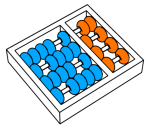
Universidade Estadual de Campinas
Instituto de Computação



Andre Rodrigues Oliveira

O Problema da Ordenação de Permutações por Reversões e Transposições

CAMPINAS
2015

Universidade Estadual de Campinas
Instituto de Computação

Andre Rodrigues Oliveira

O Problema da Ordenação de Permutações 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 Andre Rodrigues Oliveira e orientada pelo Prof. Dr. Zanoni Dias.

CAMPINAS
2015

Agência(s) de fomento e nº(s) de processo(s): CNPq, 147337/2013-5

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

OL4p

Oliveira, Andre Rodrigues, 1990-
O problema da ordenação de permutações por reversões e transposições / Andre Rodrigues Oliveira. – Campinas, SP : [s.n.], 2015.

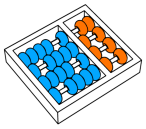
Orientador: Zanoni Dias.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

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

Informações para Biblioteca Digital

Título em outro idioma: The sorting permutations problem by reversals and transpositions
Palavras-chave em inglês:
Computational biology
Genomes
Approximation algorithms
Permutations
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Zanoni Dias [Orientador]
Maria Emilia Machado Telles Walter
Guilherme Pimentel Telles
Data de defesa: 02-10-2015
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação



Andre Rodrigues Oliveira

O Problema da Ordenação de Permutações por Reversões e Transposições

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação - UNICAMP
- Profa. Dra. Maria Emilia Machado Telles Walter
  Departamento de Ciência da Computação - CIC
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação - UNICAMP

A ata da defesa, onde constam as assinaturas dos membros da banca, está arquivada pela Universidade Estadual de Campinas.

Dedicatória

Aos meus pais João e Edilamar, minha irmã Thais, e meus sobrinhos Alícia e Davi. Amo vocês incondicionalmente.

Agradecimentos

A todos os membros da minha família, em especial meus pais João e Edilamar e à minha irmã Thais, por todo o amor, conselhos e apoio durante todos estes anos. Por estarem sempre com uma palavra de incentivo para a concretização dos meus sonhos e por celebrarem comigo cada conquista.

Ao meu orientador Zanoni Dias por todo o apoio, incentivo, ideias, críticas e principalmente pela confiança depositada em mim durante este trabalho. Por estar sempre disponível para me ajudar e por todos os conselhos valiosos dados durante este tempo.

Ao Ulisses Dias, por todas as dicas, explicações e principalmente pela ajuda durante este período, que foram fundamentais para este trabalho.

A todos os meus amigos e amigas, novos e velhos, pelo apoio, pelas comemorações e pela ajuda sempre que precisei.

Ao CNPq pelo suporte financeiro durante este período.

Resumo

Rearranjos de Genomas, diferentemente das mutações mais comuns que afetam pontualmente o genoma, são eventos de mutação globais que agem sobre grandes trechos do genoma, e são um desafio para a Teoria da Computação, uma vez que, na maioria dos casos, computar a distância entre dois genomas considerando operações globais resultam em problemas NP-Difíceis. De forma similar, podemos definir o Problema da Ordenação de Genomas, em que buscamos calcular o número mínimo de eventos de rearranjos necessários para ordenar blocos conservados de genomas, onde os genomas são representados por permutações. Um modelo de rearranjo determina quais eventos de rearranjo são permitidos para ordenar uma permutação ou transformar uma permutação em outra. Dentre os eventos de rearranjo mais comuns, podemos citar as reversões e as transposições. Modelos considerando os dois eventos, separadamente, já possuem vasta bibliografia. Entretanto, modelos considerando os dois eventos em conjunto ainda são pouco explorados. Nesta dissertação, apresentamos diversas heurísticas para ordenação de permutações por reversões e transposições considerando permutações com sinais e permutações sem sinais. Para tanto, diversas métricas compatíveis com o problema foram utilizadas em heurísticas, que mais tarde foram testadas com um grande conjunto de permutações aleatórias. As métricas que obtiveram os melhores resultados foram utilizadas em uma heurística denominada Heurística de Grafo de Ciclos. Além disso, foram criados bancos de configurações que auxiliam esta heurística no processo de ordenação de uma permutação. Os resultados obtidos pela Heurística de Grafo de Ciclos foram comparados com algoritmos já existentes na literatura, tanto os que permitem reversões e transposições, bem como os que consideram apenas um dos eventos, evidenciando que a Heurística de Grafo de Ciclos produz os melhores resultados para todos os conjuntos de permutações com sinais, e em grande parte dos conjuntos de permutações sem sinais.

Abstract

Genome rearrangements, unlike most common mutations that affects only one nucleotide, are global mutation events that affect large fragments of genomes and present a challenge to the Computational Theory Field, since in most cases, computing the distance between genomes considering global operations results in NP-Hard problems. Moreover, the Sorting Permutations by Genome Rearrangements problem seeks to compute the smallest number of rearrangement events needed to sort conserved blocks of genomes, with genomes represented by permutations. A rearrangement model determines which rearrangement events are allowed to sort a permutation or to transform a permutation into another. The two most common rearrangement events are reversals and transpositions. Rearrangement models considering reversals and transpositions separately have an extensive bibliography. However, models considering both events are still little explored. In this dissertation, we present some heuristics that sort permutations by reversals and transpositions considering signed and unsigned permutations. To do this, various metrics that are compatible with the problem were used. These heuristics were later tested with a large set of random permutations. Metrics that have achieved the best results were used in a heuristic named Cycle Graph Heuristic. In addition, cycle configurations were created and stored in a database in order to help our heuristic in the sorting process. The results obtained from Cycle Graph Heuristic were compared with existing algorithms in the literature, showing that our heuristic produces the best results for all sets of signed permutations and in many of the sets of unsigned permutations.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Problemas de rearranjo de genomas buscam, dados dois genomas, encontrar uma sequência de rearranjos que transformam um genoma no outro. Diferente das mutações pontuais que afetam moléculas constituintes do genoma, os eventos de rearranjo afetam grandes porções do genoma, tornando esta abordagem mais adequada para comparação de genomas completos.

O genoma de uma espécie é composto de cromossomos representados como um conjunto ordenado de genes. Outra forma de representação dos cromossomos se dá por meio de blocos conservados, sendo esses blocos regiões de alta similaridade entre os dois genomas comparados. Estes blocos podem ser genes ou qualquer subsequência conservada em ambos os genomas. Um problema de rearranjo de genomas consiste então em comparar genomas de dois indivíduos e encontrar a menor sequência de eventos de rearranjo que transformam um genoma em outro. O tamanho desta sequência pode ser utilizado para estimar a distância evolucionária ou mesmo o grau de parentesco entre os dois genomas comparados e parte do princípio que rearranjos de genomas são eventos relativamente raros e que a natureza sempre utiliza o menor número de operações, com base em uma teoria conhecida como Princípio da Máxima Parcimônia.

Um evento de rearranjo ocorre com a quebra de um ou mais cromossomos e os segmentos resultantes são unidos, de forma que o conjunto de genes permanece o mesmo, porém a ordem é alterada. Além disso, a orientação destes segmentos também pode ser alterada. Na literatura foram propostos diversos eventos ou operações de rearranjo de genomas, dentre os quais podemos citar fusão, fissão, reversão e transposição:

- **Fusões** ocorrem quando dois cromossomos se unem para formar um único cromossomo;
- **Fissões** ocorrem quando um cromossomo se divide em dois cromossomos;
- **Reversões** ocorrem quando um segmento do genoma é destacado e inserido no mesmo local, porém com ordem invertida em relação à sua posição original;
- **Transposições** ocorrem quando um segmento do genoma é destacado e inserido em outra posição. A ordem deste segmento não é alterada.

Um *modelo de rearranjo* determina o conjunto de operações permitidas para se transformar um genoma em outro. Nos estudos iniciais em rearranjos de genomas, o foco se deu em modelos de rearranjo que permitiam apenas um tipo de operação, capazes de explicar cenários evolutivos simples. Mais tarde, surgiram modelos que permitiam duas ou mais operações, possibilitando assim a análise de cenários evolutivos mais complexos. Esta dissertação tem como objetivo o estudo da Ordenação de Permutações quando o modelo de rearranjo permite que duas operações sejam realizadas: reversões e transposições.

O restante desta dissertação está estruturado da seguinte forma. O Capítulo 2 apresenta uma série de conceitos necessários para entender melhor o problema estudado nesta dissertação. O Capítulo 3 apresenta uma série de heurísticas que não utilizam o grafo de ciclos, bem como analisa os resultados obtidos por estas heurísticas. O Capítulo 4 apresenta uma heurística que utiliza o conceito de grafo de ciclos. Além disso, os resultados desta heurística são comparados com os resultados obtidos por algoritmos existentes na literatura. O Capítulo 5 apresenta as conclusões do estudo realizado e propõe extensões para o trabalho.

Capítulo 2

Fundamentação Teórica

Em Problemas de Rearranjos de Genomas, um genoma é representado como uma n-tupla cujos elementos representam os genes. Supondo que não haja repetição de genes, esta n-tupla é uma permutação $\pi = (\pi_1\ \ \pi_2\ \ ...\ \ \pi_n)$, com $\pi_i \in \{-n, -(n-1), ..., -2, -1, +1, +2, ..., +(n-1), +n\}$ e $|\pi_i| \neq |\pi_j| \leftrightarrow i \neq j$. Nesse caso cada elemento π_i possui um sinal, $+$ ou $-$, que indica a orientação do gene que ele representa, e a permutação é dita *com sinais*. Quando não há informação sobre a orientação dos genes, esse sinal é omitido, e a permutação é dita *sem sinais*. A posição de um elemento qualquer π_k em uma permutação é denotada por $p(\pi_k)$.

Denotamos por $u(\pi)$ o processo de remoção dos sinais dos elementos de uma permutação π. Além disso, denotamos por $neg(\pi)$ o número de elementos com sinais negativos de uma permutação π. Por exemplo, se $\pi = (+5\ \ -2\ \ -1\ \ +3\ \ +4)$, temos que $u(\pi) = (5\ \ 2\ \ 1\ \ 3\ \ 4)$ e $neg(\pi) = 2$, uma vez que existem dois elementos em π cujo sinal é negativo. Note que para qualquer permutação π sem sinais, $u(\pi) = \pi$ e $neg(\pi) = 0$.

Seja ι a *permutação identidade* definida como $\iota = (1\ \ 2\ \ ...\ \ n-1\ \ n)$. Se ι é a permutação identidade com sinais, todos os elementos de ι possuem sinal positivo. Da mesma forma, seja $\iota_{inv} = (n\ \ n-1\ \ ...\ \ 2\ \ 1)$ a permutação identidade invertida. Se ι_{inv} é a permutação identidade invertida com sinais, todos os elementos de ι_{inv} possuem sinal negativo.

Dadas duas permutações π e σ, a *composição* entre elas, denotada por $\pi \cdot \sigma$, é $\pi \cdot \sigma = (\pi_{\sigma_1}\ \ \pi_{\sigma_2}\ \ ...\ \ \pi_{\sigma_n})$, de forma similar à composição de funções.

A operação de composição induz uma estrutura de grupo sobre o conjunto formado por todas as permutações de um determinado tamanho. O grupo formado por todas as $n!$ permutações sem sinais de tamanho n juntamente com a operação de composição é chamado de *grupo simétrico* e é denotado por S_n. O grupo formado por todas as $n!2^n$ permutações com sinais de tamanho n, juntamente com a operação de composição é chamado de *grupo simétrico com sinais*, e é denotado por $S_n^{\pm}$.

Dadas duas permutações π e σ e um modelo de rearranjo M, o problema de transformar a permutação π na permutação σ consiste em encontrar a menor sequência de operações $\rho_1, \rho_2, ..., \rho_k$ pertencentes a M tal que $\pi \cdot \rho_1 \cdot \rho_2 \cdot ... \cdot \rho_k = \sigma$. O tamanho desta sequência representa a *distância* entre as permutações π e σ com respeito ao modelo M e é denotada por $d_M(\pi, \sigma)$ (neste caso, $d_M(\pi, \sigma) = k$).

Da mesma forma, Problemas de Ordenação por Rearranjos consistem em aplicar a menor sequência de operações $\rho_1, \rho_2, ...\rho_n$ permitidas pelo modelo em uma permutação π

tal que $\pi \cdot \rho_1 \cdot \rho_2 \cdot ... \cdot \rho_n = \iota$. A *distância de ordenação* de uma permutação π é o tamanho da menor sequência que transforma π na permutação identidade, e a distância entre elas com respeito a um modelo M é denotada por $d_M(\pi, \iota_n) = d_M(\pi)$.

Note que $d_M(\pi, \sigma) = d_M(\alpha)$ se tomarmos $\alpha = \sigma^{-1} \cdot \pi$, onde σ^{-1} é a inversa de σ tal que $\sigma^{-1} \cdot \sigma = \iota$, pois temos que $d_M(\pi, \sigma) = d_M(\sigma^{-1} \cdot \pi, \sigma^{-1} \cdot \sigma) = d_M(\alpha, \iota) = d_M(\alpha)$. Por exemplo, se $\pi = (1\ 3\ 2\ 5\ 4)$ e $\sigma = (2\ 4\ 5\ 1\ 3)$, então $\sigma^{-1} = (4\ 1\ 5\ 2\ 3)$, $\sigma^{-1} \cdot \sigma = (1\ 2\ 3\ 4\ 5) = \iota$, e $\alpha = \sigma^{-1} \cdot \pi = (4\ 5\ 1\ 3\ 2)$. Desta forma, a distância entre π e σ será a mesma que a distância de ordenação da permutação α.

A maior distância possível entre todas as permutações no mesmo grupo simétrico S_n (ou $S_n^{\pm}$ caso π seja uma permutação com sinais) com respeito ao modelo M é chamada de *diâmetro*, e é denotada por $D_M(n)$.

A *permutação estendida* pode ser obtida a partir de π inserindo-se dois novos elementos: $\pi_0 = 0$ e $\pi_{n+1} = n + 1$. A partir deste momento, a não ser que explicitado, estaremos sempre nos referindo à versão estendida de qualquer permutação, mesmo nos casos em que estes elementos extras forem omitidos.

2.1 Breakpoints

Um *breakpoint de reversão* ocorre entre um par de elementos π_i e π_{i+1} de uma permutação π, com $0 \leq i \leq n$, se $|\pi_i - \pi_{i+1}| \neq 1$. A permutação identidade é a única permutação que não possui breakpoints de reversão. Por exemplo, dado $\pi = (0\ 3\ 4\ 1\ 2\ 5)$, existe um breakpoint entre os elementos $(0, 3)$, $(4, 1)$ e $(2, 5)$. Breakpoints de reversão serão denotados por "$\cdot$". O número de breakpoints de reversão em uma permutação π é denotado por $b_r(\pi)$. No exemplo acima, temos que $b_r(\pi) = 3$ e $b_r(\iota) = 0$.

Um *breakpoint de reversão com sinal* ou um *breakpoint de transposição* ocorre entre um par de elementos π_i e π_{i+1}, com $0 \leq i \leq n$ se $\pi_{i+1} - \pi_i \neq 1$. A permutação identidade é a única permutação que não possui breakpoints de transposição nem breakpoints de reversão com sinal. O número de breakpoints de reversão com sinal em uma permutação com sinais π é denotado por $b_{\overline{r}}(\pi)$. O número de breakpoints de transposições em uma permutação π é denotado por $b_t(\pi)$. Por exemplo, a permutação com sinais $\pi = (+0\ +1\ -2\ +3\ +4\ -5\ +6)$ possui breakpoints de reversão com sinal entre os elementos $(+1, -2)$, $(-2, +3)$, $(+4, -5)$ e $(-5, +6)$ e, desta forma, $b_{\overline{r}}(\pi) = 4$. Já a permutação $\sigma = (0\ 1\ 2\ 4\ 3\ 5\ 6)$ possui breakpoints de transposição entre os elementos $(2, 4)$, $(4, 3)$ e $(3, 5)$ e, desta forma, $b_t(\sigma) = 3$.

2.2 Strips

Dado um modelo de breakpoint, que pode ser de reversão, de reversão com sinal ou de transposição, uma *strip* de uma permutação π é uma sequência de elementos $\pi_i ... \pi_j$, com $0 \leq i \leq j \leq n + 1$, tal que:

- $i > 0$ e (π_{i-1}, π_i) é um breakpoint;
- $j = n + 1$ ou $j < n + 1$ e (π_j, π_{j+1}) é um breakpoint;

- não existe um breakpoint entre os pares (π_k, π_{k+1}), com $i \leq k \leq j-1$.

Uma strip é dita *crescente* caso seus elementos formem uma sequência crescente e é dita ser *decrescente* caso contrário. Se a strip contém apenas um elemento ela é chamada de *singleton* e definida como crescente. Por exemplo, dada a permutação $\pi = (0\ 3\ 4\ 5\ 2\ 1\ 6)$, e considerando breakpoints de reversão, temos que π possui 4 strips: as strips $\langle 0 \rangle$ e $\langle 6 \rangle$, que são singletons, a strip $\langle 3\,4\,5 \rangle$ que é crescente e a strip $\langle 2\,1 \rangle$ que é decrescente. Se considerarmos breakpoints de transposição temos que π possui 5 strips: as strips $\langle 0 \rangle$, $\langle 2 \rangle$, $\langle 1 \rangle$, e $\langle 6 \rangle$, que são singletons, e a strip $\langle 3\,4\,5 \rangle$, que é crescente.

2.3 Grafo de Ciclos e Grafo de Breakpoints

O grafo de ciclos $G(\pi)$ de uma permutação com sinais π, é um grafo formado pelo conjunto de vértices $V(\pi)$, o conjunto de arestas pretas $E_p(\pi)$ e o conjunto de arestas cinzas $E_c(\pi)$ tais que:

- $V(\pi) = \{0, -\pi_1, +\pi_1, -\pi_2, +\pi_2, ..., -\pi_n, +\pi_n, -n{+}1\}$
- $E_c(\pi) = \bigcup_{i=1}^{n+1} \{(+(i-1), -i)\}$
- $E_p(\pi) = \bigcup_{i=1}^{n+1} \{(-\pi_i, +\pi_{i-1})\}$

Neste caso, todo vértice de $G(\pi)$ possui exatamente uma aresta cinza pareada com uma aresta preta. Isto implica na existência de uma decomposição única das arestas de $G(\pi)$ em ciclos com arestas de cores alternantes, chamados de ciclos alternantes.

O *grafo de breakpoints* de uma permutação sem sinais π, denotado por $G_b(\pi)$, possui conjunto de vértices dado por $V = \{\pi_0, \pi_1, ..., \pi_n, \pi_{n+1}\}$, que correspondem a cada elemento da notação estendida de π, e conjunto de arestas dado por $E = E_P \cup E_C$, onde E_P é o conjunto de arestas pretas e E_C é o conjunto de arestas cinzas. Existe uma aresta preta entre π_i e π_{i+1} se existe um breakpoint de reversão entre π_i e π_{i+1}, com $0 \leq i \leq n$. Existe uma aresta cinza entre dois vértices π_i e π_j, para algum $0 \leq i < j \leq n+1$ se $\pi_j = \pi_i \pm 1$ e π_i e π_j não são adjacentes em π. A idéia do grafo de breakpoints é que as arestas cinzas descrevem a ordem dos elementos na permutação identidade ι, que é nosso alvo ao ordenar π, e as arestas pretas descrevem a ordem dos elementos na permutação π. A Figura 2.1 mostra o grafo de breakpoints para a permutação $\pi = (2\ 1\ 3\ 6\ 4\ 5)$. O grafo de ciclos $G(\pi)$ de uma permutação sem sinais π deve ser obtido através de $G_b(\pi)$.

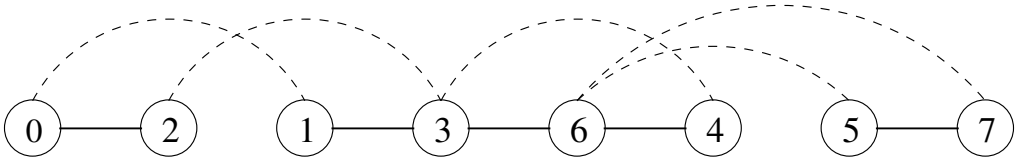


Figura 2.1: Grafo de breakpoints da permutação sem sinais $\pi = (2\ 1\ 3\ 6\ 4\ 5)$. Arestas cinzas estão representadas por arcos tracejados e arestas pretas por linhas retas.

Note que os vértices do grafo de breakpoints podem conter até 4 arestas. Desta forma, o grafo de ciclos $G(\pi)$ pode ser obtido através de uma decomposição de $G_b(\pi)$ em ciclos alternantes. Além disso, é necessário inserir as arestas cinzas e pretas que não existem em $G_b(\pi)$, ou seja, arestas pretas e cinzas entre dois vértices π_i e π_{i+1} tal que π_i e π_{i+1} são adjacentes e $|\pi_{i+1} - \pi_i| = 1$. Essas arestas inseridas formam o conjunto de ciclos com apenas uma aresta preta e uma aresta cinza em $G(\pi)$, e este conjunto será denotado por $\overline{c}_b(\pi)$.

Denota-se por $|c_b(\pi)|$ o número de ciclos de uma decomposição de $G_b(\pi)$ em ciclos alternantes com arestas disjuntas. Encontrar uma decomposição máxima no número de ciclos alternantes é um problema NP-Difícil [10]. O melhor fator de aproximação conhecido é $1.4167 + \epsilon$, para $\epsilon > 0$, proposto por Chen [12].

