de tamanho 100, praticamente nenhum dos custos obtidos por esta heurística foram menores ou iguais a BreakpointsW ou InversõesW. Note que um empate entre duas heurísticas é tratado concedendo às duas o menor custo, o que explica o motivo da soma de porcentagens das heurísticas da Figura 8 não corresponder a 100%. Esta figura mostra que, apesar de InversõesW apresentar o menor custo na média, BreakpointsW ordena muitas permutações com custos menores. Além disso, é possível observar que EntropiaW não apresentou resultados competitivos em relação às duas outras.

Um outro tipo de heurística desenvolvida neste projeto, denotada por Mix{Inv,BP}W*, utiliza como caixa-preta os passos internos de ordenação das heurísticas InversõesW e BreakpointsW. Estes passos internos referem-se aos passos que determinam a operação de melhor

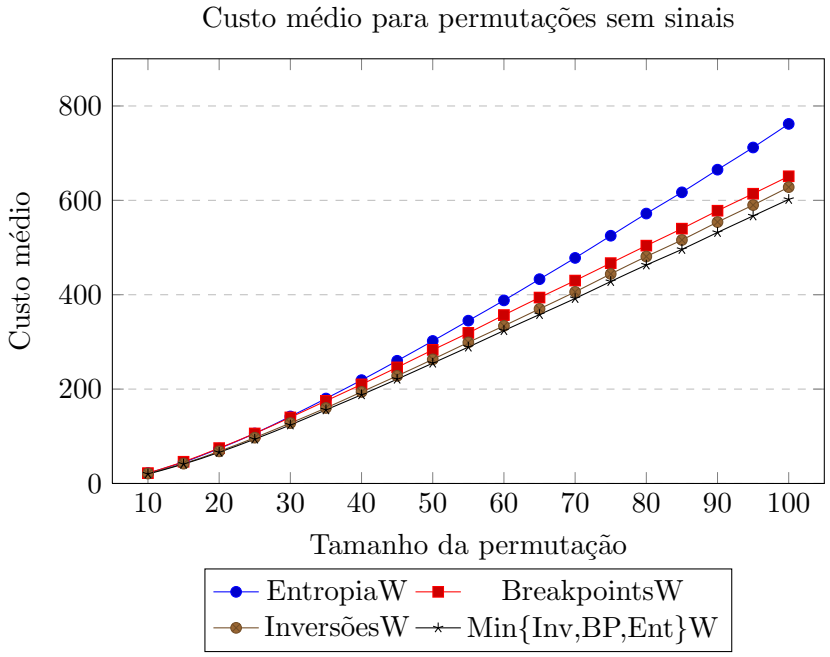

Figura 7: Mínimo obtido a partir dos resultados das heurísticas EntropiaW, BreakpointsW e InversõesW.

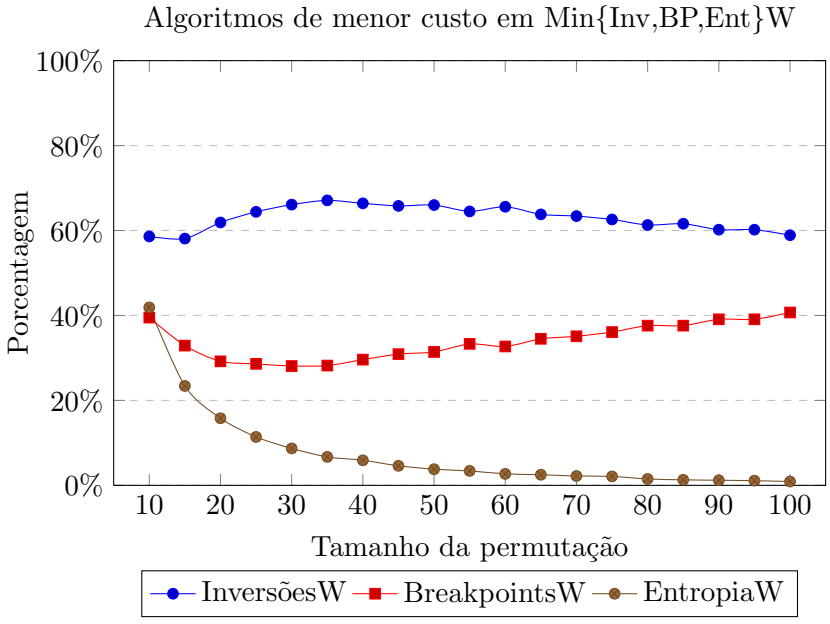

Figura 8: Porcentagem de permutações com a qual cada algoritmo obteve o menor custo.