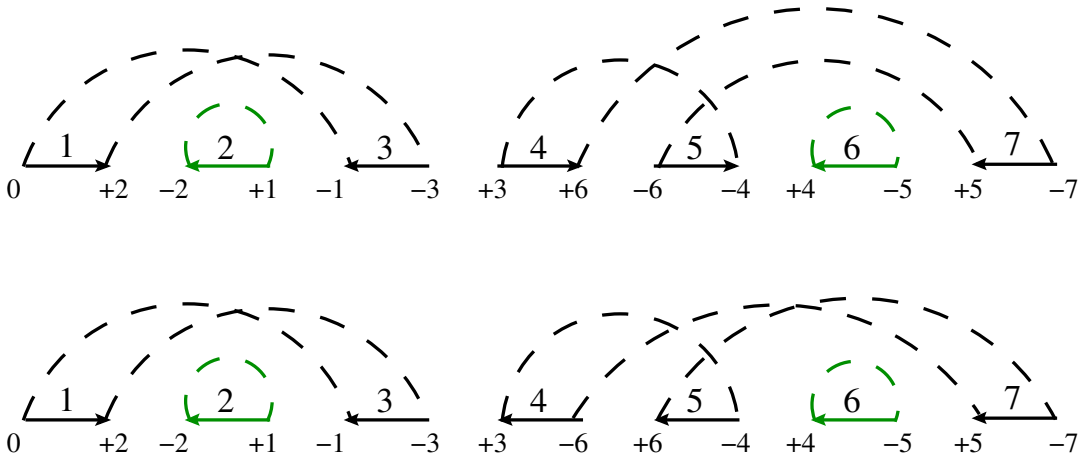


Figura 2.2: Duas possíveis decomposições do grafo de breakpoints da permutação sem sinais $\pi = (2\ 1\ 3\ 6\ 4\ 5)$, onde arestas cinzas estão representadas por arcos tracejados e arestas pretas por linhas retas. As arestas na cor verde não fazem parte da decomposição em ciclos alternantes e formam o conjunto $\overline{c_b}(\pi)$.

Para $G_b(\pi)$ da Figura 2.1, duas possíveis decomposições máximas em ciclos alternantes $c_b(\pi)$ estão representadas pelos ciclos da cor preta na Figura 2.2. Os ciclos com apenas uma aresta preta, contidos em $\overline{c}_b(\pi)$, estão representados pela cor verde. Desta forma, para permutações sem sinais, $G(\pi) = c_b(\pi) \cup \overline{c}_b(\pi)$.

Dado o grafo de ciclos $G(\pi)$ de uma permutação qualquer, as arestas pretas são rotuladas de 1 a $n + 1$, de modo que a aresta $(-\pi_i, +\pi_{i-1})$ recebe o rótulo i.

Um ciclo alternante é chamado de $k-$ciclo se ele possui k arestas pretas. Além disso, um k-ciclo é dito *par* se k é par. Caso contrário o k-ciclo é dito *ímpar*. Seja $c(\pi)$ o número de ciclos de $G(\pi)$, e seja $c_{impar}(\pi)$ o número de ciclos ímpares de $G(\pi)$. A permutação ι é a única cujo grafo de ciclos alternantes possui $n + 1$ ciclos, sendo todos ímpares.

Um ciclo C é representado pela listagem dos vértices na ordem em que aparecem no ciclo. Seja um ciclo $C = (v_1, v_2, ..., v_k)$, assume-se que v_1 é o vértice localizado mais à direita.

Seja $C = (v_1, v_2, v_3, v_4, ...)$ o ciclo da Figura 2.3 escrito utilizando a regra de listagem acima. A representação simplificada lista o ciclo em ordem dos rótulos das arestas pretas $\{(v_1, v_2), (v_3, v_4), ...\}$ e associamos a estes rótulos os sinais '+' e '−' para indicar como

as arestas pretas são percorridas. Se uma aresta preta é percorrida da direita para a esquerda, então o sinal '+' é associado a seu rótulo e dizemos que esta aresta preta é positiva. Caso contrário, o sinal '−' é associado a seu rótulo e dizemos que a aresta preta é negativa.

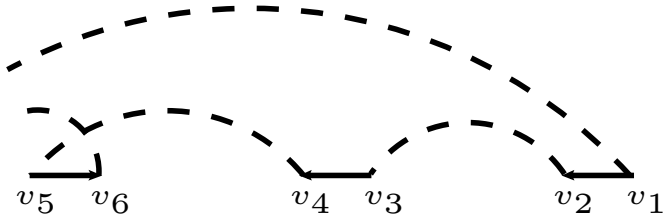


Figura 2.3: Exemplo de um grafo de ciclos genérico com um ciclo $C = (v_1, v_2, v_3, v_4, v_5, v_6, ...)$, onde arestas cinzas estão representadas por arcos tracejados e arestas pretas por linhas retas.

Dado que v_1 é o vértice mais à direita do ciclo, convencionamos que a aresta (v_1, v_2) é sempre percorrida da direita para a esquerda, e, desta forma, a primeira aresta preta de um ciclo será sempre positiva. Pelo mesmo motivo, todo 1-ciclo é sempre positivo.

Duas arestas cinzas $g_1 = (v_i, v_j)$ e $g_2 = (v_k, v_l)$ se *cruzam* caso os intervalos $[i, j]$ e $[k, l]$ se intersectam, mas um não contém o outro. Dois ciclos C_1 e C_2 se *cruzam* se existem duas arestas cinzas g_1 e g_2 que se cruzam tais que $g_1 \in C_1$ e $g_2 \in C_2$. Além disso, duas triplas $(c_1, c_2, c_3) \in C$ e $(d_1, d_2, d_3) \in D$ de arestas pretas estão *entrelaçadas* caso $c_1 < d_1 < c_2 < d_2 < c_3 < d_3$, ou $d_1 < c_1 < d_2 < c_2 < d_3 < c_3$. Note que esta tripla não precisa ser necessariamente a leitura do ciclo. Por exemplo, os ciclos $C_1 = (5, 1, 3)$ e $C_2 = (2, 4, 6)$ estão entrelaçados, mas consideramos aqui a tripla orientada de C_1 como $(5, 3, 1)$.

Um $k-$ciclo com $k \geq 3$ é chamado não orientado se suas arestas pretas formam uma sequência decrescente. Caso contrário, o ciclo é chamado de orientado. Além disso, se todas as arestas pretas de um ciclo C são positivas, C é dito *convergente*. Se C possui pelo menos uma aresta preta negativa, C é dito *divergente*.

Dado um ciclo $D = (..., d_1, d_2, ...)$ a aresta cinza p que liga a aresta preta d_1 à aresta preta d_2 é dita *direita* se $d_2 > d_1$. Caso contrário, p é dita *esquerda*. Note que qualquer ciclo não orientado $E = (e_1, e_2, ..., e_n)$ só possui uma aresta cinza direita, que liga a aresta preta e_n à aresta preta e_1. As demais arestas cinzas devem ser esquerdas pois $e_{i+1} < e_i$ para $1 \leq i < n$.

A Figura 2.4 mostra o grafo de ciclos da permutação $\pi = (+5\ -3\ -4\ -1\ -2)$, onde existem os ciclos ímpares $C_1 = (6, 4, 2)$, representado pela cor vermelha, e $C_2 = (5, -1, 3)$, representado pela cor preta. Note que o ciclo C_1 é não orientado convergente, já que as arestas pretas formam uma sequência decrescente e não existem arestas pretas negativas. Além disso, C_1 possui duas arestas cinzas esquerdas, $(6, 4)$ e $(4, 2)$, e uma aresta cinza direita $(2, 6)$. Já o ciclo C_2 é orientado divergente, uma vez que as arestas pretas não formam uma sequência decrescente e a aresta 1 possui sinal negativo. Além disso, C_2 possui uma aresta cinza esquerda $(5, -1)$ e duas arestas cinzas direitas $(-1, 3)$ e $(3, 5)$.

Um ciclo do grafo de ciclos é chamado de *curto* caso contenha no máximo 3 arestas pretas, e *longo* caso contrário. Uma permutação é dita *simples* se seu grafo de ciclos

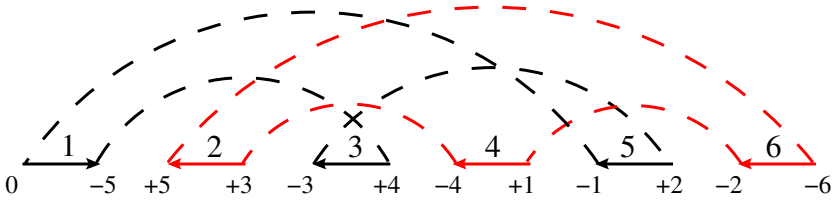


Figura 2.4: Grafo de ciclos da permutação $\pi = (+5\ -3\ -4\ -1\ -2)$.

possui apenas ciclos curtos. É possível transformar uma permutação não simples π em uma permutação simples $\hat{\pi}$, adicionando novos elementos de forma a quebrar os ciclos longos de π.

Esta abordagem de quebrar ciclos longos é amplamente utilizada na bibliografia, tanto na ordenação de permutações por transposições [19], quanto na ordenação de permutações por reversões [26]. Temos então que $\hat{\pi}$ é obtida através da inserção de novos elementos em π e, desta forma, $d(\pi) \leq d(\hat{\pi})$, uma vez que inserir novos elementos em uma permutação pode resultar em uma permutação que requer mais operações para ser ordenada.

Seja b_1 uma aresta preta de C, e sejam b_2 e b_3 as arestas pretas ligadas a b_1 por uma aresta cinza. Seja g a aresta cinza ligada a b_2 mas não b_1. Desta forma, quebramos as arestas b_3 e g para inserir dois novos vértices no grafo. Seja $b_3 = (v_b, w_b)$ e $g = (w_g, v_g)$ como mostra a Figura 2.5. Remova b_3 e g e adicione dois novos vértices, v e w. Se b_3 é uma aresta preta positiva, adicione as arestas pretas (v_b, v) e (w, w_b) e as arestas cinzas (v, v_g) e (w_g, w). Se b_3 é uma aresta preta negativa, adicione as arestas pretas (v, v_b) e (w, w_b) e as arestas cinzas (v_g, v) e (w_g, w). Com isso, o ciclo C de tamanho $k > 3$ foi transformado em dois ciclos de tamanhos 3 e $k - 2$.

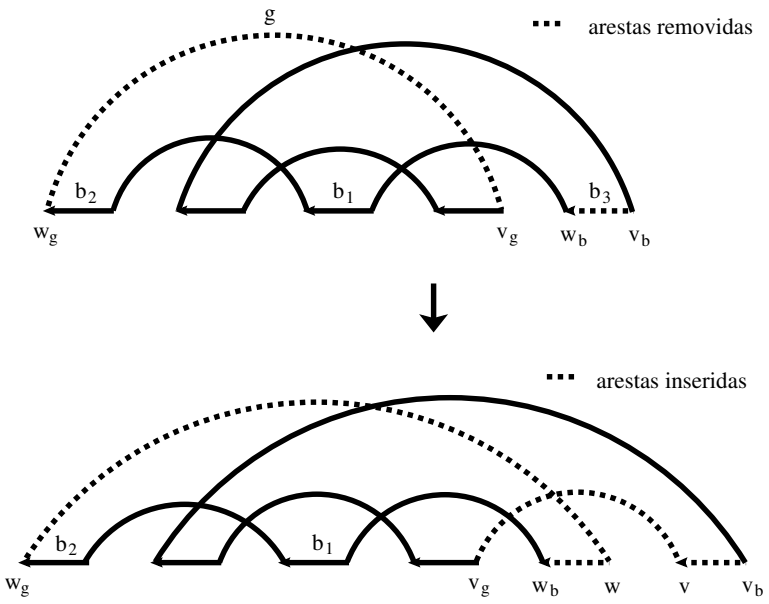


Figura 2.5: Quebra de um ciclo longo de tamanho $k = 5$ em dois ciclos de tamanhos 3 e $k - 2 = 3$, respectivamente.

A Figura 2.6 mostra a transformação da permutação $\pi = (+4\ +3\ -2\ +1)$ em uma permutação simples. Temos que $b_3 = (v_b, w_b) = (+3, +2)$, e $g = (w_g, v_g) = (+1, -2)$. Removemos as arestas b_3 e g e adicionamos os vértices v e w. Uma vez que b_3 era uma aresta negativa, adicionamos as arestas pretas $(v, v_b) = (v, +3)$ e $(w, w_b) = (w, +2)$, e as arestas cinzas $(v_g, v) = (-2, v)$ e $(w_g, w) = (+1, w)$. Desta forma, ao rotular $\hat{\pi}$ de acordo com o novo grafo de ciclos obtemos a permutação $\hat{\pi} = (+5\ +4\ -2\ -3\ +1)$. Note que $|\hat{\pi}| = |\pi| + 1$, $G(\hat{\pi})$ possui uma aresta preta a mais que $G(\pi)$ e $c(\hat{\pi}) = c(\pi) + 1$.

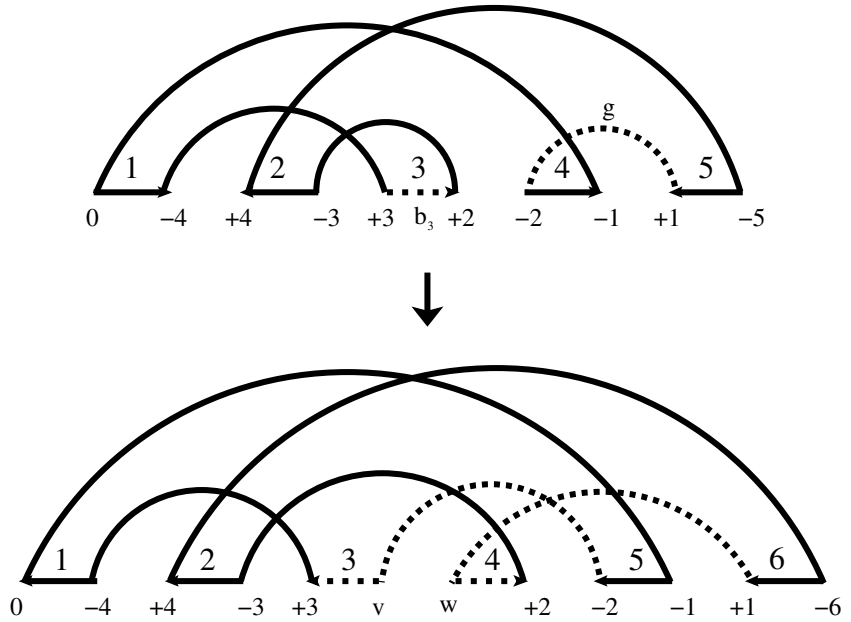


Figura 2.6: Transformação da permutação $\pi = (+4\ +3\ -2\ +1)$ em uma permutação simples $\hat{\pi} = (+5\ +4\ -2\ -3\ +1)$.

2.4 Reversão

Uma reversão $\rho_r(i, j)$ é uma operação que inverte a ordem e a orientação dos elementos entre os pontos i e j de uma permutação.

Definição 2.1. Uma reversão sem sinal $\rho_r(i, j)$, com $1 \leq i \leq j \leq n$, aplicada à permutação $\pi = (\pi_1\ ...\ \pi_{i-1}\ \pi_i\ ...\ \pi_j\ \pi_{j+1}\ ...\ \pi_n)$, gera a permutação $\pi \cdot \rho_r(i, j) = (\pi_1\ ...\ \pi_{i-1}\ \underline{\pi_j\ ...\ \pi_i}\ \pi_{j+1}\ ...\ \pi_n)$, ou seja, ocorre a inversão do segmento $\pi_i, ..., \pi_j$ de π.

Definição 2.2. Uma reversão com sinal $\overline{\rho}_r(i, j)$, com $1 \leq i \leq j \leq n$, aplicada à permutação $\pi = (\pi_1\ ...\ \pi_{i-1}\ \pi_i\ ...\ \pi_j\ \pi_{j+1}\ ...\ \pi_n)$, gera a permutação $\pi \cdot \overline{\rho}_r(i, j) = (\pi_1\ ...\ \pi_{i-1}\ \underline{-\pi_j\ ...\ -\pi_i}\ \pi_{j+1}\ ...\ \pi_n)$, ou seja, ocorre a inversão do segmento $\pi_i, ..., \pi_j$ de π e também a inversão dos sinais dos elementos deste segmento.

Quando não se conhece a orientação dos genes, temos o *Problema de Ordenação por Reversões*, que foi provado ser NP-Completo por Caprara [10]. Além disso, Berman e Karpinski [6] mostraram que este problema não é aproximável por um fator menor que 1.0008.

Kececioglu e Sankoff [30] em 1995 apresentaram o primeiro algoritmo de aproximação, com fator 2.0, consistindo basicamente em remover a maior quantidade de breakpoints a cada passo, dando prioridade para as reversões que ainda deixam strips decrescentes. Bafna e Pevzner [2] introduziram a idéia do *grafo de breakpoints*, o que resultou em um algoritmo de aproximação de fator 1.75. Posteriormente, Christie [13] apresentou a estrutura *grafo de reversões* e um algoritmo de aproximação com fator 1.5. Atualmente, o melhor algoritmo conhecido para o problema possui fator de aproximação 1.375, proposto por Berman e coautores [5] em 2002.

Note que uma reversão $\rho_r(i,j)$ corta dois pontos da permutação: entre (π_{i-1}, π_i) e (π_j, π_{j+1}). Sendo assim, uma reversão pode criar ou remover até 2 breakpoints, bem como deixar o número de breakpoints inalterado e, portanto, $\Delta b_r(\pi, \rho_r) = b_r(\pi \cdot \rho_r) - b_r(\pi) \in \{-2, -1, 0, 1, 2\}$. Um limitante inferior para a distância, proposto por Bafna e Pevzner [2], pode ser derivado desta propriedade: $d_r(\pi) \geq \frac{b_r(\pi)}{2}$. Além disso, Kececioglu e Sankoff [30] mostraram um algoritmo que ordena uma permutação com até $b_r(\pi) - 1$ reversões e, desta forma, temos como limitante superior $d_r(\pi) \leq b_r(\pi) - 1$. Já em relação ao diâmetro, Bafna e Pevzner [2] provaram que $D_r(n) = n - 1$.

Uma variação deste problema, bastante estudada na literatura, consiste em um modelo de rearranjo em que as reversões sempre ocorrem no prefixo da permutação, conhecido como Problema de Ordenação de Panquecas [22, 29], onde a idéia é, dada uma pilha de panquecas de tamanhos diferentes em uma bandeja, rearranjá-las de forma que a menor fique no topo, a segunda menor logo abaixo, e assim por diante até que a maior panqueca fique na parte inferior da bandeja. Atualmente, o melhor algoritmo conhecido para este problema, proposto por Fischer e Ginzinger [21], possui fator de aproximação 2.0. Uma *reversão de prefixo sem sinal* $\rho_{pr}(j)$ é uma reversão sem sinal $\rho_r(1,j)$ para $1 < j \leq n$. A complexidade deste problema era desconhecida até 2011, quando Bulteau, Fertin e Rusu [8] provaram que ele é NP-Difícil com uma redução do problema 3-SAT.

Para o caso em que a orientação dos genes é conhecida, denominado *Problema de Ordenação por Reversões com Sinal*, existem algoritmos polinomiais exatos, sendo que o primeiro foi apresentado por Hannenhalli e Pevzner [25], com complexidade $O(n^4)$. Atualmente, o algoritmo mais eficiente disponível na literatura possui complexidade de tempo $O(n^{\frac{3}{2}}\sqrt{\lg n})$ [37]. Para o caso em que se deseja saber apenas o valor da distância de reversão de permutações, há também um algoritmo com complexidade de tempo $O(n)$ [1].

Existe também uma variação deste problema em que as reversões sempre ocorrem no prefixo da permutação com sinais, conhecido como Problema de Ordenação de Panquecas Queimadas. Esta variação foi introduzida por Gates e Papadimitriou [22], e a idéia é que, além de ordenadas, as panquecas devem estar com o lado queimado voltado para baixo. Uma *reversão de prefixo com sinal* $\overline{\rho}_{pr}(j)$ é uma reversão com sinal $\overline{\rho}_r(1,j)$ com $1 \leq j \leq n$. A complexidade deste problema ainda é desconhecida, e o melhor algoritmo conhecido possui aproximação 2.0, proposto por Cohen e Blum [15].

2.5 Transposição

Uma transposição é um evento de rearranjo que tem a propriedade de trocar dois blocos adjacentes de lugar.

Definição 2.3. A transposição $\rho_t(i,j,k)$, com $1 \leq i < j < k \leq n+1$, aplicada à permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_{i-1}\ \underline{\pi_i\ ...\ \pi_{j-1}}\ \underline{\pi_j\ ...\ \pi_{k-1}}\ \pi_k\ ...\ \pi_n)$, gera a permutação $\pi \cdot \rho_t(i,j,k) = (\pi_1\ \pi_2\ ...\ \pi_{i-1}\ \underline{\pi_j\ ...\ \pi_{k-1}}\ \underline{\pi_i\ ...\ \pi_{j-1}}\ \pi_k\ ...\ \pi_n)$, ou seja, o segmento $\pi_i, \pi_{i+1}, ...\pi_{j-1}$ é removido e inserido após o segmento $\pi_j, \pi_{j+1}, ..., \pi_{k-1}$.

O primeiro algoritmo de aproximação para o *Problema de Ordenação por Transposições* foi apresentado por Bafna e Pevzner [3], com complexidade de tempo de $O(n^2)$ e razão de aproximação 1.5. Dentre as contribuições do trabalho de Bafna e Pevzner, é possível citar uma estrutura em grafos chamada de *Grafo de Ciclos de Transposição*, denotado por $G_t(\pi)$, com uma estrutura muito semelhante a $G(\pi)$. Esta estrutura permite a definição de limitantes inferiores e superiores mais adequados para o problema da distância de transposição.

Christie [14] propôs um algoritmo com fator de aproximação 1.5 mais simples que o de Bafna e Pevzner, mas com complexidade $O(n^4)$. Walter, Dias e Meidanis [40] desenvolveram um algoritmo com complexidade de tempo da ordem de $O(n^2)$, com fator de aproximação 2.25. Elias e Hartman [19] desenvolveram um algoritmo de aproximação na razão de 1.375 e complexidade $O(n^2)$, um avanço na razão de aproximação que não ocorria desde a publicação do trabalho de Bafna e Pevzner, oito anos antes.