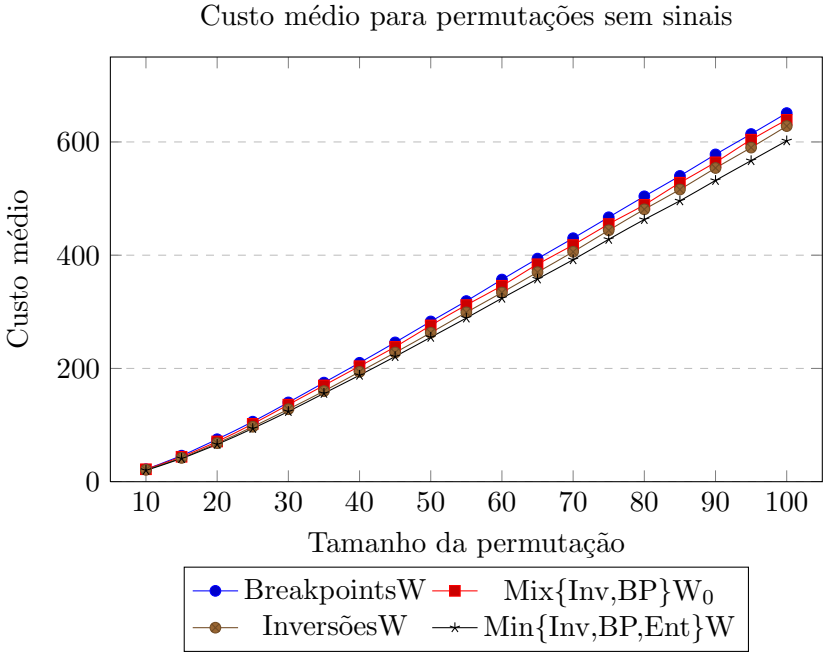

Figura 9: A cada iteração, Mix{Inv,BP}W_0 utiliza a operação de menor custo entre BreakpointsW e InversõesW.

benefício para uma determinada configuração da permutação. Dessa forma, a cada passo da ordenação, Mix{Inv,BP}W* obtém duas operações, uma gerada por BreakpointsW e outra por InversõesW. O pseudocódigo da heurística Mix{Inv,BP}W* é apresentado na Heurística 1. Cada uma das variações, W_0, W_1, W_2 e W_3, sobre esta heurística, utiliza uma métrica diferente para selecionar a operação de melhor benefício. A primeira delas, denotada por Mix{Inv,BP}W_0, escolhe a operação de menor custo dentre as duas operações geradas pelas caixas-pretas. Este resultado é apresentado na Figura 9. Esta abordagem não obteve bons resultados, pois a heurística de BreakpointsW tende a selecionar operações de menor tamanho em comparação com InversõesW. Isso não quer dizer, necessariamente, que o benefício é sempre melhor. Para poder definir qual possui o melhor benefício, é necessário utilizar outra métrica que torne estes valores comparáveis, uma vez que cada uma das heurísticas, InversõesW e BreakpointsW, utiliza uma escala diferente. Para a heurística BreakpointsW, o benefício de uma operação é dado por:

$$\text{benefício}_{BrW} = \frac{\text{número de breakpoints removidos}}{\text{custo da operação}}$$

Note que o numerador do benefício$_{BrW}$ pode variar no máximo em 3 breakpoints, que é o número de breakpoints que uma transposição pode remover ou adicionar. De maneira semelhante, para a heurística InversõesW, o benefício de uma operação é dado por:

$$\text{benefício}_{InvW} = \frac{\text{número de inversões removidas}}{\text{custo da operação}}$$

Neste caso, o numerador do beneficio_{InvW} pode variar no máximo em $n(n-1)/2$ inversões, conforme visto na Subseção 3.7. A Tabela 1 apresenta as métricas utilizadas para cada heurística. A heurística Mix{Inv,BP}W_0 utiliza a métrica W_0, a heurística Mix{Inv,BP}W_1 utiliza a métrica W_1, e assim por diante.