Em 2011, Bulteau, Fertin e Rusu provaram que o problema da distância de transposição é NP-Difícil [9] com uma redução do problema SAT, resolvendo essa questão que permaneceu em aberto durante aproximadamente 15 anos. Além disso, eles provaram que dada uma permutação π, descobrir se é possível ordenar π com exatamente $\frac{b_t(\pi)}{3}$ transposições também é NP-difícil.

Note que uma transposição $\rho_t(i,j,k)$ corta três pontos da permutação: entre (π_{i-1}, π_i), (π_{j-1}, π_j) e (π_{k-1}, π_k). Deste modo, uma transposição pode criar, manter ou remover até 3 breakpoints e, assim, $\Delta b_t(\pi, \rho_t) = b_t(\pi \cdot \rho_t) - b_t(\pi) \in \{-3, -2, -1, 0, 1, 2, 3\}$. Assim, um limitante inferior para a distância é $d_t(\pi) \geq \frac{b_t(\pi)}{3}$, proposto por Bafna e Pevzner [3]. Além disso, sempre é possível encontrar uma transposição que diminua em pelo menos 1 o número de breakpoints se $\pi \neq \iota$. Logo, temos o limite superior para a distância: $d_t(\pi) \leq b_t(\pi) - 2$, uma vez que a permutação pode ter $n+1$ breakpoints, onde n é o tamanho da permutação, e a última transposição sempre remove 3 breakpoints. Estes limitantes utilizam apenas o conceito de breakpoints. Limitantes mais precisos, propostos por Bafna e Pevzner [3], consideram o número de ciclos ímpares no grafo de ciclos da permutação π: $\frac{n+1-c_{impar}(G(\pi))}{2} \leq d_t(\pi) \leq \frac{3(n+1-c_{impar}(G(\pi)))}{4}$. Em relação ao diâmetro, Bafna e Pevzner [3] provaram o limitante inferior, $\lfloor \frac{n+1}{2} \rfloor \leq D_t(n)$, e Eriksson e coautores [20] provaram o limitante superior, $D_t(n) \leq \lfloor \frac{2n-2}{3} \rfloor$.

Existe uma variação deste problema que considera apenas transposições de prefixo, cuja complexidade é desconhecida. Uma *transposição de prefixo* $\rho_{pt}(j,k)$ é uma transposição $\rho_t(1,j,k)$ para $1 < j < k \leq n+1$. Esta variação foi introduzida em 2002 por Dias e Meidanis [18] que também apresentaram um algoritmo com fator de aproximação 2.0, sendo esta aproximação a melhor conhecida atualmente.

2.6 O Problema da Ordenação de Permutações por Reversões e Transposições

Este problema permite que tanto os eventos de reversão, definidos na Seção 2.4, quanto os eventos de transposição, definidos na Seção 2.5, ocorram durante a ordenação de uma permutação qualquer π. Uma investigação preliminar desse problema foi realizada por Blanchette, Kunisawa e Sankoff [7]. Existem duas versões para este problema, quando considera-se permutações com sinais e quando considera-se permutações sem sinais. Ambas as versões possuem complexidade desconhecida.

Para o caso de permutações com sinais, Walter, Dias e Meidanis [39] apresentaram um algoritmo de aproximação com razão 2.0 e limitantes para o diâmetro. Meidanis, Walter e Dias [34] apresentaram limitantes para a distância. Gu, Peng e Sudborough [23] acrescentaram o evento de transversão ao problema. No evento de transversão, um bloco é destacado do genoma e inserido em outra posição, mas com a ordem e a orientação invertidas. Formalmente, seja $\rho_t(i, j, k)$ uma transreversão. Então, $\pi \cdot \rho_t(i, j, k) = (\pi_1 \ ... \ \pi_{i-1} \ -\pi_{k-1} \ ... \ -\pi_j \ \pi_i \ ... \ \pi_{j-1} \ \pi_k \ ... \ \pi_n)$. Gu, Peng e Sudborough [23] apresentaram um algoritmo de aproximação na razão 2.0 quando as três operações são permitidas. Lin e Xue [32] apresentaram um algoritmo com fator de aproximação 1.75 quando, além das três operações citadas anteriormente, é adicionada também a operação *RevRev*, que aplica reversões em dois blocos consecutivos da permutação em apenas uma operação. Mais tarde, Hartman e Sharan [27] melhoraram esta razão para 1.5. Estas variações também possuem complexidade desconhecida.

Para o caso de permutações sem sinais, Walter, Dias e Meidanis [39] apresentaram um algoritmo de aproximação com razão 3. Em 2008, Rahman, Shatabda e Hasan [35] apresentaram limitantes para a distância e um algoritmo de aproximação com fator $2k$, onde k é o fator de aproximação do algoritmo utilizado para a decomposição de ciclos. Dado o melhor valor de k conhecido [31], o fator de aproximação deste algoritmo torna-se $2.8334 + \epsilon$, para qualquer $\epsilon > 0$. Lintzmayer e Dias [33] apresentaram limitantes para o diâmetro deste problema.

Sharmin e coautores [36] estudaram a variação do problema que considera apenas reversões de prefixo e transposições de prefixo, conhecida como Problema de Ordenação de Panquecas com Duas Espátulas, e desenvolveram um algoritmo 3-aproximado. Recentemente, Dias e Dias [17] desenvolveram um algoritmo 2-aproximado, sendo esta a melhor aproximação conhecida. A complexidade deste problema não é conhecida.

2.6.1 Limitantes para o Problema da Ordenação de Permutações por Reversões e Transposições

Bafna e Pevzner [3] provaram que, dado o grafo de ciclos $G_t(\pi)$, $\Delta c(\pi, \rho_t) = c(\pi \cdot \rho_t) - c(\pi) \in \{2, 0, -2\}$ e $\Delta c_{impar}(\pi, \rho_t) = c_{impar}(\pi \cdot \rho_t) - c_{impar}(\pi) \in \{2, 0, -2\}$. Assim, esta prova também é válida para $G(\pi)$ se considerarmos transposições $\rho_t(i, j, k)$ nas quais as arestas i, j e k que estão em um mesmo ciclo C do grafo possuem o mesmo sinal, o que nos leva aos lemas 2.4 e 2.5.

Lema 2.4. Seja $\rho_t(i,j,k)$ uma transposição tal que qualquer par de arestas i,j ou k que estão em um mesmo ciclo C de $G(\pi)$ possuem o mesmo sinal. Então, $\Delta c(\pi,\rho_t) \in \{2,0,-2\}$.

Lema 2.5. Seja $\rho_t(i,j,k)$ uma transposição tal que qualquer par de arestas i,j ou k que estão em um mesmo ciclo C de $G(\pi)$ possuem o mesmo sinal. Então, $\Delta c_{impar}(\pi,\rho_t) \in \{2,0,-2\}$.

Os lemas 2.6 e 2.7 lidam com casos em que a transposição é aplicada em um par de arestas pretas de um mesmo ciclo C com sinais diferentes.

Lema 2.6. Seja $\rho_t(i,j,k)$ uma transposição agindo em duas arestas de um mesmo ciclo C de $G(\pi)$ que possuem sinais diferentes. Então, $\Delta c(\pi,\rho_t) = c(\pi\cdot\rho_t) - c(\pi) \in \{1,0,-1\}$.

Demonstração. Seja $\rho_t(i,j,k)$ uma transposição agindo nas arestas pretas p, q e r, a serem definidas a seguir. Seja $p=(v_{p1},v_{p2})$ e $q=(v_{q1},v_{q2})$ duas arestas pretas com sinais diferentes de um mesmo ciclo C, e $|p|>|q|$. Assim, $C=(...,v_{q2},v_{q1},...,v_{p1},v_{p2},...)$, ou $C=(...,v_{p2},v_{p1},...,v_{q1},v_{q2},...)$. Se a aresta preta $r=(v_{r1},v_{r2})$ não pertence a C não precisamos fazer restrições a esta aresta. Caso contrário, escolhemos as arestas pretas p e q de modo que r não está no caminho que liga v_{p1} a v_{q1}. A transposição $\rho_t(i,j,k)$ criará novas arestas pretas $i'=(v_{p1},v_{q2}), j'=(v_{r1},v_{p2})$ e $k'=(v_{q1},v_{r2})$. Neste caso, as arestas pretas i' e j' estão no mesmo ciclo pois ρ_t não quebra a ligação entre v_{p1} e v_{q1}, o que nos leva a conclusão que ρ_t removerá pelo menos um ciclo de $G(\pi)$ e irá adicionar no máximo dois novos ciclos em $G(\pi)$. □

Lema 2.7. Seja $\rho_t(i,j,k)$ uma transposição agindo em duas arestas de um mesmo ciclo de $G(\pi)$ que possuem sinais diferentes. Então, $\Delta c_{impar}(\pi,\rho_t) \in \{2,0,-2\}$.

Demonstração. O Lema 2.6 mostra que um ou dois ciclos serão removidos de $G(\pi)$ e no máximo dois novos ciclos serão adicionados à $G(\pi)$. Desta forma, se assumirmos que o ciclo removido é par e os ciclos adicionados são ímpares, temos que $\Delta c_{impar}(\pi,\rho_t)=2$, no máximo. □

Hannenhalli e Pevzner [26] mostraram que uma reversão qualquer ρ_r pode aumentar em uma unidade, manter, ou diminuir em uma unidade o número de ciclos de $G(\pi)$, o que nos leva ao Lema 2.8. O Lema 2.9 mostra o efeito de ρ_r no número de ciclos ímpares do grafo.

Lema 2.8. Para qualquer reversão $\rho_r(i,j)$ aplicada em duas arestas do grafo $G(\pi)$, $\Delta c(\pi,\rho_r) = c(\pi\cdot\rho_r) - c(\pi) \in \{1,0,-1\}$.

Lema 2.9. Para qualquer reversão $\rho_r(i,j)$ aplicada em duas arestas do grafo $G(\pi)$, $\Delta c_{impar}(\pi,\rho_r) = c_{impar}(\pi\cdot\rho_r) - c_{impar}(\pi) \in \{2,0,-2\}$.

Demonstração. Se i e j são arestas pretas de ciclos diferentes C_1 e C_2, então a reversão irá remover os ciclos C_1 e C_2 de $G(\pi)$ e adicionar um novo ciclo C_3 tal que $|C_3| = |C_1| + |C_2|$. Se C_1 e C_2 são ímpares, então o novo ciclo será par e, desta forma, $\Delta c_{impar}(\pi,\rho_r) = -2$.

Se i e j são arestas de um mesmo ciclo C, então a reversão ρ_r irá remover este ciclo e adicionar um ou dois novos ciclos. Se apenas um ciclo é adicionado, então $\Delta c_{impar}(\pi,\rho_r) =$

0, uma vez que este ciclo possui o mesmo tamanho do ciclo removido. Senão, se C é ímpar, então serão adicionados um ciclo par e um ciclo ímpar e, desta forma, $\Delta c_{impar}(\pi, \rho_r) = 0$. Por último, se C é par, então os dois ciclos adicionados devem ser ímpares e, assim, $\Delta c_{impar}(\pi, \rho_r) = 2$. □

Os lemas 2.4, 2.6 e 2.8 implicam no Lema 2.10

Lema 2.10. Dado $G(\pi)$, $\Delta c(\pi, \rho_t), \Delta c(\pi, \rho_r) \in \{2, 1, 0, -1, -2\}$.

Os lemas 2.5, 2.7 e 2.9 implicam no Lema 2.11.

Lema 2.11. Dado $G(\pi)$, $\Delta c_{impar}(\pi, \rho_t), \Delta c_{impar}(\pi, \rho_r) \in \{2, 0, -2\}$.

Como $c_{impar}(\iota) = n + 1$, as operações aplicadas em uma permutação $\pi \neq \iota$ devem aumentar o número de ciclos ímpares de $c_{impar}(\pi)$ até $n + 1$. O Lema 2.11 implica no limitante inferior para a distância de ordenação de permutações com sinais, denotado por $d_{\overline{rt}}(\pi)$, apresentado no Teorema 2.12.

Teorema 2.12. $d_{\overline{rt}}(\pi) \geq \frac{n+1-c_{impar}(\pi)}{2}$.

Note que o Teorema 2.12 se aplica quando temos uma decomposição de ciclos exata para uma permutação, ou seja, quando se trata de uma permutação com sinais. No caso de permutações sem sinais, um limitante existente utiliza o conceito de breakpoints de reversão. Note que uma reversão corta uma permutação em dois pontos e uma transposição, em três. Desta forma, é possível remover no máximo três breakpoints a cada operação. Um limitante inferior trivial é apresentado no Teorema 2.13

Teorema 2.13. $d_{rt}(\pi) \geq \frac{b_r(\pi)}{3}$.

Os capítulos 3 e 4 apresentam uma série de heurísticas para o Problema da Ordenação de Permutações por Reversões e Transposições. Os limitantes definidos aqui serão utilizados para definir e verificar fatores de aproximação para heurísticas criadas, bem como para comparar os fatores de aproximação obtidos pelas heurísticas criadas com os fatores de aproximação obtidos por algoritmos existentes na literatura.

Capítulo 3

Heurísticas básicas

Neste capítulo, iremos apresentar heurísticas criadas para o Problema da Ordenação de Permutações por Reversões e Transposições. As heurísticas foram criadas com a utilização de uma ou mais métricas clássicas utilizadas em Problemas de Ordenação. Além disso, cada heurística calcula um *score* para cada operação possível, a fim de decidir qual operação será aplicada na permutação.

3.1 Códigos Esquerdo e Direito

Esta heurística baseia-se no trabalho de Gagné e Hamel [4] sobre O Problema de Ordenação de Permutações por Transposições. Os autores criaram uma métrica que é utilizada em um algoritmo de 3-aproximação para ordernar permutações por transposições.

Dada uma permutação π sem sinais, computamos dois códigos para cada elemento π_i da permutação. O *código esquerdo* de um elemento na posição π_i é o número de elementos à esquerda de π_i maiores que ele. Da mesma forma, o *código direito* de um elemento na posição π_i é o número de elementos à direita de π_i menores que ele. Formalmente, temos:

Definição 3.1. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$, o código esquerdo do elemento π_i, denotado por $lc(\pi_i)$, é definido como:

$$lc(\pi_i) = |\{\pi_j | \pi_j > \pi_i \text{ e } 0 \leq j \leq i-1\}|, \text{ para } 1 \leq i \leq n.$$

Definição 3.2. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$, o código direito do elemento π_i, denotado por $rc(\pi_i)$, é definido como:

$$rc(\pi_i) = |\{\pi_j | \pi_j < \pi_i \text{ e } i+1 \leq j \leq n+1\}|, \text{ para } 1 \leq i \leq n.$$

Por exemplo, para $\alpha = (6\ 3\ 2\ 1\ 4\ 5)$, temos que $lc(\alpha_2) = 1$, uma vez que existe um elemento à esquerda de $\alpha_2 = 3$ maior que ele (o elemento $\alpha_1 = 6$). Já $rc(\alpha_2) = 2$, pois existem dois elementos à direita de α_2 menores que ele (os elementos $\alpha_3 = 2$ e $\alpha_4 = 1$).

O código esquerdo de uma permutação π, denotado por $lc(\pi)$, é então definido como a sequência de códigos esquerdos de seus elementos, ou seja, $lc(\pi) = (lc(\pi_1)\ lc(\pi_2)\ ...\ lc(\pi_n))$. Da mesma forma, o código direito de uma permutação π, denotado por $rc(\pi)$, é definido como a sequência de códigos direitos de seus elementos, ou seja, $rc(\pi) = (rc(\pi_1)\ rc(\pi_2)\ ...\ rc(\pi_n))$.

Podemos observar que a permutação identidade ι é a única permutação na qual $lc(\iota) = rc(\iota) = (0\ \ 0\ \ ...\ \ 0)$, uma vez que para qualquer elemento ι_i da permutação ι não existem elementos à esquerda de π_i maiores que ele, nem elemenos à direita de π_i menores que ele. Abaixo temos um exemplo dos códigos esquerdo e direito para a permutação α dada acima.

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

Dado um código qualquer de uma permutação π, chamamos de *platô* uma sequência maximal de elementos do código que possuem o mesmo valor não-nulo. O número de platôs de um código qualquer $c(\pi)$ é denotado por $pl(c(\pi))$, onde $c(\pi)$ pode ser tanto $lc(\pi)$ quanto $rc(\pi)$.

Definição 3.3. Dados os códigos esquerdo e direito de uma permutação π, denotamos por $pl(\pi)$ o mínimo entre $pl(lc(\pi))$ e $pl(rc(\pi))$.

Para o exemplo da permutação α apresentada acima, $pl(lc(\alpha)) = 4$, uma vez que temos as seguintes sequências de mesmo valor não-nulas: (1), (2), (3) e (1 1). Já $pl(rc(\alpha)) = 3$, pois temos as seguintes sequências de mesmo valor não-nulas: (5), (2) e (1). Desta forma, $pl(\alpha) = min(pl(lc(\alpha)), pl(rc(\alpha))) = min(4, 3) = 3$. A permutação ι é a única permutação com $pl(\iota) = 0$, já que temos $min(pl(lc(\iota)), pl(rc(\iota))) = min(0, 0) = 0$. Os seguintes lemas foram apresentados por Gagné e Hamel [4]:

Lema 3.4. Dada uma permutação $\pi = (\pi_1\ \ \pi_2\ \ ...\ \ \pi_n)$ sem sinais, o platô mais à esquerda de $lc(\pi)$ pode ser removido por uma transposição envolvendo os elementos deste platô sem criar novos platôs em $lc(\pi)$. Similarmente, o platô mais à direita de $rc(\pi)$ pode ser removido com uma transposição envolvendo os elementos deste platô sem criar novos platôs em $rc(\pi)$.

Demonstração. Suponha que o platô mais à esquerda de $lc(\pi)$ está localizado entre as posições i e $j - 1$. Por definição, todos os códigos de elementos à esquerda de i são iguais a 0 e os elementos $\pi_1, \pi_2, ..., \pi_{i-1}$ estão em ordem crescente. Então, se $lc(\pi_i) = v$, a transposição $\rho_t(i, j, k)$ com $k = i - v$ remove o primeiro platô sem criar novos platôs. Similarmente, suponha que o platô mais à direita de $rc(\pi)$ está localizado entre as posições i e $j-1$. Então todos os códigos de elementos à direita de $j-1$ são iguais a 0 e os elementos $\pi_j, \pi_{j+1}, ..., \pi_n$ estão em ordem crescente. Então, se $rc(\pi_i) = v$, a transposição $\rho_t(i, j, k)$, com $k = j + v$ remove o último platô sem criar novos platôs. □

Lema 3.5. Dada uma permutação $\pi = (\pi_1\ \ \pi_2\ \ ...\ \ \pi_n)$ sem sinais, $d_t(\pi) \leq pl(\pi)$.

Demonstração. Sabemos que $pl(\pi)$ contém o número de platôs de π. Aplique a transposição descrita no Lema 3.4 removendo, a cada passo, um platô. Após aplicar $pl(\pi)$ operações, $pl(\pi) = 0$ e, desta forma, $\pi = \iota$ e $d_t(\pi) \leq pl(\pi)$. □

Para permutações com sinais, não é verdade que sempre é possível remover um platô sem criar novos platôs, bem como podem existir permutações σ com número $pl(\sigma) = 0$ e

$\sigma \neq \iota$. Por exemplo, dada a permutação $\sigma = (-4\ +1\ +2\ +3)$, $pl(\sigma) = 0$ uma vez que temos $lc(\sigma) = (0\ 0\ 0\ 0)$ e $rc(\sigma) = (0\ 0\ 0\ 0)$, pois, para qualquer elemento σ_i de σ não existem elementos à esquerda maiores que σ_i nem elementos à direita menores que σ_i.

Definição 3.6. Dada uma permutação π, $score_{lc+rc}(\pi) = pl(u(\pi)) + neg(\pi)$.

Desta forma, definimos então a pontuação de uma permutação π como $score_{lc+rc}(\pi) = pl(u(\pi)) + neg(\pi)$. Por exemplo, para a permutação σ dada acima, temos que $u(\sigma) = (4\ 1\ 2\ 3)$, $lc(u(\sigma)) = (0\ 1\ 1\ 1)$ e $rc(u(\sigma)) = (3\ 0\ 0\ 0)$ e, desta forma, $pl(u(\sigma)) = 1$. Assim, $score_{lc+rc}(\sigma) = pl(u(\sigma)) + neg(\sigma) = 1 + 1 = 2$.

Lema 3.7. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível diminuir o valor de $score_{lc+rc}(\pi)$ com uma operação.

Demonstração. Se π é uma permutação sem sinais, o valor de $neg(\pi)$ será sempre nulo, e $u(\pi) = \pi$. Desta forma, temos que $score_{lc+rc}(\sigma) = pl(\sigma)$, e o Lema 3.4 garante que sempre é possível diminuir seu valor. Se π é uma permutação com sinais, e enquanto $pl(u(\pi)) > 0$, aplique a operação descrita no Lema 3.4. Como a transposição não troca os sinais dos elementos envolvidos, o valor de $neg(\pi)$ não será alterado com esta operação, garantindo assim que $score_{lc+rc}$ vai necessariamente diminuir. Caso $pl(u(\pi)) = 0$ e $\pi \neq \iota$, temos que $neg(\pi) > 0$. Para diminuir o valor de $score_{lc+rc}(\pi)$, basta aplicar uma reversão unitária ρ_r em algum elemento negativo de π. Esta reversão fará com que $neg(\pi \cdot \rho_r) = neg(\pi) - 1$, e o valor de $pl(u(\pi))$ não será alterado, uma vez que $u(\pi \cdot \rho_r) = u(\pi)$. □

O valor de $score_{lc+rc}(\pi)$ somente será nulo quando $pl(u(\pi)) = 0$, ou seja, quando $u(\pi) = \iota$, e $neg(\pi) = 0$, implicando assim em $\pi = \iota$. Logo, $score_{lc+rc}(\pi) = 0 \iff \pi = \iota$.

Assim, a heurística irá testar todas as reversões e transposições possíveis, e aplicará a operação que resulte no menor valor de $score_{lc+rc}$ a cada passo. A heurística ordena a permutação uma vez que o Lema 3.7 garante que sempre é possível diminuir o valor de $score_{lc+rc}$, até que $score_{lc+rc}$ seja igual a 0 e, desta forma, $\pi = \iota$.

A complexidade de tempo da função *NumeroPlatos* é $\mathcal{O}(n^2)$: calcular a lista de códigos esquerdo e direito pode ser realizado em $\mathcal{O}(n^2)$, e calcular o número de platôs esquerdo e direito custa $\mathcal{O}(n)$.

A complexidade de tempo total da heurística é de $\mathcal{O}(n^6)$, uma vez que existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, aplicar cada operação custa $\mathcal{O}(n)$, e calcular o valor de $score_{lc+rc}$ custa $\mathcal{O}(n^2)$. Como são necessários no máximo $\mathcal{O}(n)$ passos para ordenar a permutação, a complexidade total da heurística é $\mathcal{O}(n^6)$.

Função 1 $NumeroPlatos(\pi)$ (Complexidade: $\mathcal{O}(n^2)$)

1: $lc \leftarrow$ lista de códigos esquerdos de π
2: $rc \leftarrow$ lista de códigos direitos de π
3: $pl \leftarrow$ número de platôs em lc
4: $pr \leftarrow$ número de platôs em rc
5: **return** $min(pl, pr)$

Heurística 2 $lc + rc(\pi)$ (Complexidade: $\mathcal{O}(n^6)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     best ← ∞
4:     para cada operação ρ válida faça
5:         σ ← π · ρ
6:         p ← NumeroPlatos(u(σ))
7:         score_{lc+rc} ← p + neg(σ)
8:         se best > score_{lc+rc} então
9:             best ← score_{lc+rc}
10:            α ← σ
11:        fim se
12:    fim para
13:    π ← α
14:    dist ← dist + 1
15: fim enquanto
16: return dist
```

3.2 Breakpoints

Esta heurística baseia-se no conceito de breakpoints, apresentado na Seção 2.1. Como é permitido aplicar tanto reversões quanto transposições, quando a permutação não possuir sinais será utilizado o conceito de breakpoints para reversões. A heurística procura remover o maior número possível de breakpoints a cada operação, até que $\pi = \iota$.

Definição 3.8. Dada uma permutação π sem sinais, $score_{BP}(\pi) = b_r(\pi)$.

Lema 3.9. Dada uma permutação sem sinais $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$, com $\pi \neq \iota$, sempre é possível diminuir o valor de $score_{BP}(\pi)$ com uma operação.

Demonstração. A idéia é aumentar a primeira strip da permutação a cada operação, removendo o breakpoint à direita sem introduzir novos breakpoints. A primeira strip da permutação é sempre uma strip crescente. Seja a o maior elemento da primeira strip, localizado na posição i. O elemento $a+1$ deve estar localizado necessariamente à direita de a, na posição $j > i+1$, por definição. Se $a+1$ está no começo de uma strip, ou é o único elemento de uma strip, aplique a transposição $\rho_t(i+1, j, k+1)$, sendo k a posição do último elemento da strip que começa na posição j, de modo a mover esta strip ao lado da primeira strip, como mostra a Figura 3.1. Se seu sucessor está no final de uma strip, aplique a reversão $\rho_r(i+1, j)$, como mostra a Figura 3.2. □

O Lema 3.9 garante que a heurística removerá todos os breakpoints de π até que $score_{BP}(\pi) = 0$ e, consequentemente, $\pi = \iota$.

A heurística busca uma operação a partir da posição atual dos elementos. Primeiramente, verifica a existência de uma transposição que remove três breakpoints. Caso contrário, a heurística verifica a existência de uma operação que remove dois breakpoints. Por último, a heurística busca uma reversão ou uma transposição que remove apenas um

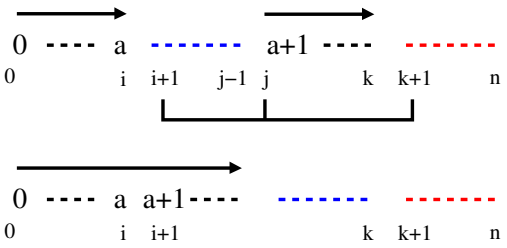


Figura 3.1: Remoção do breakpoint entre as posições i e $i+1$ de uma permutação sem sinais com a transposição $\rho_t(i+1, j, k+1)$.

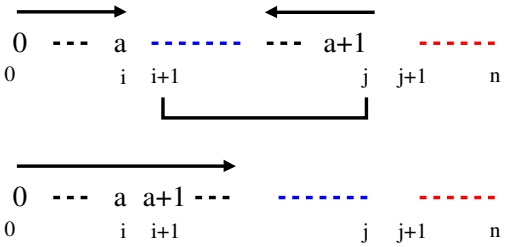


Figura 3.2: Remoção do breakpoint entre as posições i e $i+1$ de uma permutação sem sinais com a reversão $\rho_r(i+1, j)$.

breakpoint, sendo esta operação garantida pelo Lema 3.9. A operação encontrada é então aplicada na permutação.

A complexidade de tempo desta heurística é de $\mathcal{O}(n^2)$, uma vez que procurar uma operação leva $\mathcal{O}(n)$ com o auxílio de um vetor que armazena as posições dos elementos da permutação, e são necessárias $\mathcal{O}(n)$ buscas de operações até que a permutação esteja ordenada. Note que, apesar de ser uma heurística, é garantido que o resultado seja uma 3−aproximação, uma vez que é possível remover até 3 breakpoints com uma operação mas, eventualmente, uma operação removerá apenas 1 breakpoint.

Heurística 3 Breakpoints(π) (Complexidade: $\mathcal{O}(n^2)$)

1: $dist \leftarrow 0$
2: **enquanto** $(\pi \neq \iota)$ **faça**
3: Procure uma transposição que remova 3 breakpoints
4: Caso contrário, procure uma operação que remova 2 breakpoints
5: Caso contrário, encontre a operação que remove 1 breakpoint
6: $\pi \leftarrow \pi \cdot \rho$ tal que ρ seja a operação com menor valor de $score_{BP}(\pi \cdot \rho)$.
7: $dist \leftarrow dist + 1$
8: **fim enquanto**
9: **return** $dist$

Definição 3.10. Dada uma permutação π com sinais, $score_{BP}(\pi) = b_t(\pi)$.

Lema 3.11. Dada uma permutação com sinais π, com $\pi \neq \iota$, sempre é possível diminuir o valor de $score_{BP}$ com uma operação.

Demonstração. A idéia aqui é a mesma do Lema 3.9: aumentar a primeira strip da permutação a cada operação, removendo o breakpoint à direita desta strip sem introduzir

novos breakpoints. A primeira strip da permutação é sempre uma strip crescente. Seja a o maior elemento da primeira strip, que deve estar localizado na última posição da strip e possuir sinal positivo, por definição. Seja i a posição de a na permutação.

Procure a posição do elemento $(a+1)$, que deve estar localizado necessariamente à direita de a. Seja $j > i$ a posição do elemento $(a+1)$. Se o elemento $(a+1)$ possui sinal negativo, então ou ele pertence a uma strip com um único elemento, ou ele é o último elemento de uma strip decrescente, por definição. Aplique a reversão $\rho_r(i+1, j)$, como mostra a Figura 3.3, para remover o breakpoint entre as posições i e $i+1$ (eventualmente, o elemento $-(a+1)$ estará na posição $j = i+1$, então será aplicada uma reversão unitária $\rho_r(j, j)$ apenas invertendo sinal negativo do elemento $(a+1)$). Se o elemento $(a+1)$ possui sinal positivo, então ou ele pertence a uma strip com um único elemento, ou ele é o primeiro elemento de uma strip crescente, por definição. Aplique a transposição $\rho_t(i+1, j, k+1)$, sendo k a posição do último elemento da strip que começa na posição j, de modo a mover esta strip para a posição $i+1$ e remover o breakpoint entre as posições i e $i+1$, como mostra a Figura 3.4. □

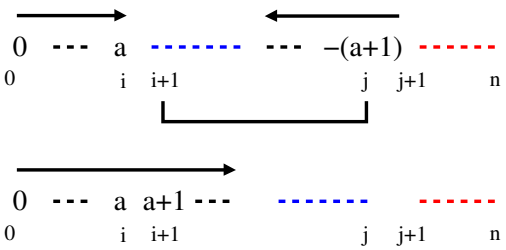


Figura 3.3: Remoção do breakpoint entre as posições i e $i+1$ de uma permutação com sinais com a reversão $\rho_r(i+1, j)$.

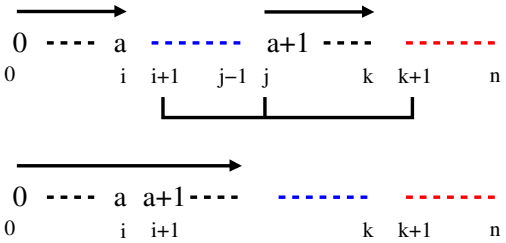


Figura 3.4: Remoção do breakpoint entre as posições i e $i+1$ de uma permutação com sinais com a transposição $\rho_t(i+1, j, k+1)$.

Assim, o Lema 3.11 garante que a heurística removerá pelo menos um breakpoint da permutação com sinais π a cada passo, até que $score_{BP}(\pi) = 0$ e, consequentemente, $\pi = \iota$.

3.3 LIS

Esta heurística baseia-se no *Problema da Subsequência Crescente Máxima (LIS)*. Seja $A = [a_1\ a_2\ ...\ a_n]$ uma sequência qualquer de números inteiros. Uma sequência $B =$

$[b_1\ b_2\ ...\ b_m]$ é dita *subsequência* de A se $B \subseteq A$, $m \leq n$ e $\forall\ b_i, b_j$ com $i < j$, $\exists\ a_k, a_l$ tal que $k < l$, $b_i = a_k$ e $b_j = a_l$. A subsequência B é dita *crescente* se, para qualquer par de elementos adjacentes b_i, b_{i+1}, com $1 \leq i < m$, $b_i < b_{i+1}$.

Assim, o *Problema da Subsequência Crescente Máxima* busca encontrar uma subsequência crescente de elementos tal que o tamanho desta subsequência seja o maior possível. Por exemplo, dada a sequência $A = (4\ 2\ 3\ 5\ 1\ 7\ 6\ 8)$, temos que $B = (2\ 3\ 5\ 1\ 7)$ é uma subsequência de A, mas não é crescente. Já a sequência $C = (2\ 3\ 5\ 7)$, além de subsequência de A é também crescente, mas não é máxima: a sequência $D = (2\ 3\ 5\ 6\ 8)$ é uma subsequência crescente de A e máxima, ou seja, não é possível encontrar outra subsequência crescente com número de elementos maior que 5. Note que pode existir mais de uma subsequência crescente máxima em uma sequência: a subsequência $E = (2\ 3\ 5\ 7\ 8)$ de A também é crescente e máxima.

O Problema da Subsequência Crescente Máxima já foi abordado em trabalhos considerando ordenação de permutações por transposições: Guyer, Heath e Vergara [24] criaram uma heurística baseada na subsequência crescente máxima para ordenar permutações por transposições. Mais tarde, Heat e Vergara [28] mostraram que são necessários no máximo $n - |LIS(\pi)|$ transposições para ordenar uma permutação sem sinais, uma vez que sempre é possível aumentar o tamanho da $LIS(\pi)$ em uma unidade com uma transposição, como mostra o Lema 3.12.

A LIS é uma métrica compatível com o problema de ordenação de permutações pois, quanto maior o tamanho da LIS de uma permutação, mais próxima ela está da permutação ι, uma vez que $LIS(\iota) = \iota$. Assim, o que buscamos ao utilizar a LIS é, a cada operação, aumentar seu tamanho até que, eventualmente, $LIS(\pi) = \pi$ e, desta forma, $\pi = \iota$.

Lema 3.12. Dada uma permutação sem sinais π, com $\pi \neq \iota$, sempre é possível aumentar o tamanho da LIS com uma operação.

Demonstração. Seja a um elemento da permutação π tal que $a \notin LIS(\pi)$. Aplique uma transposição ρ_t que mova o elemento a para a esquerda de b tal que $b \in LIS(\pi)$ e b seja o menor elemento da LIS maior que a. Se não existe $b > a$, aplique uma transposição ρ_t que mova o elemento a para a direita de $c \in LIS(\pi)$ tal que c seja o último elemento da LIS. Com isso, $a \in LIS(\pi \cdot \rho_t)$, e $|LIS(\pi \cdot \rho_t)| > |LIS(\pi)|$. □

Por exemplo, para $\alpha = (1\ 3\ 6\ 2\ 4\ 5)$, obtemos $LIS(\alpha) = (1\ 2\ 4\ 5)$ e, assim, temos os seguintes valores para a:

a	b	c	ρ_t	$\alpha \cdot \rho_t$	$LIS(\alpha \cdot \rho_t)$
3	4	-	$\rho_t(2,3,5)$	$(1\ 6\ 2\ 3\ 4\ 5)$	$(1\ 2\ 3\ 4\ 5)$
6	-	5	$\rho_t(3,4,7)$	$(1\ 3\ 2\ 4\ 5\ 6)$	$(1\ 2\ 4\ 5\ 6)$

Definição 3.13. Dada uma permutação π qualquer, $score_{LIS}(\pi) = |LIS(u(\pi))| - neg(\pi)$.

Para permutações com sinais, não é verdade que sempre é possível aumentar o tamanho da LIS com uma transposição. Desta forma, definimos $score_{LIS}(\pi) = |LIS(u(\pi))| - neg(\pi)$. Por exemplo, para $\sigma = (+2\ -3\ -1\ -4\ +6\ +5)$, temos que $|LIS(u(\sigma))| = |(2\ 3\ 4\ 6)| = 4$. Assim, $score_{LIS}(\sigma) = |LIS(u(\sigma))| - neg(\sigma) = 4 - 3 = 1$.

Lema 3.14. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível aumentar o valor de $score_{LIS}(\pi)$ com uma operação.

Demonstração. Se π é uma permutação sem sinais, o valor de $neg(\pi)$ será sempre nulo, e $u(\pi) = \pi$. Desta forma, temos que $score_{LIS}(\pi) = |LIS(\pi)|$, e o Lema 3.12 garante que sempre é possível aumentar seu valor.

Se π é uma permutação com sinais, e enquanto $|LIS(u(\pi))| < n$, aplique a operação descrita no Lema 3.12. Como a transposição não troca os sinais dos elementos envolvidos, o valor de $neg(\pi)$ não será alterado com esta operação, garantindo assim que $score_{LIS}$ vai necessariamente aumentar. Caso $|LIS(u(\pi))| = n$ e $score_{LIS}(\pi) \neq n$, temos que $neg(\pi) > 0$. Para aumentar o valor de $score_{LIS}(\pi)$, basta aplicar uma reversão unitária ρ_r em algum elemento negativo de π. Esta reversão fará com que $neg(\pi \cdot \rho_r) = neg(\pi) - 1$, e o valor de $|LIS(u(\pi))|$ não será alterado, uma vez que $u(\pi \cdot \rho_r) = u(\pi)$. □

A heurística aplica todas as operações possíveis calculando, para cada π_i resultante, o valor de $score_{LIS}(\pi_i)$. De forma gulosa, é escolhida a operação que resulta no maior valor de $score_{LIS}$ a cada passo. O Lema 3.14 garante que a heurística ordena a permutação, uma vez que sempre é possível aumentar o valor de $score_{LIS}(\pi)$. Dado que o maior valor possível para $|LIS(\pi)|$ é n, quando $\pi = \iota$, e $neg(\pi) \geq 0$, temos que $score_{LIS}(\pi) = n \iff \pi = \iota$.

Função 4 LenLIS(π) (Complexidade: $\mathcal{O}(n \log n)$)

```
1: lis ← |LIS(π)|, calculado utilizando programação dinâmica e busca binária
2: return lis
```

Heurística 5 LIS(π) (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     α ← π
4:     best ← −∞
5:     para cada operação ρ válida faça
6:         σ ← π · ρ
7:         lis ← LenLIS(σ)
8:         score_LIS ← lis − neg(σ)
9:         se best < score_LIS então
10:            best ← score_LIS
11:            α ← σ
12:        fim se
13:    fim para
14:    π ← α
15:    dist ← dist + 1
16: fim enquanto
17: return dist
```

A complexidade de tempo desta heurística é de $\mathcal{O}(n^5 \log n)$: existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π e aplicar cada operação custa $\mathcal{O}(n)$. Calcular o valor de

$score_{LIS}$ de uma permutação custa $\mathcal{O}(n \log n)$, totalizando assim $\mathcal{O}(n^4 \log n)$. Como são necessários no máximo $\mathcal{O}(n)$ passos para ordenar a permutação, a complexidade total da heurística é, então, $\mathcal{O}(n^5 \log n)$.

3.4 LIS+LDS

Esta heurística baseia-se na combinação da Subsequência Crescente Máxima, denominada *LIS*, e da Subsequência Decrescente Máxima, denominada *LDS*. Seja $A = (a_1\ a_2\ ...\ a_n)$ uma sequência de números inteiros. Uma subsequência $B = (b_1\ b_2\ ...\ b_m)$ de A é dita *decrescente* se, para qualquer par de elementos adjacentes b_i, b_{i+1}, com $1 \leq i < m$, $b_i > b_{i+1}$.

Por exemplo, dada a sequência $A = (4\ 2\ 3\ 5\ 1\ 7\ 6\ 8)$, $B = (7\ 6)$ é uma subsequência decrescente de A, mas não é máxima: a sequência $C = (4\ 2\ 1)$, é uma subsequência decrescente de A e máxima, ou seja, não é possível encontrar outra subsequência decrescente com número de elementos maior que 3. Note que também podem existir mais de uma subsequência decrescente máxima de uma sequência: a subsequência $D = (4\ 3\ 1)$ de A também é decrescente e máxima.

A utilização apenas da LIS como métrica para ordenar permutações possui uma grande desvantagem quando podemos aplicar reversões durante a ordenação: se $LDS(\pi)$ é maior que $LIS(\pi)$ seria mais interessante aumentar esta subsequência decrescente e, eventualmente, aplicar uma reversão que transforme esta subsequência decrescente em uma subsequência crescente de elementos.

Lema 3.15. Dada uma permutação sem sinais π, com $\pi \neq \iota_{inv}$, sempre é possível aumentar o tamanho da LDS com uma operação.

Demonstração. Seja a um elemento da permutação π tal que $a \notin LDS(\pi)$. Aplique uma transposição ρ_t que mova o elemento a à esquerda do elemento b tal que $b \in LDS(\pi)$ e b seja o maior elemento da LDS menor que a. Se não existe $b < a$, aplique uma transposição ρ_t que mova o elemento a à direita de $c \in LDS(\pi)$ tal que c seja o último elemento da LDS. Com isso, $a \in LDS(\pi \cdot \rho_t)$, e $|LDS(\pi \cdot \rho_t)| = |LDS(\pi)| + 1$. □

Por exemplo, para $\alpha = (1\ 4\ 5\ 3\ 2\ 6)$, obtemos $LDS(\alpha) = (5\ 3\ 2)$ e, assim, temos os seguintes valores para a:

a	b	c	ρ_t	$\alpha \cdot \rho_t$	$LDS(\alpha \cdot \rho_t)$
1	-	2	$\rho_t(1,2,6)$	(4 5 3 2 1 6)	(5 3 2 1)
4	3	-	$\rho_t(2,3,4)$	(1 5 4 3 2 6)	(5 4 3 2)
6	5	-	$\rho_t(3,6,7)$	(1 4 6 5 3 2)	(6 5 3 2)

O cálculo do tamanho da LDS de uma permutação π é simples: basta inverter todos os elementos de π e chamar a mesma função que realiza o cálculo da LIS. Seja $\pi_{inv} = \pi \cdot \rho_r(1, n)$, onde $n = |\pi|$. Desta forma, $|LDS(\pi)| = |LIS(\pi_{inv})|$.

Definição 3.16. Dada uma permutação π qualquer, $score_{LDS}(\pi) = |LIS(u(\pi_{inv}))| - neg(\pi_{inv})$.

Para permutações com sinais, não é verdade que sempre é possível aumentar o tamanho da LDS com uma transposição. Desta forma, definimos $score_{LDS}(\pi) = |LIS(u(\pi_{inv}))| - neg(\pi_{inv})$. Por exemplo, para $\sigma = (+2\ -3\ -1\ -4\ +6\ +5)$, temos que $|LIS(u(\sigma_{inv}))| = |(1\ 3)| = 2$. Assim, $score_{LDS}(\sigma) = |LIS(u(\sigma_{inv}))| - neg(\sigma_{inv}) = 2 - 3 = -1$.

Lema 3.17. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível aumentar o valor de $score_{LDS}(\pi)$ com uma operação.

Demonstração. Se π é uma permutação sem sinais, o valor de $neg(\pi_{inv})$ será sempre nulo, e $u(\pi) = \pi$. Desta forma, $score_{LDS}(\pi) = |LIS(\pi_{inv})|$, e o Lema 3.12 garante que existe uma operação que aumentará seu valor. Se π é uma permutação com sinais, e enquanto $|LIS(u(\pi_{inv})| < n$, aplique a operação descrita no Lema 3.15. Como a transposição não troca os sinais dos elementos envolvidos, o valor de $neg(\pi_{inv})$ não será alterado com esta operação, garantindo assim que $score_{LDS}$ vai necessariamente aumentar. Caso $|LIS(u(\pi_{inv}))| = n$ e $score_{LDS}(\pi) \neq n$, temos que $neg(\pi_{inv}) > 0$. Para aumentar o valor de $score_{LDS}(\pi)$, basta aplicar uma reversão unitária ρ_r em algum elemento negativo de π_{inv}. Esta reversão fará com que $neg(\pi_{inv} \cdot \rho_r) = neg(\pi_{inv}) - 1$, e o valor de $|LIS(u(\pi_{inv}))|$ não será alterado, uma vez que $u(\pi_{inv} \cdot \rho_r) = u(\pi_{inv})$. □

Definição 3.18. Dada uma permutação π qualquer, $score_{LIS+LDS}(\pi) = max(score_{LIS}(\pi), score_{LDS}(\pi))$.