Métrica	Pontuação
W_0	Menor custo
W_1	$\frac{-\Delta inv}{n(n-1)/2} + \frac{-\Delta bp}{3}$
W_2	$\frac{\Delta inv}{n(n-1)/2} \times \frac{\Delta bp}{3}$
W_3	$-\Delta inv^{-\Delta bp}$

Tabela 1: Métricas utilizadas para cada uma das heurísticas.

A Figura 10 apresenta os resultados de custo médio obtidos utilizando as quatro métricas propostas. É possível observar que, apesar da métrica W_1 ter obtido o melhor resultado, a métrica W_2 ficou em segundo lugar com um resultado muito próximo de W_1.

Uma vez que as heurísticas Mix{Inv,BP}W_1 e Mix{Inv,BP}W_2 apresentaram os melhores resultados, aplicamos as métricas W_1 e W_2 em novas heurísticas, denotadas por Mix2{Inv,BP}W_1 e Mix2{Inv,BP}W_2. O pseudocódigo da heurística Mix2{Inv,BP}W* é apresentada na Heurística 2. Na variação Mix2{Inv,BP}W*, em vez de aplicar a métrica utilizando a melhor operação pelas heurísticas InversõesW e BreakpointsW, a métrica é aplicada para todas as operações possíveis. Dada uma operação qualquer ρ aplicada sobre uma permutação π, primeiramente calcula-se a variação no número de inversões $\Delta_{inv}(\pi, \rho)$ e a variação no número de breakpoints $\Delta_{bp}(\pi, \rho)$. Em seguida, utiliza-se a métrica selecionada para calcular a pontuação da operação. Após verificar todas as operações possíveis, a de maior pontuação será a escolhida. Os resultados obtidos a partir destas variações são apresentadas na Figura 11.

A Figura 12 apresenta quantas vezes cada um dos resultados das heurísticas InversõesW e BreakpointsW foram utilizados no grupo de heurísticas Mix{Inv,Bp}W*. O caso de empate entre os resultados foi tratado concedendo às duas heurísticas o melhor resultado, o que explica a soma resultar em mais de 100%. É possível observar que utilizando as métricas W_0 e W_1, a heurística de BreakpointsW retorna a operação de melhor benefício na maioria dos casos. Entretanto, InversõesW retornou mais operações de melhor benefício quando as métricas W_2 e W_3 foram utilizadas.

A Figura 13 apresenta os resultados de mais algumas heurísticas que não levam em consideração o tamanho da operação no cálculo de custo. A heurística GRIMM+Lin baseia-se no programa GRIMM [22], que fornece a solução ótima para permutações com sinais. O problema da ordenação por reversões para permutações sem sinais foi provado ser NP-Difícil [8], enquanto que para permutações com sinais, é possível encontrar a solução em tempo polinomial. A heurística GRIMM+Lin utiliza a decomposição de ciclos de Lin e Jiang [21] para transformar uma permutação sem sinais em uma com sinais, e então utilizando GRIMM para realizar a ordenação. Esta decomposição possui fator de aproximação de $1.4193 + \epsilon$, para $\epsilon > 0$. O algoritmo apresentado por Walter, Dias e Meidanis [16], denotado por Walter, ordena por reversões e tranposições permutações sem sinais com fa-