Lema 3.19. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível aumentar o valor de $score_{LIS+LDS}(\pi)$ com uma operação.

Demonstração. Uma vez que $score_{LIS+LDS} = max(score_{LIS}, score_{LDS})$, e dados os lemas 3.14 e 3.17, é garantido que existe uma operação que aumenta o valor de $score_{LIS+LDS}(\pi)$ também. □

Esta heurística utiliza então a combinação entre LIS e LDS da seguinte forma: aplique todas as operações possíveis em π e calcule $score_{LIS+LDS}$ para cada permutação resultante, e aplique a operação que gerou a permutação com o maior valor de $score_{LIS+LDS}$. Caso exista mais de uma permutação com o maior valor de $score_{LIS+LDS}$, aplique a operação cujo $score_{LIS+LDS} = score_{LIS}$, uma vez que o uso da métrica LDS requer uma operação adicional para ordenar a permutação.

A complexidade deste algoritmo é a mesma que a versão que considera apenas LIS, $\mathcal{O}(n^5 \log n)$: existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, aplicar cada operação custa $\mathcal{O}(n)$ e calcular o valor de $score_{LIS+LDS}$ custa $\mathcal{O}(n \log n)$. Como são necessários no máximo $\mathcal{O}(n)$ passos até que a permutação esteja ordenada, a complexidade total da heurística é $\mathcal{O}(n^5 \log n)$.

Função 6 LenLDS(π) (Complexidade: $\mathcal{O}(n \log n)$)

1: $\pi_{inv} \leftarrow \pi \cdot \rho_r(1, n)$
2: **return** LenLIS(π_{inv})

Heurística 7 $LIS + LDS$ (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     best ← −∞
4:     bestlis ← −∞
5:     para cada operação ρ válida faça
6:         σ ← π · ρ
7:         lis ← LenLIS(σ)
8:         lds ← LenLDS(σ)
9:         score_{LIS+LDS} ← max(lis, lds) − neg(σ)
10:        se (best < score_{LIS+LDS}) ou (best = score_{LIS+LDS} e bestlis < lis) então
11:            best ← score_{LIS+LDS}
12:            bestlis ← lis
13:            α ← σ
14:        fim se
15:    fim para
16:    π ← α
17:    dist ← dist + 1
18: fim enquanto
19: return dist
```

3.5 LIS+BP

Outra abordagem que pode ser explorada é a utilização da métrica LIS em conjunto com a métrica breakpoints na hora de calcular a pontuação para uma operação.

Definição 3.20. Dada uma permutação π qualquer, $score_{LIS+BP}(\pi) = score_{LIS}(\pi) - b(\pi)$.

Lema 3.21. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível aumentar o valor de $score_{LIS+BP}(\pi)$ com uma operação.

Demonstração. Se $|score_{LIS}(\pi)| < |\pi|$, o Lema 3.14 garante que sempre é possível aumentar o valor de $score_{LIS}$ com uma operação. Além disso, é possível aumentar o valor de $score_{LIS}$ sem aumentar o número de breakpoints da permutação, bastando que a transposição não leve apenas o elemento i, mas sim a strip que contém o elemento i. Se o menor elemento i que não faz parte de uma LIS está localizado na mesma strip de um elemento que já faz parte da LIS, então esta strip é necessariamente decrescente. Desta forma, basta aplicar uma reversão nas extremidades desta strip e o elemento i estará presente na LIS, a LIS necessariamente irá aumentar, e o número de breakpoints irá, no máximo, diminuir. Se $|score_{LIS}(\pi)| = |\pi|$, temos que $u(\pi) = \iota$ e $neg(\pi) = 0$ por definição e, desta forma, $\pi = \iota$ e $b(\pi) = 0$. □

Por exemplo, seja a permutação sem sinais $\sigma = (4\ 2\ 3\ 5\ 1\ 7\ 6\ 8)$. Assim, temos que $LIS(\sigma) = (2\ 3\ 5\ 6\ 8)$ e $score_{LIS} = |LIS(\sigma)| = 5$. Além disso, temos que $b_r(\pi) = 6$. Logo, a pontuação desta permutação seria $score_{LIS+BP}(\sigma) = 5 - 6 = -1$.

Já para a permutação com sinais $\alpha = (4\ \ 2\ \ 3\ \ -5\ \ -1\ \ 7\ \ 6\ \ 8)$, temos que $LIS(u(\sigma)) = (2\ \ 3\ \ 5\ \ 6\ \ 8)$ e, desta forma, $score_{LIS} = |LIS(\sigma)| - neg(\sigma) = 5 - 2 = 3$. Além disso, temos que $b_t(\pi) = 7$. Assim, a pontuação desta permutação seria $score_{LIS+BP}(\sigma) = 3 - 7 = -4$.

Heurística 8 LIS+BP(π) (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     best ← −∞
4:     para cada operação ρ válida faça
5:         σ ← π · ρ
6:         score_{LIS+BP} ← LenLIS(σ) − b(σ)
7:         se best < score_{LIS+BP} então
8:             best ← score_{LIS+BP}
9:             α ← σ
10:        fim se
11:    fim para
12:    π ← α
13:    dist ← dist + 1
14: fim enquanto
15: return dist
```

A complexidade de tempo desta heurística é $\mathcal{O}(n^5 \log n)$: existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, aplicar cada operação custa $\mathcal{O}(n)$. Calcular $score_{LIS}$ leva tempo $\mathcal{O}(n \log n)$, e calcular o número de breakpoints da permutação custa $\mathcal{O}(n)$. Como são necessários $\mathcal{O}(n)$ passos até que a permutação esteja ordenada, temos, então que a complexidade total da heurística é $\mathcal{O}(n^5 \log n)$.

3.6 Entropia

Esta heurística baseia-se no nível de "desordem" dos elementos da permutação. Dada uma permutação π, o valor da desordem de um elemento π_i é dado pelo módulo da diferença entre o valor absoluto de π_i e sua posição atual i, ou seja,

$$desordem(\pi_i) = ||\pi_i| - i|$$

O valor da entropia de uma permutação π é a soma da desordem de todos seus elementos, ou seja,

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

.

Lema 3.22. Dada uma permutação π tal que $u(\pi) \neq \iota$, sempre é possível diminuir o valor da entropia com uma operação.

Demonstração. Seja $\pi_k = i$ o menor elemento da permutação tal que $desordem(\pi_k) \neq 0$, e seja $entro(\pi) = W$. Vamos analizar a aplicação da transposição $\rho_t(i, k, k+1)$, que move o elemento i da posição k para a posição i.

Dentre os $(k-i)$ elementos que estão à esquerda de π_k antes de aplicar a transposição, podem existir, no máximo, $(k-i-1)$ elementos em suas posições corretas e, consequentemente, com valor de desordem igual à zero: os elementos entre as posições $(i+1)$ e $(k-1)$. O elemento π_i deve, obrigatoriamente, ter valor de desordem não nula, e $\pi_i > \pi_k$ por definição. Após a aplicação da transposição, todos os elementos com desordem nula que estavam entre as posições $(i+1)$ e $(k-1)$ terão o valor de desordem incrementado em uma unidade, aumentando então em $(k-i-1)$ unidades o valor da entropia de $\pi \cdot \rho_t$. O elemento que estava na posição π_i deve, obrigatoriamente diminuir em uma unidade seu valor de desordem. Como o elemento i estará em sua posição correta após a transposição, seu valor de desordem agora será nulo, diminuindo assim o valor da entropia de $\pi \cdot \rho_t$ em $(k-i)$ unidades.

Assim, $entro(\pi \cdot \rho_t) = entro(\pi) + (k-i-1) - 1 - (k-i) = W - 2$. Para o caso em que o número de elementos entre as posições $(i+1)$ e $(k-1)$ com desordem nula seja menor que $(k-i-1)$, o valor final da entropia será ainda menor que $W-2$, uma vez que, assim como π_i, estes elementos terão seu valor de desordem decrescidos em uma unidade. □

Por exemplo, para a permutação sem sinais $\sigma = (1\ 4\ 3\ 5\ 2\ 6)$, temos os seguintes valores de desordem:

$$\begin{aligned} \sigma &= (1\ 4\ 3\ 5\ 2\ 6) \\ desordem(\sigma) &= (0\ 2\ 0\ 1\ 3\ 0) \\ entro(\sigma) &= 2+1+3=6 \end{aligned}$$

O menor elemento i da permutação tal que $desordem(\sigma_k) \neq 0$ é o elemento $i = 2$, na posição $k = 5$. Assim, ao aplicar a transposição $\rho_t(i,k,k+1) = \rho_t(2,5,6)$ temos que $\sigma \cdot \rho_t = (1\ 2\ 4\ 3\ 5\ 6)$ e $desordem(2) = 0$. Além disso, o elemento $(\sigma \cdot \rho_t)_4 = 3$ foi deslocado de sua posição correta, aumentando o valor de sua desordem em uma unidade. Por sua vez, os elementos $(\sigma \cdot \rho_t)_3 = 4$ e $(\sigma \cdot \rho_t)_5 = 5$, que possuíam valor de desordem não-nula em σ, terão seus valores de desordem decrescidos em uma unidade. Assim, temos os seguintes valores de desordem:

$$\begin{aligned} \sigma \cdot \rho_t &= (1\ 2\ 4\ 3\ 5\ 6) \\ desordem(\sigma \cdot \rho_t) &= (0\ 0\ 1\ 1\ 0\ 0) \\ entro(\sigma \cdot \rho_t) &= 1+1=2 \end{aligned}$$

Definição 3.23. Dada uma permutação π qualquer, $score_{ENTRO}(\pi) = entro(\pi) + neg(\pi)$.

Note que, se π é uma permutação sem sinais, $entro(\pi) = 0 \iff \pi = \iota$. Entretanto, quando π é uma permutação com sinais, $entro(\pi) = 0$ implica apenas que $u(\pi) = \iota$, e não necessariamente que $\pi = \iota$, pois podem existir elementos em suas posições corretas porém com sinais negativos. Desta forma, definimos o $score_{ENTRO}(\pi)$ de uma permutação π como a soma do valor da entropia da permutação e do número de elementos negativos desta permutação.

Lema 3.24. Dada uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível diminuir o valor de $score_{ENTRO}(\pi)$ com uma operação.

Demonstração. Se π é uma permutação sem sinais, o valor de $neg(\pi)$ será sempre nulo. Desta forma, $score_{ENTRO}(\pi) = entro(\pi)$, e o Lema 3.22 garante que existe uma operação que dimimui o valor de $score_{ENTRO}$. Se π é uma permutação com sinais, e enquanto $entro(\pi) > 0$, aplique a operação descrita no Lema 3.22. Como a transposição não troca os sinais dos elementos envolvidos, o valor de $neg(\pi)$ não será alterado com esta operação, garantindo assim que $score_{ENTRO}$ vai necessariamente diminuir. Caso $entro(\pi) = 0$, e $score_{ENTRO}(\pi) > 0$, temos que $neg(\pi) > 0$. Aplique uma reversão unitária ρ_r em algum elemento negativo de π. Esta reversão fará com que $neg(\pi \cdot \rho_r) = neg(\pi) - 1$, e o valor de $entro(\pi)$ não será alterado, uma vez que a reversão aplicada não troca elementos de lugar. □

A heurística testa todas as operações aplicáveis em π a cada passo e realiza a operação que possui o menor valor de $score_{ENTRO}$, até que $score_{ENTRO}(\pi) = 0$ e, consequentemente, $\pi = \iota$. A heurística eventualmente ordena a permutação uma vez que o Lema 3.24 garante que sempre é possível diminuir o valor de $score_{ENTRO}(\pi)$ para $\pi \neq \iota$.

Função 9 NumEntropia(π) (Complexidade: $\mathcal{O}(n)$)

```
1: e ← 0
2: para cada i ∈ [1...n] faça
3:     e ← e + |π_i − i|
4: fim para
5: return e
```

Heurística 10 Entropia(π) (Complexidade: $\mathcal{O}(n^5)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     best ← ∞
4:     α ← π
5:     para cada operação ρ válida faça
6:         σ ← π · ρ
7:         score_ENTRO ← NumEntropia(σ) + neg(σ)
8:         se best > score_ENTRO então
9:             best ← score_ENTRO
10:            α ← σ
11:        fim se
12:    fim para
13:    π ← α
14:    dist ← dist + 1
15: fim enquanto
16: return dist
```

A complexidade de tempo desta heurística é $\mathcal{O}(n^5)$: existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, aplicar cada operação custa $\mathcal{O}(n)$, e calcular o valor de $score_{ENTRO}$ para uma permutação custa $\mathcal{O}(n)$. Como são necessários no máximo $\mathcal{O}(n)$ passos, a complexidade total da heurística é $\mathcal{O}(n^5)$.

3.7 Inversões

Esta heurística utiliza o conceito de inversões para ordenar a permutação. Existe uma *inversão* entre dois elementos π_i e π_j da permutação π se $|\pi_i| > |\pi_j|$ e $j > i$, ou seja, existe um elemento π_j à direita de π_i cujo valor em módulo é menor que ele. O número de inversões em uma permutação π, denotado por $inv(\pi)$, é dado pela soma do número de inversões para cada um de seus elementos. Por exemplo, os elementos da permutação $\sigma = (2\ 5\ 3\ 4\ 1\ 6)$ possuem os seguintes números de inversões: $inv(2) = 1$, pois o elemento 1 está à direita de 2, $inv(5) = 3$, pois os elementos 3, 4 e 1 estão à direita de 5, $inv(3) = 1$, pois o elemento 1 está à direita de 3, $inv(4) = 1$, pois o elemento 1 está à direita de 4, e $inv(1) = 0$, pois não existem elementos à direita de 1 menores que ele. Desta forma, $inv(\sigma) = (1\ 3\ 1\ 1\ 0\ 0) = 6$.

É possível calcular o número de inversões de uma permutação em $\mathcal{O}(n^2)$, testando todos os pares de elementos de π. Entretanto, existe uma forma mais eficiente de calcular o número de inversões utilizando o método de divisão e conquista do algoritmo MergeSort.

O número de inversões quando temos apenas um elemento é sempre igual a zero. Suponha agora que sabemos o número de inversões em duas metades de permutações (agora ordenadas), de tamanhos $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$ respectivamente, durante a execução do MergeSort. O que precisamos saber agora é o número de inversões durante a execução da função *merge()* do MergeSort, que juntará as duas metades.

Vamos utilizar o índice i para percorrer o vetor a, que contém metade dos elementos da permutação, e j para percorrer o vetor b, que contém a outra metade dos elementos da permutação, durante a execução da função *merge()*. Em qualquer passo, se $b[j]$ é escolhido pela função como próximo elemento da permutação ordenada, temos que $a[i] > b[j]$, então existem $len(a) - i$ inversões, uma vez que o vetor está ordenado e, assim, os elementos $a[i+1], a[i+2], ...a[n/2]$ também são maiores que $b[j]$. Recursivamente, é possível calcular o número de inversões enquanto a permutação é ordenada pelo algoritmo e, desta forma, conseguimos calcular com complexidade de tempo $\mathcal{O}(n \log n)$, complexidade do algoritmo MergeSort.

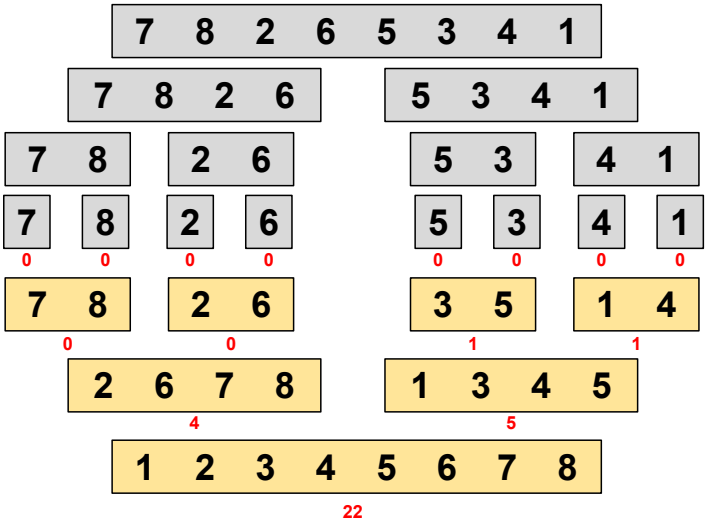


Figura 3.5: Ordenação por MergeSort da permutação $\sigma = (7\ 8\ 2\ 6\ 5\ 3\ 4\ 1)$.

Função 11 MergeAndCount(A, B) (Complexidade: $\mathcal{O}(n)$)

```
1: i ← 0
2: j ← 0
3: count ← 0
4: l ← ∅
5: enquanto a < |A| e b < |B| faça
6:     next = min(A[a], B[b])
7:     l ← l + [next]
8:     se B[b] = next então
9:         b ← b + 1
10:        count ← count + |A| − a
11:    senão
12:        a ← a + 1
13:    fim se
14: fim enquanto
15: se a < |A| então
16:    l ← l + |A[a : |A|]|
17: fim se
18: se b < |B| então
19:    l ← l + |B[b : |B|]|
20: fim se
21: return (count, l)
```

Considere a Figura 3.5, que representa os passos para a ordenação da permutação $\sigma = (7\ 8\ 2\ 6\ 5\ 3\ 4\ 1)$ pelo algoritmo MergeSort. Na fase de divisão, representada pelos quadrados cinzas, não precisamos nos preocupar com o número de inversões. Quando temos apenas conjuntos unitários, atribuímos número de inversões igual a 0, representados em vermelho. A cada merge entre dois conjuntos, atribuímos ao conjunto resultante a soma do número de inversões em cada um mais o número de inversões obtidos durante o *merge*. Por exemplo, quando o algoritmo faz o merge dos conjuntos $(3, 5)$ e $(1, 4)$, temos que número de inversões do novo conjunto será igual a 5, resultante da soma de inversões dos dois conjuntos, acrescidos das 3 inversões identificadas pela função merge: começando com $i = j = 0$, o primeiro elemento escolhido será $b[0] = 1$ e, desta forma, existem

Função 12 MergeSort(L) (Complexidade: $\mathcal{O}(n \log n)$)

```
1: se |L| = 1 então
2:     return (0, L)
3: fim se
4: m = |π|/2
5: A ← L[1 : m]
6: B ← L[m + 1 : |π|]
7: (invA, SortedA) ←MergeSort(A)
8: (invB, SortedB) ←MergeSort(B)
9: (count, SortedL) ←MergeAndCount(SortedA, SortedB)
10: return (count+invA+invB, SortedL)
```

$len(a) - i = 2 - 0 = 2$ inversões. Após a escolha do elemento 3, $i = j = 1$, e o próximo elemento a ser escolhido será novamente $b[1] = 4$ e, agora, existe $len(a) - i = 2 - 1 = 1$ inversões totalizando, desta forma, 5 inversões.

Lema 3.25. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ sem sinais, com $\pi \neq \iota$, sempre é possível diminuir o número de inversões em pelo menos uma unidade com uma transposição.

Demonstração. Seja $\pi_i = a$ o menor elemento fora de sua posição a e, assim, $inv(\pi_i) = 0$, uma vez que não existe $\pi_j < a$ tal que $j > i$. Aplique a transposição $\rho_t(a, i, i + 1)$, colocando o elemento a na posição π_a. Após aplicar esta transposição, o número de inversões dos elementos π_j, com $a < j \leq i$, diminuirá em uma unidade, uma vez que o elemento a é menor que todos os elementos deste intervalo. □

Por exemplo, seja a permutação $\sigma = (1\ 4\ 5\ 2\ 3)$. O menor elemento fora de sua posição é $a = 2$, que está na posição $i = 4$. Desta forma, temos o seguinte valor para o número de inversões de σ:

$$\begin{aligned} \sigma = &\quad (1\ 4\ 5\ 2\ 3) \\ inv(\sigma) = &\quad (0\ 2\ 2\ 0\ 0) \quad = 4 \end{aligned}$$

Note que ao aplicar a transposição $\rho_t(a, i, i + 1) = \rho_t(2, 4, 5)$ os valores de $inv(\pi_j)$, para $2 < j \leq i$ diminuem em uma unidade, destacados em negrito:

$$\begin{aligned} \sigma \cdot \rho_t = &\quad (1\ 2\ 4\ 5\ 3) \\ inv(\sigma \cdot \rho_t) = &\quad (0\ 0\ \mathbf{1}\ \mathbf{1}\ 0) \quad = 2 \end{aligned}$$

Note que, se π é uma permutação sem sinais, $inv(\pi) = 0 \iff \pi = \iota$. Entretando, se π é uma permutação com sinais, $inv(\pi) = 0$ não implica necessariamente em $\pi = \iota$. Por exemplo, a permutação $\pi = (+1\ -2\ +3\ -4\ -5)$ possui $inv(\pi) = 0$, uma vez que em módulo os elementos da permutação formam uma sequência crescente, entretanto, $\pi \neq \iota$.

Definição 3.26. Dada uma permutação π qualquer, $score_{INV}(\pi) = inv(\pi) + neg(\pi)$.

Desta forma, definimos como $score_{INV}$ de uma permutação o número de inversões que ela possui somados ao número de elementos negativos da permutação. Assim, quando $score_{INV} = 0$, temos que $inv(\pi) = 0$ e não existem elementos negativos na permutação e, desta forma, $\pi = \iota$.

Lema 3.27. Dada uma permutação $\pi = (\pi_1\ \pi_2\ ...\ \pi_n)$ tal que $\pi \neq \iota$, sempre é possível diminuir o valor de $score_{INV}(\pi)$ com uma operação.

Demonstração. Se π é uma permutação sem sinais, o valor de $neg(\pi)$ será sempre nulo. Desta forma, $score_{INV}(\pi) = inv(\pi)$, e o Lema 3.25 garante que existe uma operação que dimimui o valor de $score_{INV}$. Se π é uma permutação com sinais, e enquanto $inv(\pi) > 0$, aplique a operação descrita no Lema 3.25. Como a transposição não troca os sinais dos elementos envolvidos, o valor de $neg(\pi)$ não será alterado com esta operação, garantindo assim que $score_{INV}$ vai necessariamente diminuir. Caso $inv(\pi) = 0$, e $score_{INV}(\pi) > 0$, temos que $neg(\pi) > 0$. Aplique uma reversão unitária ρ_r em algum elemento negativo de π. Esta reversão fará com que $neg(\pi \cdot \rho_r) = neg(\pi) - 1$, e o valor de $inv(\pi)$ não será alterado, uma vez que a reversão aplicada não troca elementos de lugar. □

Dada a permutação $\sigma = (+4\ -2\ -1\ -5\ +3)$, temos que $inv(\sigma) = (3\ 1\ 0\ 1\ 0) = 5$. Assim, $score_{INV}(\sigma) = inv(\sigma) + neg(\sigma) = 5 + 3 = 8$.

Assim, a heurística escolhe a cada passo a operação cujo $score_{INV}$ da permutação resultante seja o menor possível. O Lema 3.27 garante que a heurística irá ordenar uma vez que sempre é possível diminuir seu valor com uma operação até que $score_{INV}(\pi) = 0$ e, assim, $\pi = \iota$.

Heurística 13 Inversões(π) (Complexidade: $\mathcal{O}(n^5 \log n)$)

```
1: dist ← 0
2: enquanto (π ≠ ι) faça
3:     best ← ∞
4:     para cada operação ρ válida faça
5:         σ ← π · ρ
6:         (inv, sort) ← MergeSort(u(σ))
7:         score_INV ← inv + neg(σ)
8:         se best > score_INV então
9:             best ← score_INV
10:            α ← σ
11:        fim se
12:    fim para
13:    π ← α
14:    dist ← dist + 1
15: fim enquanto
16: return dist
```

A complexidade de tempo desta heurística é $\mathcal{O}(n^5 \log n)$: existem $\mathcal{O}(n^3)$ reversões e transposições aplicáveis em π, aplicar cada operação custa $\mathcal{O}(n)$, e calcular o valor de $score_{INV}$ com o auxílio da função MergeSort custa $\mathcal{O}(n \log n)$. Como são necessários no máximo $\mathcal{O}(n)$ passos, a complexidade total da heurística é $\mathcal{O}(n^5 \log n)$.

3.8 Resultados

Com o objetivo de comparar a performance das heurísticas apresentadas neste capítulo, criamos dois bancos de permutações de tamanho n, para $10 \leq n \leq 100$, e $n \equiv 0 \pmod 5$, sendo um banco composto apenas de permutações com sinais e outro apenas com permutações sem sinais.

Para cada valor de n, foram feitas 10000 execuções do Algoritmo 14, tal que cada execução gerasse uma permutação distinta das anteriores. Em cada execução, a partir da permutação ι, o Algoritmo 14 aplica 10 reversões e 10 transposições. Assim, qualquer permutação contida nos bancos de permutações pode ser ordenada com no máximo 20 operações cada. Para criar o banco de permutações sem sinais, as reversões aplicadas pelo algoritmo não inverteram a orientação dos elementos envolvidos na operação, bem como não foram aplicadas reversões que afetam apenas um elemento da permutação. Já para criar o banco de permutações com sinais, as reversões aplicadas inverteram os sinais de cada elemento envolvido na operação.

Algoritmo 14 Gerador de Permutações Aleatórias

1: $i \leftarrow 0$
2: $j \leftarrow 0$
3: $k \leftarrow 0$
4: $perm \leftarrow \iota_n$
5: $nrev \leftarrow 10$
6: $ntransp \leftarrow 10$
7: $sort \leftarrow \emptyset$
8: **enquanto** $nrev + ntrans > 0$ **faça**
9: $op \leftarrow rand(r, t)$
10: **se** $op = r$ **e** $nrev > 0$ **então**
11: $i \leftarrow rand(1, n)$
12: $j \leftarrow rand(1, n)$
13: $perm \leftarrow rev(perm, i, j)$
14: $sort \leftarrow sort + [[i, j]]$
15: $nrev \leftarrow nrev - 1$
16: **senão se** $op = t$ **e** $ntransp > 0$ **então**
17: $i \leftarrow rand(1, n)$
18: $j \leftarrow rand(1, n)$
19: $k \leftarrow rand(1, n + 1)$
20: $perm \leftarrow transp(perm, i, j, k)$
21: $sort \leftarrow sort + [[i, j, k]]$
22: $ntransp \leftarrow ntransp - 1$
23: **fim se**
24: **fim enquanto**
25: **return** *(perm, sort)*

Todas as heurísticas foram executadas com cada uma das permutações, e a distância média retornada para cada conjunto de permutações de tamanho n, para permutações sem sinais e permutações com sinais estão nas tabelas 3.1 e 3.2, respectivamente. As heurísticas Entropia, Inversões e Breakpoints foram denotadas por **ENTRO**, **INV** e **BP**, respectivamente. Note que quanto melhor a heurística, menor o número de operações que ela utiliza. Assim, uma heurística A produz resultados melhores que uma heurística B se, dado uma permutação ou conjunto de permutações, a distância média obtida por A é menor que a distância média obtida por B.

Note que, com exceção das permutações sem sinais de tamanho 10, cujas heurísticas LC+RC e LIS+BP apresentaram o melhor e o segundo melhor resultado, respectivamente, a heurística Breakpoints apresenta sempre os melhores resultados tanto para permutações sem sinais com tamanho maior que 10 quanto para qualquer tamanho de permutações com sinais.

Comparando as heurísticas LIS+BP e Breakpoints, podemos notar que os resultados da primeira apenas foram melhores com permutações sem sinais de tamanho 10, ou seja, para qualquer permutação utilizada nos testes sem sinais com tamanho maior ou igual a 15, ou para qualquer permutação utilizada nos testes com sinais, a métrica LIS na média acaba atrapalhando a performance da métrica Breakpoints.

Os resultados das heurísticas LIS+BP e LC+RC mostram que tanto para permutações com sinais, quanto para permutações sem sinais ambas apresentam resultados medianos, sendo que a heurística LIS+BP obteve resultados levemente melhores que a heurística LC+RC. Isso mostra que a métrica Breakpoints ajuda nos resultados da métrica LIS quando comparado com a utilização apenas da métrica LIS, uma vez que os resultados da heurística LIS+BP são sempre melhores na média que os resultados da heurística LIS.

Já as demais heurísticas, LIS, LIS+LDS e Entropia, apresentaram os piores conjuntos de resultados na média, tanto para permutações com sinais, quanto para permutações sem sinais, chegando a apresentar distância média acima de 40 para determinados conjuntos, o que representa o dobro da distância máxima das permutações do banco de permutações. Os resultados também mostram que a métrica LDS melhorou os resultados da utilização apenas da métrica LIS com permutações sem sinais, e piorou os resultados da utilização apenas da métrica LIS com permutações com sinais.

n	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
LC+RC	4.41	7.18	9.84	12.29	14.46	16.35	18.02	19.53	20.86	22.10	23.14	24.13	25.01	25.79	26.52	27.17	27.75	28.35	28.87
LIS	4.66	7.74	10.77	13.66	16.29	18.78	21.09	23.29	25.35	27.37	29.19	31.07	32.77	34.40	36.18	37.74	39.26	40.86	42.30
LIS+LDS	4.60	7.67	10.68	13.53	16.17	18.63	20.94	23.10	25.13	27.16	28.92	30.80	32.47	34.16	35.84	37.38	38.86	40.46	41.92
ENTRO	5.01	8.54	12.09	15.42	18.41	21.16	23.66	25.90	27.91	29.84	31.41	33.03	34.39	35.76	36.97	38.02	39.02	40.02	40.83
LIS+BP	4.44	7.16	9.75	12.14	14.29	16.21	17.95	19.46	20.81	22.08	23.12	24.17	25.07	25.89	26.62	27.29	27.91	28.48	29.03
INV	4.88	8.24	11.56	14.70	17.49	20.03	22.26	24.28	26.13	27.84	29.22	30.58	31.76	32.90	33.95	34.86	35.66	36.45	37.15
BP	4.48	7.07	9.45	11.53	13.26	14.73	16.00	17.03	17.93	18.73	19.31	19.86	20.33	20.74	21.06	21.34	21.55	21.75	21.95

Tabela 3.1: Distâncias médias para permutações sem sinais.

n	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
LC+RC	7.33	11.05	14.37	17.20	19.60	21.55	23.22	24.65	25.86	26.88	27.87	28.58	29.34	29.97	30.46	31.08	31.48	31.86	32.29
LIS	7.59	11.63	15.24	18.36	21.02	23.21	25.14	26.78	28.22	29.39	30.53	31.44	32.35	33.15	33.78	34.53	35.05	35.58	36.05
LIS+LDS	7.56	11.60	15.20	18.32	20.99	23.18	25.11	26.74	28.19	29.36	30.51	31.41	32.33	33.13	33.77	34.53	35.05	35.57	36.04
ENTRO	8.30	13.12	17.55	21.50	24.94	27.90	30.45	32.74	34.78	36.57	38.22	39.57	40.87	42.15	43.23	44.31	45.16	46.04	46.91
LIS+BP	7.13	10.67	13.80	16.53	18.86	20.79	22.53	24.00	25.26	26.37	27.44	28.30	29.13	29.87	30.47	31.17	31.68	32.19	32.69
INV	7.93	12.44	16.62	20.28	23.43	26.11	28.46	30.46	32.19	33.75	35.11	36.23	37.34	38.36	39.19	40.09	40.77	41.43	42.00
BP	6.90	10.03	12.59	14.59	16.17	17.38	18.34	19.07	19.66	20.08	20.50	20.74	20.94	21.18	21.26	21.42	21.52	21.60	21.68

Tabela 3.2: Distâncias médias para permutações com sinais.

Capítulo 4

Heurística de Grafo de Ciclos

Este capítulo apresenta uma heurística desenvolvida em nosso trabalho que ordena permutações com sinais e permutações sem sinais por reversões e transposições que decide quais operações serão aplicadas principalmente com base no grafo de ciclos da permutação. Dessa forma, é necessário primeiramente encontrar uma decomposição em ciclos da permutação a ser ordenada.

No caso de permutações com sinais, esta decomposição é única, como explicado na Seção 2.3. Já no caso de permutações sem sinais, é preciso encontrar uma decomposição de ciclos, de preferência máxima, mas este é um problema NP-Difícil [10]. Assim, optamos por utilizar uma decomposição de ciclos aproximada apresentada por Lin e Jiang [31], cujo fator de aproximação é de 1.4193 que, apesar de ser inferior ao melhor fator de aproximação de 1.4167 apresentado por Chen [12], possui um algoritmo mais simples.

Dada a decomposição de ciclos $G(\sigma)$ de uma permutação σ sem sinais, podemos rotular os vértices da mesma forma que é realizada a rotulação do grafo de ciclos de permutações com sinais. O primeiro vértice recebe a rotulação $+0$ e o último vértice recebe o rótulo $-(n+1)$. Os demais vértices podem ser rotulados seguindo a regra de rotulação de grafo de ciclos a partir do vértice $+0$. Desta forma, é possível obter uma permutação com sinais π a partir do grafo de ciclos da permutação σ da seguinte forma: se dois vértices consecutivos são da forma $(\sigma_i, -\sigma_i)$, então o sinal deste elemento será negativo em π. Caso contrário, o sinal deste elemento será positivo.

Por exemplo, dada a permutação sem sinais $\sigma = (1\ 3\ 2\ 5\ 4)$, o algoritmo de Lin e Jiang [31] retorna a decomposição de ciclos apresentada na Figura 4.1. A partir desta decomposição, podemos rotular os vértices começando pelo vértice mais a esquerda, cujo valor é sempre 0, e obter a permutação com sinais $\pi = (+1\ -3\ -2\ -5\ -4)$.

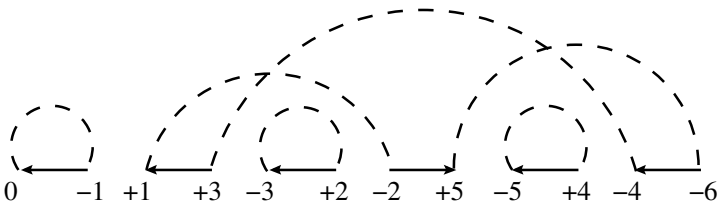


Figura 4.1: Decomposição de ciclos da permutação sem sinais $\sigma = (1\ 3\ 2\ 5\ 4)$.

Dada a decomposição única de ciclos $G(\alpha)$ de uma permutação α com sinais (como visto na Seção 2.3), temos que os vértices de $G(\alpha)$ já foram rotulados pela regra mencionada acima. Por exemplo, dada a permutação com sinais $\alpha = (+1\ -3\ -2\ +5\ +4)$, obtemos a decomposição única de ciclos apresentada na Figura 4.2. Como os vértices de α já foram rotulados seguindo a mesma regra, temos que obter uma permutação com sinais π a partir da rotulação dos vértices de $G(\alpha)$ implica em $\pi = \alpha$.

A partir deste momento, trabalharemos sempre com a permutação com sinais π e seu grafo de ciclos, obtidos a partir da decomposição de ciclos.

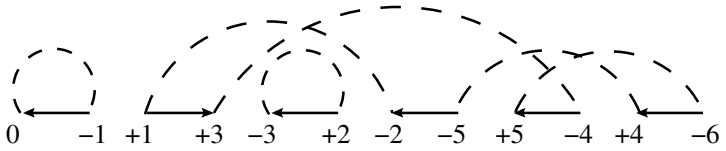


Figura 4.2: Decomposição de ciclos da permutação com sinais $\alpha = (+1\ -3\ -2\ +5\ +4)$.

Os resultados apresentados na Seção 3.8 mostram que a heurística Breakpoints, definida na Seção 3.2, obteve os melhores resultados em todos os conjuntos de permutações com sinais e em 90% dos conjuntos de permutações sem sinais. Assim, a heurística que apresentamos nesta seção primeiramente procura por operações que removam pelo menos dois breakpoints de π. Caso não encontre uma operação que remova pelo menos dois breakpoints, a heurística procura aumentar o número de ciclos ímpares do grafo, analisando as características dos ciclos existentes. Por último, caso não seja possível aumentar o número de ciclos ímpares do grafo com uma operação ou aumentar o número de ciclos do grafo com garantia de aproximação 1.5, a heurística irá procurar uma sequência de operações em um banco de configurações criado. As próximas três seções detalham as etapas utilizadas pela heurística.

4.1 Remoção de breakpoints

A verificação de uma operação que remova breakpoints de π está dividida em 3 partes: remoção de três breakpoints com uma transposição, remoção de dois breakpoints com uma reversão e remoção de dois breakpoints com uma transposição.

Remoção de 3 breakpoints com uma transposição

O primeiro passo realizado pela heurística consiste em procurar uma transposição que remova três breakpoints e, se tal operação existir, ela será aplicada à π. O Lema 4.1 mostra a configuração do ciclo que permite a remoção de três breakpoints e a operação necessária.

Lema 4.1. Dados o grafo de ciclos $G(\pi)$ de uma permutação π e uma transposição ρ_t que remove três breakpoints, ρ_t é aplicada nas três arestas pretas de um 3-ciclo orientado convergente de $G(\pi)$.

Demonstração. Para remover 3 breakpoints com uma transposição, a permutação deve possuir três elementos i, j e k, com $p(i) < p(k) < p(j)$, tais que $p(j+1) = p(i)+1$, $p(i+1) = p(k)+1$ e $p(k+1) = p(j)+1$. Além disso, os sinais dos pares $(i,(i+1))$, $(j,(j+1))$ e $(k,(k+1)$ devem ser iguais. A Figura 4.3 mostra as arestas cinzas que necessariamente devem existir em $G(\pi)$ para que esta operação exista, implicando em um 3-ciclo orientado convergente na forma $C = (p(k+1), p(j+1), p(i+1))$. Seja a a aresta preta que liga os vértices i e $(j{+}1)$, b a aresta preta que liga os vértices k e $(i{+}1)$, e c a aresta preta que liga os vértices j e $(k{+}1)$. Assim, a transposição $\rho_t(a,b,c)$ transforma este 3-ciclo em 3 ciclos unitários, resultando em $\Delta c(\pi,\rho_t) = 2$ e $\Delta c_{impar}(\pi,\rho_t) = 2$. □

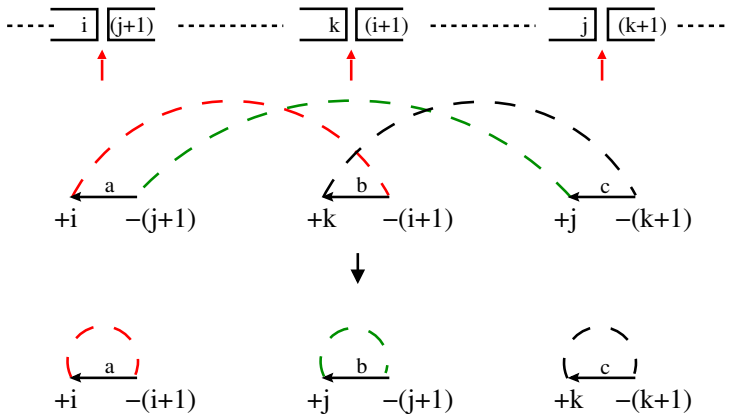


Figura 4.3: Configuração necessária em $G(\pi)$ para existir uma transposição que remova 3 breakpoints.

Remoção de 2 breakpoints com uma reversão

O segundo passo da remoção de breakpoints, caso não encontre uma transposição que remova três breakpoints, será procurar uma reversão que remova dois breakpoints e, caso encontre, esta reversão será aplicada em π. O Lema 4.2 mostra a configuração do ciclo que permite a remoção de dois breakpoints com uma reversão e a operação necessária para tal.

Lema 4.2. Dados o grafo de ciclos $G(\pi)$ de uma permutação π, e uma reversão ρ_r que remove dois breakpoints, ρ_r é aplicada nas duas arestas pretas de um único 2-ciclo divergente de $G(\pi)$.

Demonstração. Para remover 2 breakpoints com uma reversão, a permutação deve possuir dois elementos i e j, com $p(i) < p(j)$, tais que $p(j) = p(i)+1$ e $p(j+1) = p(i+1)+1$. Além disso, temos obrigatoriamente $i \times (i+1) < 0$ e $j \times (j+1) < 0$. A Figura 4.4 mostra as arestas que necessariamente devem existir em $G(\pi)$ para que esta reversão exista, implicando em um 2-ciclo divergente na forma $C = (p(i+1), -p(j))$. Seja a a aresta preta que liga os vértices i e j, b a aresta preta que liga os vértices $-(i{+}1)$ e $(j{+}1)$ e $(b{-}1)$ a aresta preta à esquerda de b. Assim, a reversão $\rho_r(a, b-1)$ transforma este 2-ciclo em 2 ciclos unitários, resultando em $\Delta c(\pi,\rho_r) = 1$ e $\Delta c_{impar}(\pi,\rho_r) = 2$. □

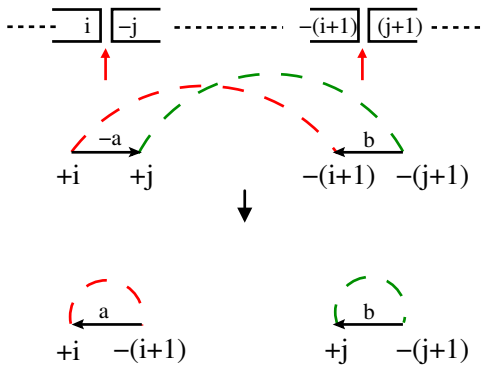


Figura 4.4: Configuração necessária em $G(\pi)$ para existir uma reversão que remova 2 breakpoints.

Remoção de 2 breakpoints com uma transposição

O terceiro e último passo da remoção de breakpoints, caso não encontre uma reversão que remova dois breakpoints, será procurar uma transposição que remova dois breakpoints e, se encontrar, aplicá-la em π. O Lema 4.3 mostra a configuração do ciclo que permite a remoção de dois breakpoints com uma transposição e a operação necessária para tal.

Lema 4.3. Dados o grafo de ciclos $G(\pi)$ de uma permutação π, e uma transposição ρ_t que remove dois breakpoints, ρ_t é aplicada em três arestas pretas de um mesmo k-ciclo de $G(\pi)$, com $k > 3$.

Demonstração. Para remover 2 breakpoints com uma transposição, a permutação deve possuir dois elementos i e j, com $p(i) < p(j)$, tais que

- $p(j+1) = p(i) + 1$ e $p(j) > p(i+1) > p(j+1)$, removendo os dois primeiros breakpoints, ou
- $p(i) < p(j)$, $p(i+1) = p(j) + 1$ e $p(j+1) > p(i+1)$, removendo o primeiro e o último breakpoint, ou
- $p(i) > p(j) > p(i+1)$ e $p(j+1) = p(i) + 1$, removendo os dois últimos breakpoints.

Além disso, os sinais dos elementos que formam os pares $(i, (i+1))$ e $(j, (j+1))$ devem ser iguais. A Figura 4.5 mostra as arestas que necessariamente devem existir em $G(\pi)$ para que cada uma destas transposições exista, implicando em um k-ciclo com $k > 3$. Assim, a transposição $\rho_t(a, b, c)$ transforma este k-ciclo em 2 ciclos unitários e um $(k-2)$-ciclo. Desta forma, $\Delta c(\pi, \rho_t) = 2$ e $\Delta c_{impar}(\pi, \rho_t) = 2$. □

Considerações gerais

Note que qualquer operação encontrada nesta etapa aumenta $c_{impar}(\pi)$. Note também que eventualmente não será possível encontrar operações que removam dois ou três breakpoints. A heurística não busca uma operação que remova um breakpoint por dois

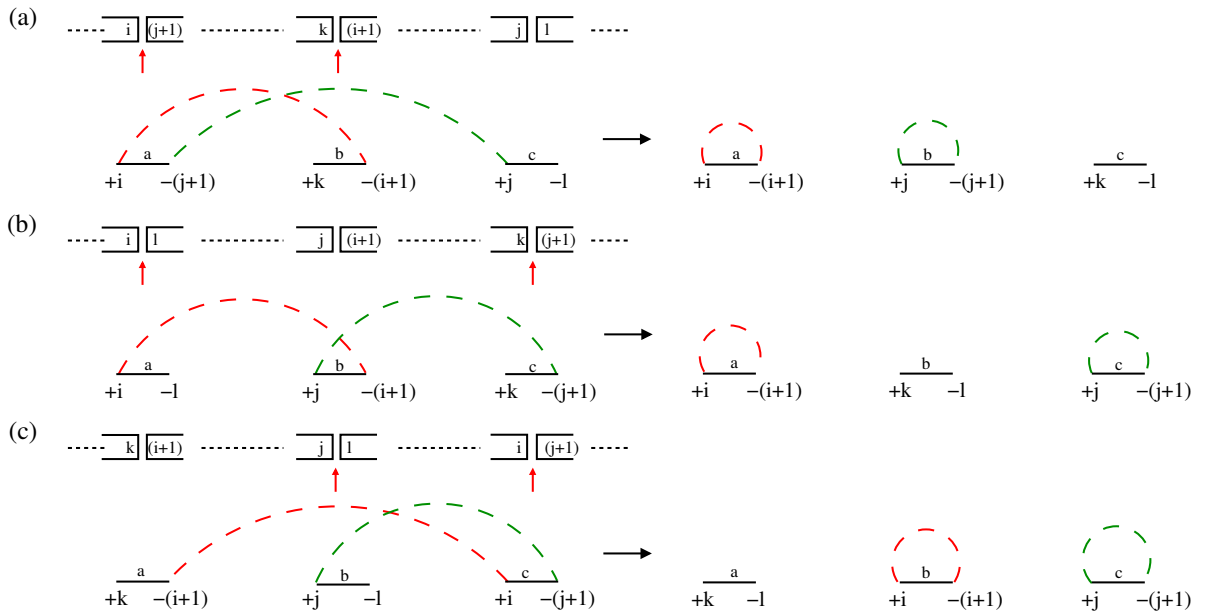


Figura 4.5: Configurações necessárias em $G(\pi)$ para existir uma transposição que remova 2 breakpoints.

motivos: primeiramente, o Lema 3.11 garante que sempre existirá tal operação, e desta forma, a heurística acabaria realizando sempre as mesmas operações e retornando os mesmos resultados da heurística Breakpoints. Além disso, ao remover um breakpoint não é garantido que $c_{impar}(\pi)$ irá aumentar ao aplicar a operação encontrada. A Função 15 resume a ordem dos passos seguidos por esta etapa.

Função 15 RemoveBreakpoints($G(\pi)$)