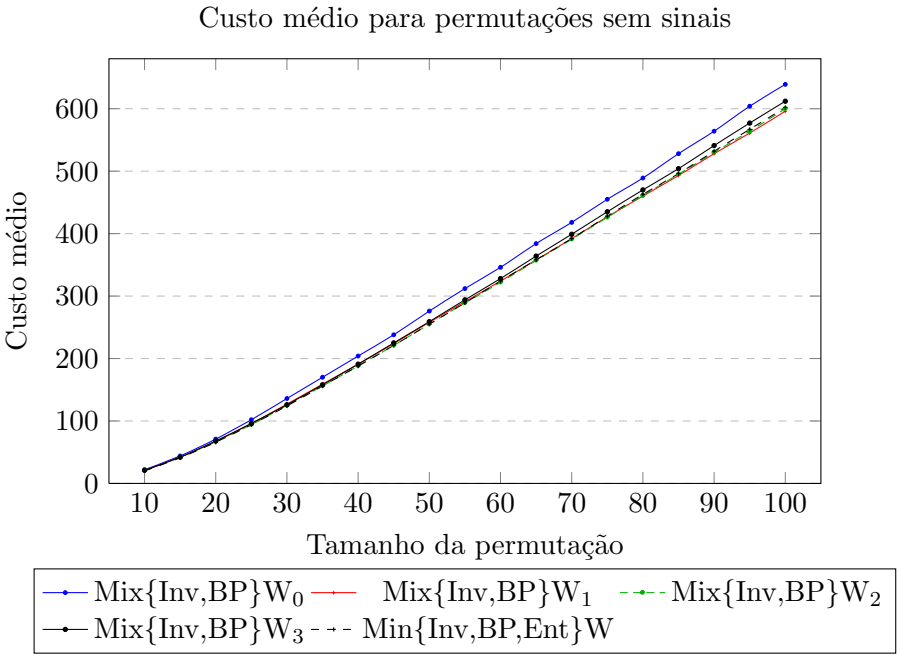

Figura 10: A cada iteração, Mix{Inv,BP}W* utiliza a operação de menor custo entre BreakpointsW e InversõesW.

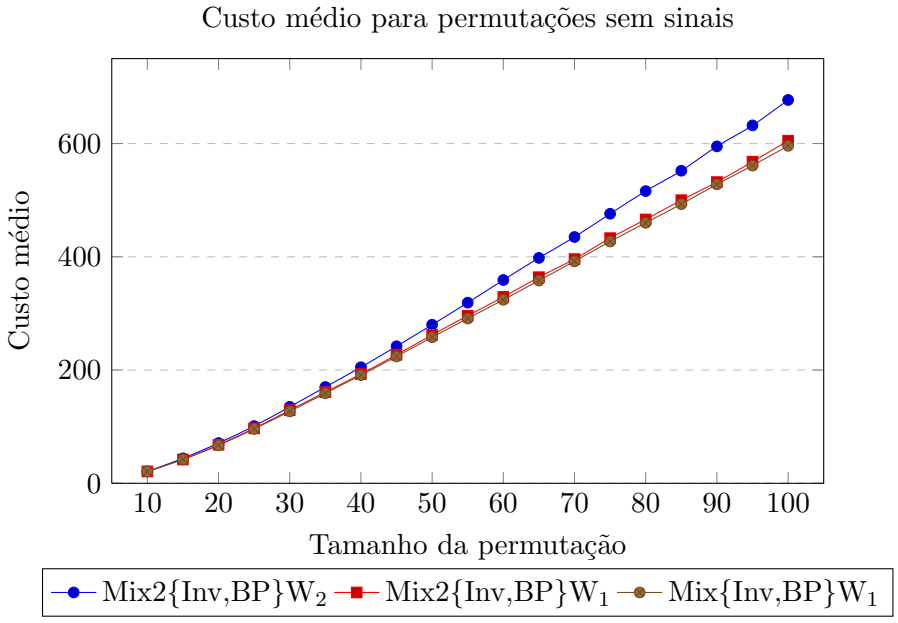

Figura 11: Mix2{Inv,BP}W_1 e W_2 testam todas as operações. A de maior pontuação é escolhida.

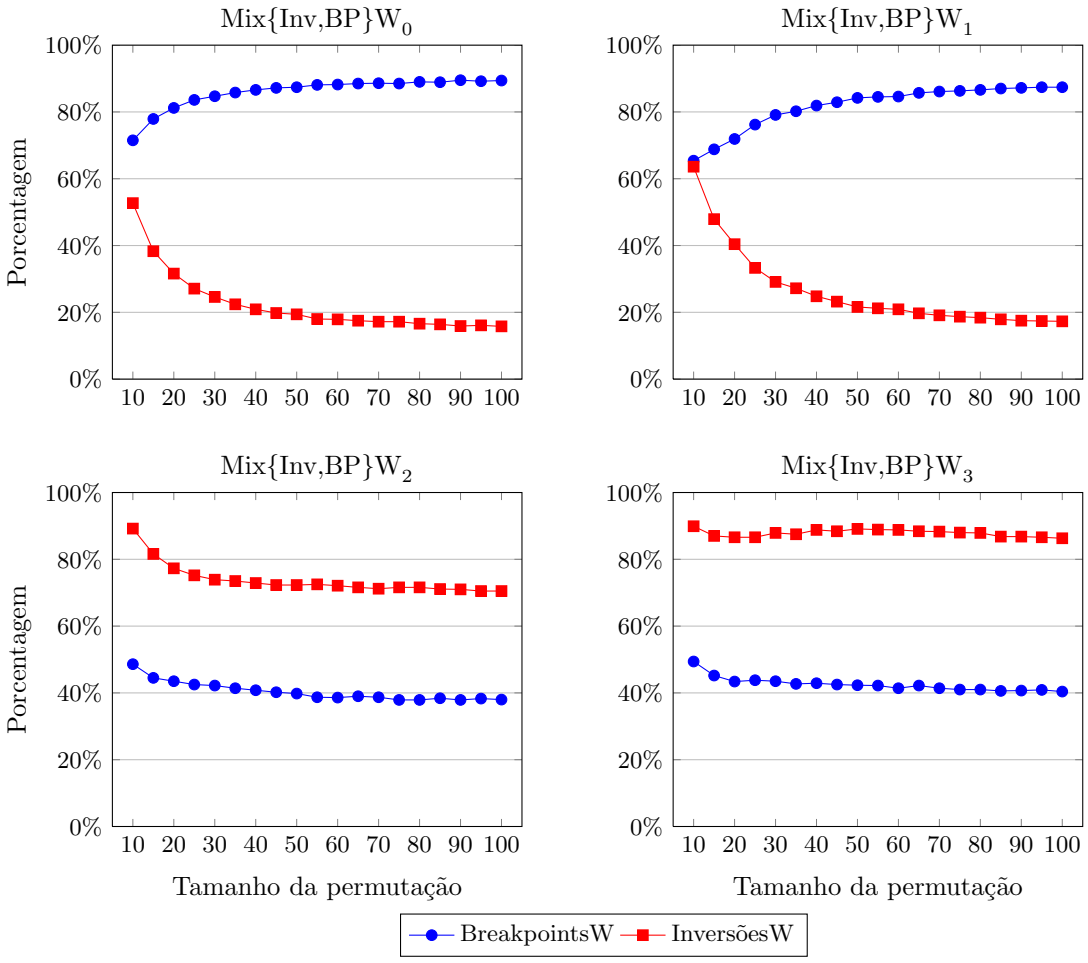

Figura 12: Para cada uma das quatro heurísticas apresentadas, temos um perfil que indica quantas vezes BreakpointsW e InversõesW retornaram o melhor resultado.

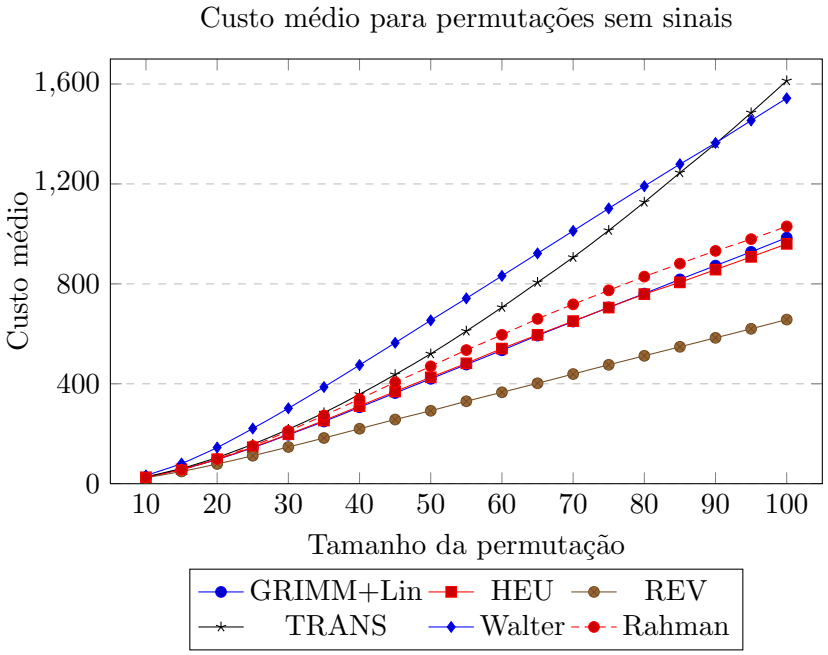

Figura 13: Custo médio de heurísticas que não possuem modificações para a versão do problema com distância ponderada.