```
1: se existe ρ_t segundo o Lema 4.1 então
2:     return ρ_t
3: fim se
4: se existe ρ_r segundo o Lema 4.2 então
5:     return ρ_r
6: fim se
7: se existe ρ_t segundo o Lema 4.3 então
8:     return ρ_t
9: fim se
10: return ∅
```

4.2 Análise dos ciclos existentes

Nesta parte, a heurística irá classificar os ciclos de $G(\pi)$ de acordo com suas características e verificar a existência de uma operação que aumente primeiramente o número de ciclos ímpares. Em último caso, a heurística irá verificar uma operação que aumente o número de ciclos pares sem alterar o número de ciclos ímpares do grafo de ciclos $G(\pi)$, dado que esta operação garante o fator de aproximação de 1.5.

Ciclos orientados convergentes com duas distâncias ímpares

No primeiro passo da análise, a heurística procura por triplas orientadas em ciclos orientados que aumentem o número de ciclos ímpares do grafo com uma operação. Seja a distância $dist(r,t)$, entre duas arestas pretas r e t no mesmo ciclo C, o número de arestas cinzas em C que devem ser percorridas para ir de r para t, considerando a orientação do ciclo. O Lema 4.4 mostra a configuração necessária para existir um ciclo orientado que permita uma transposição que aumente o número de ciclos ímpares, bem como a transposição que deve ser aplicada para que $c_{impar}(\pi)$ aumente.

Lema 4.4. Dados o grafo de ciclos $G(\pi)$ de uma permutação π e uma tripla de arestas pretas orientadas de um ciclo C orientado, se existem pelo menos duas distâncias ímpares entre as arestas de C, então existe uma transposição que aumenta o número de ciclos ímpares de $G(\pi)$.

Demonstração. Uma transposição em uma tripla orientada (a,b,c) de um ciclo orientado irá gerar 3 ciclos c_1, c_2 e c_3, com tamanhos $|c_1| = dist(a,b)$, $|c_2| = dist(b,c)$, $|c_3| = dist(c,a)$. Se exatamente duas distâncias entre as arestas que formam esta tripla orientada forem ímpares, a soma das três distâncias implica que C é ciclo par e, ao aplicar a transposição, temos que dois novos ciclos ímpares serão criados (os dois que possuem tamanho igual as duas distâncias ímpares acima). Se as três distâncias são ímpares, então a soma das três distâncias implica que C também é um ciclo ímpar. Como este ciclo é removido e são adicionados três ciclos ímpares, temos que $\Delta c_{impar}(\pi, \rho_t) = 2$. A Figura 4.6 exemplifica um ciclo orientado onde $dist(a,b) = x$, $dist(b,c) = z$ e $dist(c,a) = y$. Note que a transposição $\rho_t(a,b,c)$ cria 3 novos ciclos, sendo o ciclo a com x arestas cinzas, o ciclo b com y arestas cinzas e o ciclo c com z arestas cinzas. □

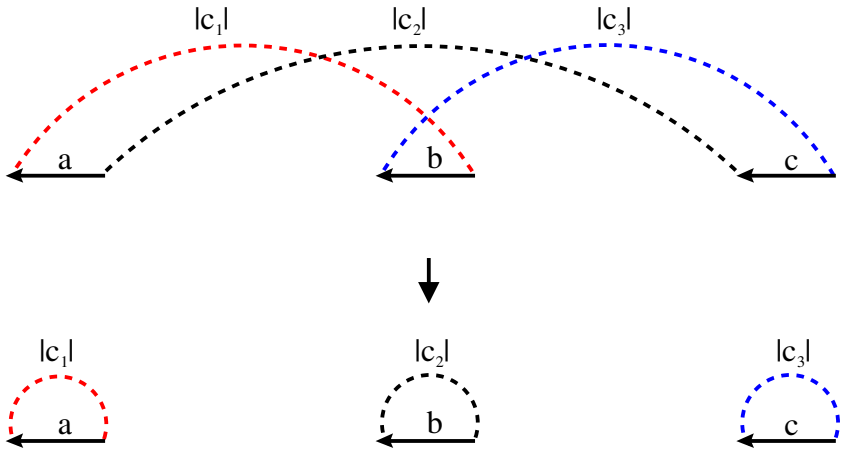


Figura 4.6: Aplicação de uma transposição $\rho_t(a,b,c)$ em $G(\pi)$, transformando o ciclo orientado (c,...,a,...,b,...) em três novos ciclos com tamanhos i, j e k. Neste caso os arcos pontilhados não representam arestas cinzas, mas sim a existência de caminhos com $|c_1|$, $|c_2|$ e $|c_3|$ arestas cinzas, respectivamente.

Assim, o que a heurística busca neste passo são ciclos orientados tais que existam pelo menos duas distâncias ímpares em uma tripla orientada deste ciclo, garantindo assim a

criação de dois novos ciclos ímpares com uma transposição. Note que não podem existir triplas orientadas de ciclos orientados com mais de uma distância unitária entre as arestas pretas, uma vez que são ciclos tais que uma transposição remove dois ou três breakpoints da permutação, e os passos da etapa anterior de remoção de breakpoints já eliminaram tais ciclos. Após a listagem de todos os ciclos com a restrição acima, nossa heurística dará preferência para o ciclo cuja tripla orientada esteja entrelaçada com uma tripla de um ciclo não-orientado com duas distâncias ímpares, pois, como mostra a Figura 4.7, a transposição transforma este ciclo em um ciclo orientado, garantindo assim que na próxima iteração exista uma transposição que aumente o número de ciclos ímpares. Caso não existam tais ciclos, a preferência será de um ciclo cuja tripla orientada esteja entrelaçada com uma tripla de um ciclo não-orientado apenas. Caso tais ciclos também não existam, a heurística aplicará a transposição em qualquer tripla orientada de um ciclo orientado com duas distâncias ímpares.

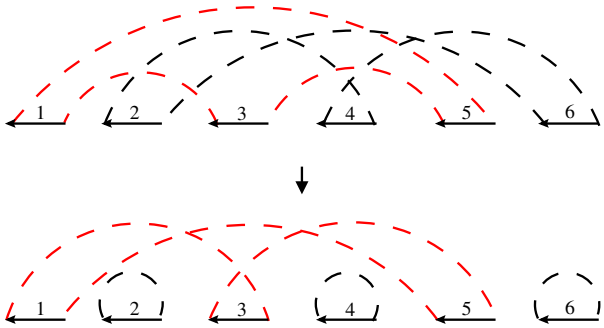


Figura 4.7: Aplicação de uma transposição $\rho_t(2, 4, 6)$ em $G(\pi)$, transformando o ciclo não-orientado $(5, 3, 1)$ em um ciclo orientado $(5, 1, 3)$.

Ciclos pares divergentes

No segundo passo da análise, a heurística procura por duas arestas pretas divergentes em um ciclo par divergente que, ao aplicar uma reversão, aumente o número de ciclos ímpares do grafo. O Lema 4.5 mostra como esta reversão afeta o ciclo e em quais arestas ela deve ser aplicada.

Lema 4.5. Seja $G(\pi)$ o grafo de ciclos de uma permutação π. Se existe um ciclo C par divergente em $G(\pi)$, então existe uma reversão que aumenta o número de ciclos ímpares de $G(\pi)$.

Demonstração. Seja $C = (..., -i, j, ...)$ o ciclo divergente de tamanho n par. Ao aplicar a reversão $\rho_r(i, j-1)$, será criado um ciclo C_1 com uma aresta preta e um ciclo C_2 com $(n-1)$ arestas. Como C é par, C_2 será necessariamente impar, garantindo, assim, que $\Delta c_{impar}(\pi, \rho_r) = 2$. A Figura 4.8 mostra um ciclo divergente e os dois ciclos resultantes após a aplicação da reversão. □

Note que não podem existir ciclos pares divergentes com apenas duas arestas pretas, uma vez que são ciclos tais que uma reversão removeria dois breakpoints da permutação,

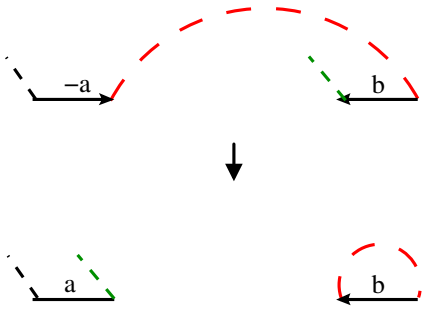


Figura 4.8: Aplicação de uma reversão $\rho_r(a, b-1)$ em $G(\pi)$, transformando o ciclo par divergente em dois ciclos ímpares.

e o segundo passo da etapa anterior de remoção de breakpoints já eliminou tais ciclos. Após a listagem de todos os ciclos C com a restrição acima, nossa heurística aplicará a reversão no ciclo par C tal que exista um ciclo D orientado com duas distâncias ímpares que contenha uma aresta preta divergente entre as arestas i e j pois, como mostra a Figura 4.9. Esta reversão transformará este ciclo orientado divergente D em um ciclo orientado convergente, garantindo assim que na próxima iteração exista uma transposição que aumente o número de ciclos ímpares. Caso não exista tal ciclo, a heurística aplicará a reversão em qualquer ciclo par divergente C.

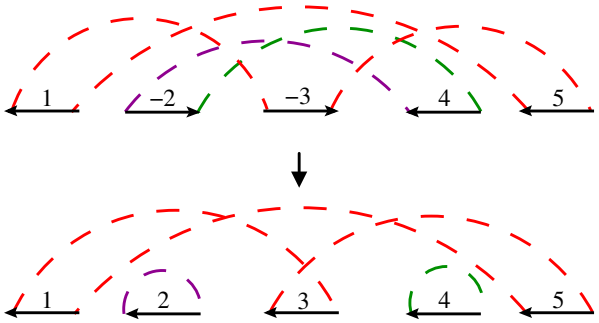


Figura 4.9: Aplicação da reversão $\rho_r(2, 3)$ em $G(\pi)$, transformando o ciclo $(4, -2)$ em dois ciclos ímpares unitários, e transformando o ciclo orientado divergente $(5, 1, -3)$ em um ciclo orientado convergente $(5, 1, 3)$.

Ciclos pares convergentes que se cruzam

No terceiro passo, a heurística procura por uma tripla de arestas pretas em dois ciclos pares C_1 e C_2 tal que exista pelo menos um aresta cinza de C_1 adjancente a uma aresta preta da tripla cruzando com pelo menos uma aresta cinza de C_2 adjacente a uma aresta preta da tripla, e aplica uma transposição nesta tripla. O Lema 4.6 mostra como esta transposição afeta os ciclos e em quais arestas ela deve ser aplicada.

Lema 4.6. Dados o grafo de ciclos $G(\pi)$ de uma permutação π e dois ciclos pares C_1 e C_2 tais que exista pelo menos um aresta cinza de C_1 adjacente a uma aresta preta da tripla cruzando com pelo menos uma aresta cinza de C_2 adjacente a uma aresta preta da

tripla, então existe uma transposição que aumenta o número de ciclos ímpares de $G(\pi)$ e cria um ciclo orientado.

Demonstração. Seja $C_1 = (..., c, a, ...)$ e $C_2 = (..., b, ...)$ dois ciclos pares tais que $c > b > a$. Ao aplicar a transposição $\rho_t(a, b, c)$, os ciclos C_1 e C_2 serão removidos de π e dois novos ciclos C_a e C_b serão criados tais que C_a possui tamanho $|C_1| - 1$ e C_b possui tamanho $|C_2| + 1$. Como C_1 e C_2 são pares, então C_a e C_b são ímpares e, desta forma, $\Delta c_{impar}(\pi, \rho_t) = 2$. A Figura 4.10 mostra dois ciclos pares gerando um ciclo unitário e um ciclo ímpar com uma transposição.

A aresta cinza representada na Figura 4.10 pela cor azul é removida de C_1 e inserida no ciclo unitário C_b, e as demais arestas cinzas de C_1 e C_2 formam o ciclo C_a. Note que a aresta cinza representada pela cor vermelha é uma aresta *direita* tal que apenas seu vértice à direita é movido pela transposição também à direita, logo é uma aresta *direita* em C_a. Da mesma forma, a aresta cinza representada pela cor verde é uma aresta *direita* tal que apenas seu vértice esquerdo é movido pela transposição também à esquerda, logo é uma aresta *direita* em C_a. Sendo assim, o ciclo C_a possui pelo menos duas arestas cinzas direitas e, desta forma, é necessariamente orientado. □

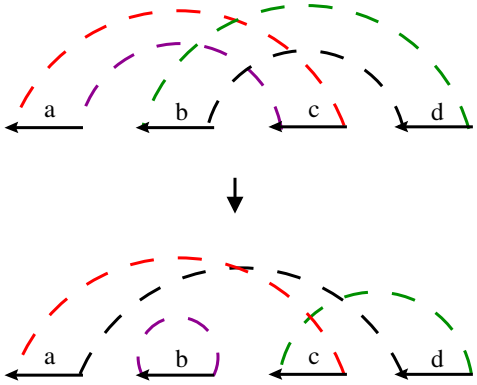


Figura 4.10: Aplicação de uma transposição $\rho_t(a, b, c)$ em dois ciclos pares que se cruzam em $G(\pi)$, gerando dois ciclos ímpares.

Da mesma forma que no primeiro passo da análise, nossa heurística dará preferência para o par de ciclos cuja tripla esteja entrelaçada com uma tripla de um ciclo não-orientado com duas distâncias ímpares. Caso contrário, a preferência será para o o par de ciclos cuja tripla esteja entrelaçada com uma tripla de um ciclo não-orientado apenas. Caso tal ciclo também não exista, a heurística aplicará a transposição em qualquer par de ciclos pares convergentes que se cruzam.

Ciclos pares convergentes

No quarto passo, a heurística procura por uma tripla de arestas pretas em dois ciclos pares C_1 e C_2, e aplica uma transposição nesta tripla, gerando dois novos ciclos ímpares. O Lema 4.7 mostra como esta transposição afeta os ciclos e em quais arestas ela deve ser aplicada.

Lema 4.7. Dados o grafo de ciclos $G(\pi)$ de uma permutação π e dois ciclos pares C_1 e C_2 que não se cruzam, então existe uma transposição que aumenta o número de ciclos ímpares de $G(\pi)$.

Demonstração. Seja $C_1 = (..., b, a, ...)$ e $C_2 = (..., c, ...)$ dois ciclos pares tais que não exista nenhuma aresta cinza de C_1 que cruza com uma aresta cinza de C_2. Ao aplicar a transposição $\rho_t(a, b, c)$, os ciclos C_1 e C_2 serão removidos de π e dois novos ciclos C_a e C_b serão criados tais que C_a possui tamanho $|C_1| - 1$ e C_b possui tamanho $|C_2| + 1$. Como C_1 e C_2 são pares, então C_a e C_b são ímpares e, desta forma, $\Delta c_{impar}(\pi, \rho_t) = 2$. A Figura 4.11 mostra dois ciclos pares gerando um ciclo unitário e um ciclo ímpar com a transposição $\rho_t(a, b, c)$. □

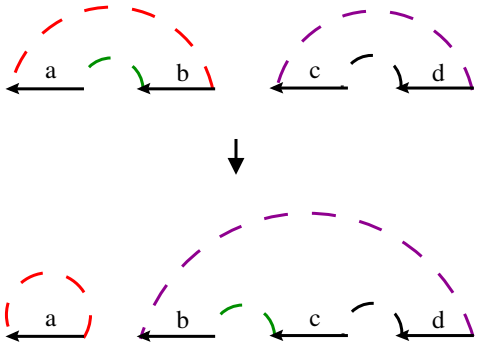


Figura 4.11: Aplicação de uma transposição $\rho_t(a, b, c)$ em dois ciclos pares $C_1 = (b, a)$ e $C_2 = (d, c)$ que não se cruzam $G(\pi)$, gerando dois ciclos ímpares $C_a = (a)$ e $C_b = (d, c, b)$.

Diferentemente de quando os dois ciclos se cruzam, o ciclo gerado aqui será não-orientado. Da mesma forma que no primeiro passo da análise, nossa heurística dará preferência para o par de ciclos cuja tripla esteja entrelaçada com uma tripla de ciclo não-orientado com duas distâncias ímpares. Caso contrário, a heurística dará preferência para o par de ciclos cuja tripla esteja entrelaçada com uma tripla de um ciclo não-orientado apenas. Caso tal ciclo também não exista, a heurística aplicará a transposição em qualquer par de ciclos pares convergentes que não se cruzam.

Ciclo orientado convergente sem duas distâncias ímpares

Se a heurística não encontrou nenhuma operação com os passos anteriores da análise, então não é possível aumentar o número de ciclos ímpares de $G(\pi)$. Como quinto passo, antes de utilizarmos o banco de ordenações de configurações, a heurística procura uma tripla orientada em um ciclo orientado que não se encaixa nos passos anteriores. Bafna e Pevzner [3] provaram que ciclos orientados convergentes que não possuem uma tripla com duas distâncias ímpares geram uma sequência de operações com fator de aproximação 1.5.

Da mesma forma que nos passos anteriores, nossa heurística dará preferência para o ciclo cuja tripla orientada esteja entrelaçada com uma tripla de um ciclo não-orientado com duas distâncias ímpares, e, caso não exista, para o ciclo cuja tripla orientada esteja entrelaçada com uma tripla de um ciclo não-orientado apenas. Caso tal ciclo também

não exista, a heurística aplicará a transposição em qualquer tripla orientada de um ciclo orientado.

A Figura 4.12 mostra a aplicação de uma sequência de transposições em um ciclo orientado convergente. A primeira transposição aplicada não gera novos ciclos ímpares, mas garante que as próximas duas transposições aumentem em 2 unidades cada o número de ciclos ímpares do grafo de ciclos. Desta forma, temos que o fator de aproximação é de 1.5.

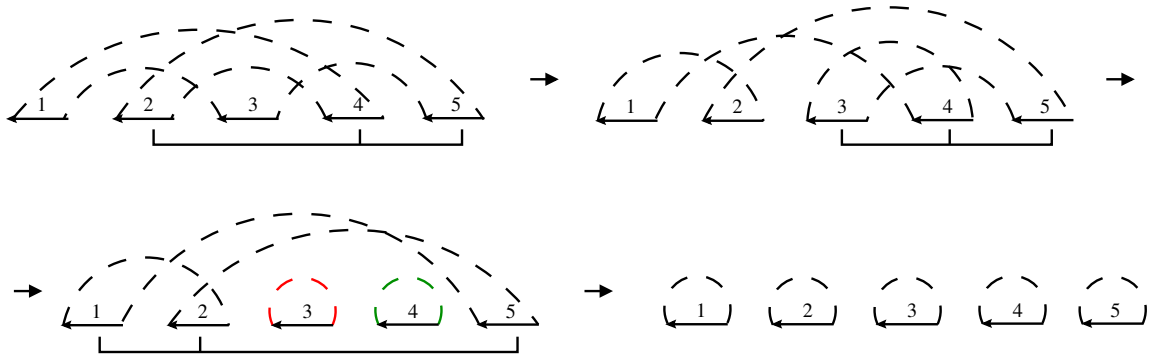


Figura 4.12: Aplicação da sequência de transposições $\rho_t(2,4,5)$, $\rho_t(3,4,5)$ e $\rho_t(1,2,5)$ no ciclo orientado convergente $C = (5,3,1,4,2)$.

Considerações gerais

Note que qualquer operação encontrada nesta etapa garante a aproximação de 1.5. Note também que eventualmente não será possível encontrar operações que aumentem o número de ciclos ímpares ou o número de ciclos da permutação. A Função 16 resume a ordem dos passos seguidos por esta etapa.

Função 16 AumentaCiclos($G(\pi)$)

1: **se** existe ρ_t segundo o Lema 4.4 **então**
2: **return** ρ_t
3: **fim se**
4: **se** existe ρ_r segundo o Lema 4.5 **então**
5: **return** ρ_r
6: **fim se**
7: **se** existe ρ_t segundo o Lema 4.6 **então**
8: **return** ρ_t
9: **fim se**
10: **se** existe ρ_t segundo o Lema 4.7 **então**
11: **return** ρ_t
12: **fim se**
13: **se** existe ρ_t aplicável em um ciclo ímpar que aumente o número de ciclos pares **então**
14: **return** ρ_t
15: **fim se**
16: **return** $\emptyset$

4.3 Bancos de configurações

A partir deste ponto, não é possível aumentar nem o número de ciclos ímpares nem o número de ciclos da permutação com uma operação. Desta forma, transformaremos a permutação π em uma permutação simples $\hat{\pi}$, adicionando novos elementos de forma a quebrar os ciclos longos de $G(\pi)$, conforme explicado na Seção 2.3. Uma transformação de uma permutação π em uma permutação simples $\hat{\pi}$ é dita *segura* se o limitante inferior, que neste caso é $d_{rt}(\hat{\pi}) \geq \frac{n+1-c_{impar}(\pi)}{2}$, não é alterado. Lin e Xue [32] mostraram que toda permutação admite uma transformação segura para uma permutação simples. Esta transformação não garante que $d_{rt}(\pi) = d_{rt}(\hat{\pi})$, mas garante que ambas possuem o mesmo limitante inferior. Hannenhalli e Pevzner [26] mostraram que uma sequência de ordenação de $\hat{\pi}$ mimetiza uma sequência de ordenação de π com no máximo o mesmo número de operações.

Criação do banco de configurações com reversões e transposições

Temos então que $G(\hat{\pi})$ possui no máximo um ciclo par convergente, e outros 3-ciclos. Só pode existir um ciclo par pois, caso contrário, a etapa de análise dos ciclos encontraria uma operação que transformaria dois ciclos pares em dois ciclos ímpares. O conjunto de todas as configurações de 3-ciclos existentes é $\{(3,2,1),(3,1,2),(3,-2,1),(3,-1,2),(3,2,-1),(3,1,-2),(3,-2,-1),(3,-1,-2)\}$. Entretanto, o ciclo $(3,1,2)$ não pode existir em $G(\hat{\pi})$, uma vez que todos os ciclos orientados foram previamente eliminados pelas etapas anteriores. Além disso, os ciclos $\{(3,-1,-2),(3,-2,1),(3,2,-1)\}$ são equivalentes, e possuem $(3,-2,1)$ como representação canônica. Da mesma forma, os ciclos $\{(3,-1,2),(3,1,-2),(3,-2,-1)\}$ são equivalentes, e possuem $(3,-2,-1)$ como representação canônica.