tor de aproximação 3, utilizando breakpoints. O algoritmo que ordena permutações com transposições, apresentado por Bafna e Pevzner [12] é denotado por TRANS. O algoritmo REV ordena permutações por reversões. A heurística de Grafo de Ciclos, denotada aqui por HEU, apresentou o melhor resultado quando o custo de uma operação é unitário [1], dentre todas as heurísticas apresentadas nas Figuras 5 e 13. Todavia, quando consideramos o tamanho da operação, HEU obteve resultados medianos. Apesar de superar várias heurísticas da Figura 13, as heurísticas de Entropia e Inversões, por exemplo, retornaram custos médios menores que HEU.

A Figura 14 apresenta os resultados obtidos por heurísticas que modificam a heurística HEU, de forma que considere operações ponderadas. A heurística HEU_W modifica HEU de forma que, em vez de selecionar qualquer operação disponível, um teste seja realizado de tal forma que a operação de melhor benefício seja a escolhida. Isto ocorre nas etapas 1 e 2, uma vez que a etapa 3 utiliza um banco de configurações onde a escolha de operações é única. Conforme esperado, os custos médios desta variação diminuíram em relação à heurística original. Como esta variação não alterou a etapa 3, uma nova variação foi criada, denotada por HEU_W_BPW, que substitui inteiramente a etapa 3 pela heurística BreakpointsW. Para comparar os resultados obtidos pelas novas variações de HEU, exibimos também os resultados obtidos por Mix{Inv,BP}W_1. Como é possível observar pela Figura 14, HEU não obteve bons resultados em nenhuma das variações testadas. Era esperado que, por possuir os menores custos médios para o caso unitário, esta heurística retornasse bons resultados para o caso com distância ponderada.

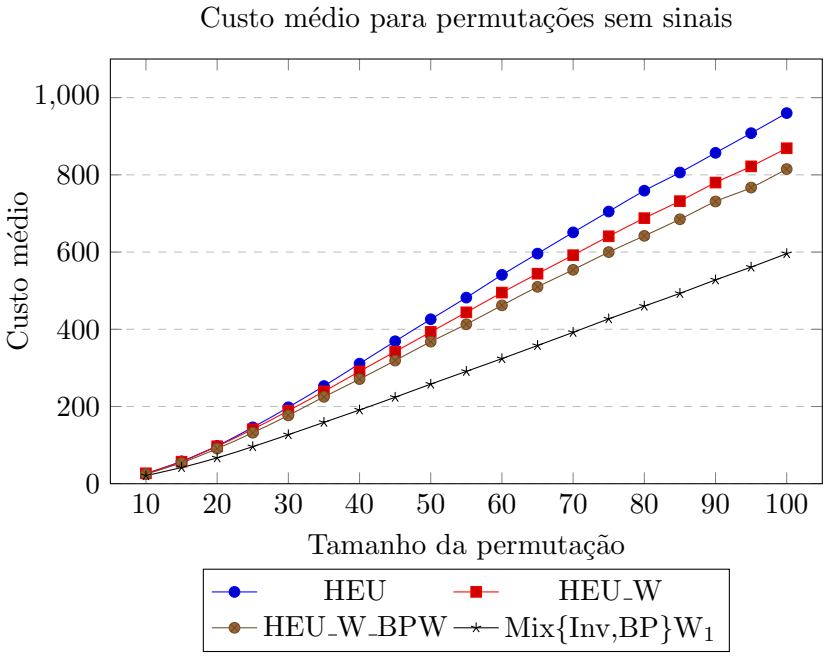

Figura 14: Custo médio de HEU e suas variações. Mix{Inv,BP}W_1 está presente como forma de comparação.

A Tabela 2 apresenta em detalhe os custos médios, para permutações de tamanho 100, de todas as heurísticas analisadas neste projeto. É possível observar que a melhor heurística foi Mix{Inv,BP}W_1. A métrica W_2 obteve bons resultados quando combinada com a heurística Mix{Inv,BP}W*, porém quando combinada com a heurística Mix2{Inv,BP}W*, apresentou custos médios excessivamente altos, maiores que os da heurística BreakpointsW. Por outro lado, a métrica W_1 apresentou bons resultados quando utilizada em ambas as heurísticas Mix{Inv,BP}W* e Mix2{Inv,BP}W*. As heurísticas não ponderadas LIS, TRANS e LIS+LDS foram as que apresentaram os piores resultados em geral. Além disso, as heurísticas HEU_W e HEU_W_BPW apresentaram os piores resultados dentre as heurísticas ponderadas. Dentre as heurísticas não ponderadas, REV e Inversões obtiveram os melhores resultados, competindo, por exemplo, com EntropiaW.

Heurística	Custo
Mix{Inv,BP}W_1	596
Mix{Inv,BP}W_2	600
Min{Inv,BP,Ent}W	602
Mix2{Inv,BP}W_1	605
Mix{Inv,BP}W_3	612
InversõesW	630
Mix{Inv,BP}W_0	639
BreakpointsW	651
REV	657
Mix2{Inv,BP}W_2	677
EntropiaW	762
Inversões	796
HEU_W_BPW	815
Entropia	833
HEU_W	869
LIS+BP	956
HEU	960
GRIMM+Lin	985
Rahman	1030
LC+RC	1081
Breakpoints	1092
Walter	1543
LIS+LDS	1611
TRANS	1613
LIS	1626

Tabela 2: Custo médio, para permutações sem sinais de tamanho 100, de todas as heurísticas analisadas neste projeto.

5 Conclusão

Com os resultados obtidos, verificou-se inicialmente que a heurística de Inversões apresentou o melhor resultado dentre as heurísticas básicas que não levam em consideração o tamanho das operações. Em seguida, constatou-se que as versões ponderadas das heurísticas de Inversões e Breakpoints, denotadas por InversõesW e BreakpointsW, obtiveram os melhores resultados dentre as heurísticas ponderadas. Com isso, criamos outras heurísticas baseadas em InversõesW e BreakpointsW que obtiveram custos ainda menores.

Por outro lado, era esperado que a heurística de Grafo de Ciclos, denotada por HEU neste trabalho e que possui os melhores resultados quando o custo é unitário, apresentasse bons resultados para o caso com distância ponderada. Nem HEU nem as variações desta heurística criadas para distância ponderada atingiram essa expectativa, apresentando custos médios acima do esperado.

Heurística 1: Mix{Inv,BP}W*(π) (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1  dist ← 0
2  enquanto π ≠ ι faça
3      ρ ← operação de melhor benefício de acordo com BreakpointsW(π)
4      σ ← operação de melhor benefício de acordo com InversõesW(π)
5      Calcule Δ_inv(π, ρ) e Δ_inv(π, σ)
6      Calcule Δ_bp(π, ρ) e Δ_bp(π, σ)
7      score_BP ← W*(Δ_bp(π, ρ), Δ_inv(π, ρ))
8      score_Inv ← W*(Δ_bp(π, σ), Δ_inv(π, σ)
9      se score_BP > score_Inv então
10         α ← π · ρ
11     senão
12         α ← π · σ
13     fim
14     π ← α
15     dist ← dist + 1
16 fim
17 retorna dist
```

Heurística 2: Mix2{Inv,BP}W*(π) (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1  dist ← 0
2  enquanto π ≠ ι faça
3      best ← 0
4      para cada operação ρ válida faça
5          Calcule Δ_inv(π, ρ) e Δ_bp(π, ρ)
6          score ← W*(Δ_inv(π, ρ),Δ_bp(π, ρ))
7          score ← score/custo(ρ)
8          se score > best então
9              α ← π · ρ
10             best ← score
11         fim
12     fim
13     π ← α
14     dist ← dist + 1
15 fim
16 retorna dist
```

Referências

[1] A. R. Oliveira e Z. Dias. "O Problema da Ordenação de Permutações por Reversões e Transposições". Dissertação de mestrado, Instituto de Computação, Universidade Estadual de Campinas, São Paulo, 2015.

[2] R. Y. Pinter and S. Skiena. "Genomic sorting with length-weighted reversals". *Genome Informatics*, vol. 13, pp. 103-111, 2002.

[3] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. "Improved bounds on sorting by length-weighted reversals". *Journal of Computer and System Sciences*, vol. 74, no. 5, pp. 744–774, 2008.

[4] S. Hannenhalli and P. Pevzner. "Tranforming men into mice (polynomial algorithmic for genomic distance problem)". In *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science - FOCS'95*, pages 581–592, 1995.

[5] A. Bergeron. "A very elementary presentation of the Hannenhalli-Pevzner theory". *Discrete Applied Mathematics*, vol. 146, pp. 134–145, 2005.

[6] E. Tannier, A. Bergeron, and M.-F. Sagot. "Advances on sorting by reversals". *Discrete Applied Mathematics*, vol. 155, no. 6-7, pp. 881-888, 2007.

[7] D. A. Bader, B. M. E. Moret, and M. Yan. "A linear-time algorithm for computing inversion distance between signed permutations with an experimental study". *Journal of Computational Biology*, vol. 8, no. 5, pp. 483–491, 2001.

[8] A. Caprara. "Sorting permutations by reversals and Eulerian cycle decompositions". *SIAM Journal on Discrete Mathematics*, vol. 12, no. 1, pp. 91–110, 1999.

[9] A. Caprara. "Sorting by reversals is difficult". In *Proceedings of the 1st Annual International Conference on Computational Molecular Biology (RECOMB'1997)*, pages 75–83, Santa Fe, New Mexico, USA, 1997.

[10] P. Berman, S. Hannenhalli, and M. Karpinski. "1.375-approximation algorithm for sorting by reversals". In *Proceedings of the 10th European Symposium on Algorithms (ESA'2002)*, Rome, Italy, 2002, pp. 200–210.

[11] L. Bulteau, G. Fertin, and I. Rusu. "Sorting by transpositions is difficult". *SIAM Journal on Discrete Mathematics*, vol. 26, no. 3, pp. 1148–1180, 2012.

[12] V. Bafna and P. A. Pevzner. "Sorting by transpositions". *SIAM Journal on Discrete Mathematics*, vol. 11, no. 2, pp. 224–240, 1998.

[13] D. A. Christie. "Genome rearrangement problems". PhD thesis, Glasgow University, 1998.

[14] M. E. M. T. Walter, Z. Dias, and J. Meidanis. "A new approach for approximating the transposition distance". In *Proceedings of the 7th International Symposium on String Processing Information Retrieval (SPIRE'2000)*, pages 199–208, A Coruña, Spain, 2000.

[15] I. Elias and T. Hartman. "A 1.375-approximation algorithm for sorting by transpositions". *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 369–379, 2006.

[16] M. E. M. T. Walter, Z. Dias, and J. Meidanis. "Reversal and transposition distance of linear chromosomes". In *Proceedings of the 5th International Symposium on String Processing Information Retrieval (SPIRE'1998)*, pages 96–102, Santa Cruz, Bolivia, 1998.

[17] F. Swidan, M. Bender, D. Ge, S. He, H. Hu, and R. Pinter. "Sorting by length-weighted reversals: Dealing with signs and circularity". In *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science, S. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, Eds. Springer Berlin Heidelberg, 2004, vol. 3109, pp. 32–46

[18] A. Rahman, S. Shatabda, and M. Hasan. "An approximation algorithm for sorting by reversals and transpositions". *Journal of Discrete Algorithms*, vol. 6 no. 3, pp 449–457, 2008.

[19] X. Chen. "On sorting unsigned permutations by double-cut-and-joins". *Journal of Combinatorial Optimization*, vol. 25, no. 3, pp 339–351, 2013.

[20] M. Benoît-Gagné and S. Hamel. "A new and faster method of sorting by transpositions". In *Proceedings of the 18th Annual Conference on Combinatorial Pattern Matching (CPM'2007)*, pages 131–141, Berlin, Heidelberg, 2007.

[21] G. Lin and T. Jiang. "A further improved approximation algorithm for breakpoint graph decomposition". *Journal of Combinatorial Optimization*, vol. 8, no. 2, pp. 183–194, 2004.

[22] G. Tesler. "GRIMM: genome rearrangements web server". *Bioinformatics*, vol. 18, no. 3, pp. 492–493, 2002.