Desta forma, fizemos combinações entre estes ciclos e procuramos uma sequência de operações que quando aplicadas nesta combinação alcançasse o fator de aproximação desejado, que fixamos em 1.8. Uma sequência de operações $s = \rho_1, \rho_2, ...\rho_k$ é dita *válida* se $\hat{\sigma} = \hat{\pi} \cdot \rho_1 \cdot \rho_2 \cdot ... \cdot \rho_k$ é uma permutação simples e $\frac{2k}{c_{impar}(\hat{\sigma}) - c_{impar}(\hat{\pi})} \leq 1.8$. Note que uma sequência não precisa ser necessariamente uma ordenação.

Para construir o banco de configurações, dada uma configuração de ciclos inicial, adicionamos novos ciclos a esta configuração iterativamente. Um novo ciclo era adicionado de tal forma que ele deveria cruzar uma aresta cinza de qualquer um dos ciclos já presentes na configuração. Assim, este procedimento sempre produziria configurações conexas tal que para qualquer ciclo C_i da configuração existe um ciclo C_j com pelo menos uma aresta cinza de C_j cruzando uma aresta cinza de C_i.

Note que nem todo grafo gerado pode ser obtido de uma permutação. Por exemplo, sejam x e y duas arestas pretas em um ciclo $C = (..., x, y, ...)$ ou $C = (..., -x, -y, ...)$, e seja g a aresta cinza que liga x e y. A aresta g é dita um *open gate* se não existe outra aresta cinza de C cruzando com g no grafo de ciclos. Uma configuração de ciclos que possui open gates não pode ser obtida por nenhuma permutação. Esta abordagem foi introduzida por Elias e Hartman [19] quando enumeraram todas as configurações de ciclos geradas a partir do ciclo $(3,2,1)$. Eles conseguiram provar que é possível encontrar

uma sequência de ordenação por transposições com fator de aproximação 1.375 quando o grafo de ciclos possui mais de 8 ciclos. Para combinações com menor quantidade de ciclos, eles apresentaram uma sequência com fator de aproximação 1.5. Nós utilizamos o grupo de configurações dos autores para reduzir o esforço computacional da criação do novo banco. Assim, o novo banco considera apenas combinações que possuem pelo menos um ciclo com uma aresta preta negativa, ou configurações com um ciclo par convergente.

Seja $G'(\pi)$ o grafo obtido a partir de $G(\pi)$ com a adição das *arestas azuis*, arestas que ligam o vértice i com o vértice $-i$, para $1 \leq i \leq n$, e uma aresta que liga o vértice 0 com o vértice $-(n+1)$. Após adicionar as arestas azuis, as arestas pretas são removidas de $G'(\pi)$. A Figura 4.13 mostra a transformação de $G(\pi)$ em $G'(\pi)$.

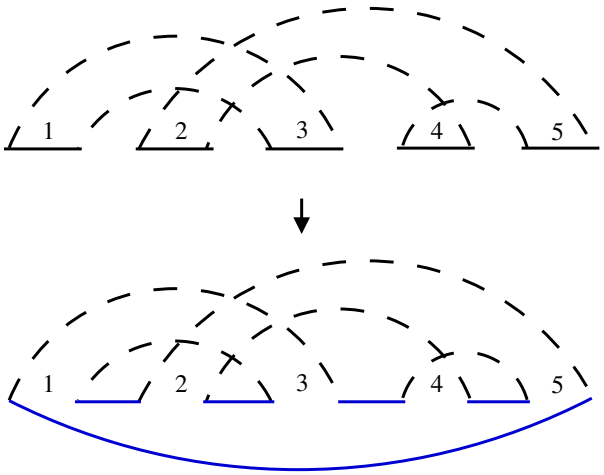


Figura 4.13: Transformação de $G(\pi)$ em $G'(\pi)$, adicionando as arestas azuis, e removendo as arestas pretas, representadas por retas pretas.

De maneira análoga a definição de open gates, Caprara [11] mostrou que se existe um caminho hamiltoniano em $G'(\pi)$, então $G(\pi)$ é gerado por uma permutação. A configuração $G'(\pi)$ da Figura 4.13 possui um caminho hamiltoniano, e, desta forma, é gerada por alguma permutação, que neste exemplo é a permutação $\pi = (-3\ -4\ +1\ -2)$. Já a configuração $G'(\pi)$ da Figura 4.14 não possui um caminho hamiltoniano, uma vez que sua decomposição gera dois ciclos, representados pelas cores verde e vermelho. Desta forma, não existe uma permutação π que gere $G(\pi)$.

Se uma configuração de ciclos A gera uma permutação, então todas as extensões que cruzam com arestas cinzas foram consideradas. Se $G'(\pi)$ não possuia um caminho hamiltoniano que percorresse todas as arestas azuis, então o grafo possuia necessariamente mais de um ciclo. Desta forma, somente foram consideradas as extensões que cruzassem as arestas cinzas do menor ciclo de $G'(\pi)$.

Iniciamos nosso banco de configurações com 3 configurações: uma configuração contendo o ciclo par $(2, 1)$, uma configuração contendo o ciclo ímpar canônico $(3, -2, 1)$ e uma configuração contendo o ciclo ímpar canônico $(3, -2 - 1)$. Nós estendemos cada configuração adicionando um dos 7 ciclos a seguir: $(3, 2, 1), (3, -2, 1), (3, -1, 2), (3, 2, -1)$, $(3, 1, -2), (3, -2, -1)$ e $(3, -1, -2)$. Após a extensão da configuração A em uma configuração B, verificamos se já não existia uma configuração canônica de B no banco de

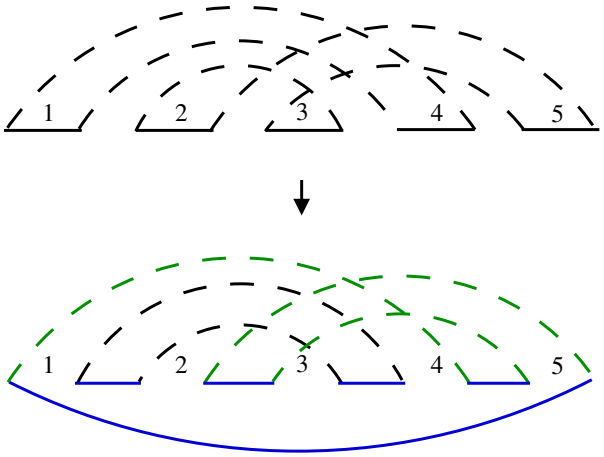


Figura 4.14: Transformação de $G(\pi)$ em $G'(\pi)$, com $G'(\pi)$ contendo 2 ciclos.

configurações. Além disso, não inserimos no banco as configurações equivalentes a B por rotação ou espelhamento. Se B não possuia uma configuração canônica no banco, tentávamos computar uma sequência válida para B utilizando um algoritmo *Branch and Bound* e se essa sequência válida fosse possível, B não precisaria receber mais extensões, uma vez que qualquer extensão C gerada a partir de B poderia utilizar a mesma sequência válida de B.

As configurações geradas possuem algumas propriedades:

- Possuem no máximo um ciclo de tamanho 2, uma vez que este ciclo é utilizado apenas na inicialização e não na extensão de uma configuração;
- Os ciclos ímpares devem ser um dos ciclos a seguir: $(3, 2, 1)$, $(3, -2, 1)$, $(3, -1, 2)$, $(3, 2, -1)$, $(3, 1, -2)$, $(3, -2, -1)$ e $(3, -1, -2)$;
- A configuração é conexa, ou seja, para qualquer ciclo C_i existe um ciclo C_j tal que uma aresta cinza de C_i cruza uma aresta cinza de C_j;
- A configuração não possui apenas ciclos do tipo $(3, 2, 1)$, uma vez que não existe uma configuração inicial com este ciclo.

Optamos por considerar apenas a representação canônica de cada configuração para reduzir o tempo de busca pelas sequências válidas. Foram criados dois bancos de configurações:

1. *banco par*, onde toda configuração foi gerada a partir do ciclo par $(2, 1)$;
2. *banco ímpar*, onde toda configuração foi gerada a partir dos ciclos $(3, -2, 1)$ e $(3, -2, -1)$.

A Figura 4.15 mostra um exemplo de uma extensão da configuração $(3, -2, -1)$ com outro ciclo $(3, -2, -1)$ tal que a configuração resultante possui uma sequência válida com fator 1.5.

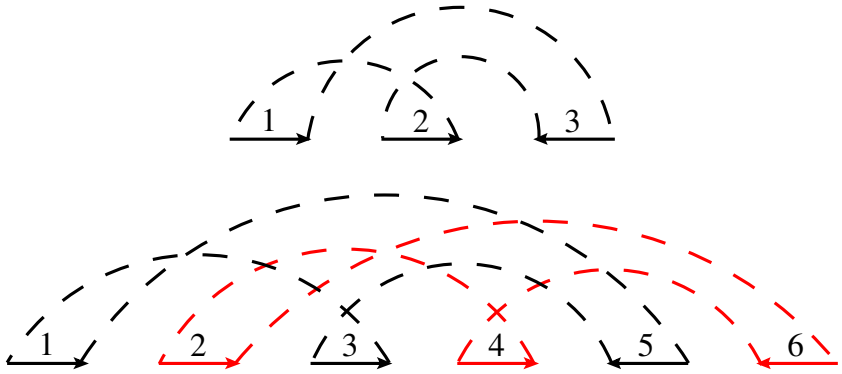


Figura 4.15: A configuração $(3, -2, -1)$, que não possui uma sequência válida com fator menor ou igual a 1.8. Uma extensão da configuração $(3, -2, -1)$ com um ciclo $(3, -2, -1)$, gerando a configuração $(6, -4, -2), (5, -3, -1)$, cuja sequência válida possui as operações $\rho_r(1, 4), \rho_t(2, 4, 6), \rho_r(1, 4)$ e fator 1.5.

Para o banco par, conseguimos encontrar uma sequência válida para toda configuração com até 4 ciclos. Além disso, a aproximação máxima obtida foi de 1.667, quando dentre 5 operações, 3 aumentaram o número de ciclos ímpares da configuração. Desta forma, não foi preciso gerar permutações com 5 ou mais ciclos neste banco.

Para o banco ímpar, ainda existem configurações com 4 ciclos no banco ímpar que não possuem uma sequência válida. Além disso, para configurações com 5 ciclos ou mais nosso algoritmo de enumeração de configurações ainda é proibitivo computacionalmente. Dado que gerar todas as configurações estendendo essas configurações com 4 ciclos e procurar uma sequência válida era proibitivo, decidimos não estender tais configurações.

Por este motivo, optamos por criar um novo banco composto por configurações com 3-ciclos, considerando agora que uma sequência será válida se esta sequência aplicada à permutação gera uma permutação que também é simples e $\frac{2k}{c_{impar}(\hat{\sigma}) - c_{impar}(\hat{\pi})} \leq 2.0$, ou seja, uma sequência que possua uma aproximação com fator 2.0.

O ciclo $(3, -2, -1)$ possui uma sequência válida com fator de aproximação 2.0 e, desta forma, ele não necessita ser candidato a ciclo inicial. Da mesma forma, os ciclos $(3, -1, 2)$, $(3, 1, -2)$ também possuem uma ordenação com aproximação 2.0, e por este motivo esses ciclos e o ciclo $(3, -2, -1)$ não fazem parte do conjunto de ciclos que estendem uma configuração. Assim, ficou definido que o ciclo inicial seria $(3, -2, 1)$, e estendemos as configurações que não possuíam uma sequência válida com um dos ciclos do conjunto $\{(3, 2, 1), (3, -2, 1), (3, 2, -1), (3, -1, -2)\}$. Para esse banco, todas as configurações com exatamente 3 ciclos possuem sequência válida e, deste modo, não foi preciso estender nenhuma dessas configurações. Com isso, todas as configurações que geram permutações possuem uma sequência válida em um dos bancos de configurações criados.

Desta forma, foi então criado o *banco par*, com todas as configurações e suas respectivas sequências válidas cujo fator de aproximação é, no máximo, 1.667, o *banco ímpar*, com configurações e suas respectivas sequências válidas, cujo fator de aproximação é no máximo 1.8, e o banco ímpar com todas as configurações e suas respectivas sequências válidas cujo fator de aproximação é no máximo 2.0, que definimos como *banco 2.0*.

A Tabela 4.1 sumariza a quantidade de configurações válidas em cada banco de configu-

rações criado, bem como do banco de configurações apresentado por Elias e Hartman [19], separadas pelo número de ciclos de cada banco.

Banco	Número de ciclos	Número de configurações	Total
par	2	2	1806
	3	161	
	4	1643	
ímpar	2	6	443748
	3	1468	
	4	442274	
2.0	1	1	257
	2	6	
	3	250	
EH	3	1	4002165
	4	22	
	5	5696	
	6	53898	
	7	377877	
	8	1450662	
	9	1077521	
	10	1034940	
	11	1140	
	12	264	
	13	144	

Tabela 4.1: Número de configurações com uma sequência válida para cada um dos bancos de configurações criados.

Busca de uma sequência válida

O próximo passo da heurística para ordenar a permutação é encontrar uma sequência de operações válidas para $\hat{\pi}$ em um de nossos bancos de configurações e aplicar as operações correspondentes em π.

Primeiramente, a heurística irá procurar uma sequência válida no banco criado por Elias e Hartman, que chamaremos de *banco EH*, que contém apenas configurações de combinações do ciclo $(3, 2, 1)$ e suas sequências válidas contendo transposições apenas. Se $G(\hat{\pi})$ contém apenas ciclos não-orientados convergentes, ele será estendido a partir de $(3, 2, 1)$ com os demais ciclos de $\hat{\pi}$, até que exista esta configuração estendida no banco EH, e a sequência válida para esta configuração encontrada será mimetizada em π, garantindo o fator de aproximação de no máximo 1.5. Note que é garantido que nossa heurística encontrará uma sequência válida, uma vez que Elias e Hartman provaram que é possível encontrar uma sequência de transposições com aproximação 1.375 quando o grafo de ciclos possui mais de 8 ciclos, e uma sequência de transposições com aproximação 1.5 quando o grafo de ciclos possui um número menor de ciclos.

Caso a heurística não encontre uma sequência no banco de Elias e Hartman, temos que $\hat{\pi}$ possui pelo menos um ciclo par ou um ciclo ímpar divergente. Assim, o próximo passo será, caso $\hat{\pi}$ contenha o ciclo $(2, 1)$, estender este ciclo com os demais ciclos de $\hat{\pi}$ até encontrar no banco par uma sequência válida, garantindo que o fator de aproximação seja no máximo 1.667. Da mesma forma que no banco de Elias e Hartman, é garantido que nossa heurística encontrará uma sequência válida, uma vez que o banco par foi gerado para configurações com até 4 ciclos, e existe uma sequência válida para qualquer configuração (gerada por uma permutação) que contenha o ciclo $(2, 1)$.

Caso a heurística não encontre uma sequência no banco par, então $\hat{\pi}$ possui apenas ciclos ímpares e obrigatoriamente um ciclo ímpar divergente. Assim, será escolhido um ciclo de $\hat{\pi}$ como configuração inicial, e esta configuração será estendida com os demais ciclos procurando, a cada extensão, uma sequência válida no banco ímpar até que a configuração seja estendida com no máximo mais 3 ciclos. Caso encontre uma sequência válida, esta sequência será mimetizada em π, garantindo um fator de aproximação de no máximo 1.8. Caso contrário, não existe uma sequência válida com aproximação 1.8.

Neste ponto, $\hat{\pi}$ contém apenas ciclos ímpares, sendo pelo menos um deles obrigatoriamente divergente, e não existe uma extensão com até 3 novos ciclos em $\hat{\pi}$ com uma sequência válida cuja aproximação seja 1.8. Sendo assim, vamos recorrer ao banco 2.0. Da mesma forma que no passo anterior, será escolhido um ciclo de $\hat{\pi}$ como configuração inicial, e esta configuração será estendida com os demais ciclos procurando, a cada extensão, uma sequência válida no banco 2.0. Dado que este banco possui uma sequência válida para todas as configurações que geram uma permutação com até 3 ciclos, é garantido que a heurística encontrará uma sequência válida com fator de aproximação 2.0.

Por exemplo, considere a permutação sem sinais $\sigma = (8\ 9\ 1\ 6\ 2\ 3\ 4\ 5\ 7\ 10)$. O algoritmo de decomposição de ciclos de Lin e Jiang [31] retorna a decomposição em ciclos G apresentada na Figura 4.16.

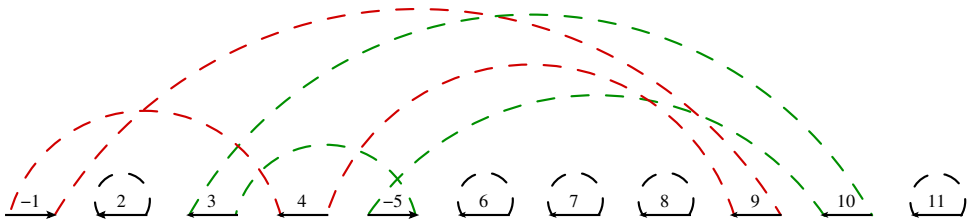


Figura 4.16: Decomposição de ciclos da permutação sem sinais $\sigma = (8\ 9\ 1\ 6\ 2\ 3\ 4\ 5\ 7\ 10)$ pelo algoritmo de Lin e Jiang [31], correspondente à decomposição de ciclos da permutação com sinais $\pi = (+8\ +9\ -1\ +6\ +2\ +3\ +4\ +5\ -7\ +10)$.

Desta forma, temos que esta decomposição $G(\sigma)$ obtida corresponde à decomposição $G(\pi)$ da permutação com sinais $\pi = (+8\ +9\ -1\ +6\ +2\ +3\ +4\ +5\ -7\ +10)$. Além disso, π é uma permutação simples, uma vez que $G(\pi)$ possui apenas ciclos curtos e, por este motivo, $G(\hat{\pi}) = G(\pi)$.

Note que em $G(\pi)$ não existe nenhuma configuração de ciclos que admita o aumento do número de ciclos ímpares com uma operação ou o aumento do número de ciclos com uma operação. Desta forma, a heurística buscará por sequências válidas nos bancos de

configurações criados. Como existem apenas ciclos não orientados divergentes em $G(\pi)$, a heurística não realizará a busca no *banco EH* que contém apenas ciclos convergentes. Além disso, $G(\pi)$ só possui ciclos ímpares e, por este motivo, a heurística não realizará a busca no *banco par*. A heurística procurará então no *banco ímpar* uma configuração com sequência válida, iniciando com o ciclo $(10, -5, 3)$, que ao realizar uma nova rotulação corresponde ao ciclo $(3, -2, 1)$. Como não existe uma sequência válida com aproximação de no máximo 1.8 para este ciclo, a heurística estende este ciclo com o outro ciclo ímpar de $G(\pi)$, ficando então com a configuração $[(10, -5, 3), (9, 4, -1)]$ que, ao realizar uma nova rotulação, corresponde à configuração $[(6, -4, 2), (5, 3, -1)]$. Também não existe uma sequência válida com aproximação 1.8 para esta configuração e, como não existem mais ciclos para estender a configuração, a heurística irá procurar no *banco 2.0*. Neste banco, também não existe uma sequência válida com aproximação 2.0 para a configuração $[(3, -2, 1)]$. Entretanto, existe uma sequência válida para a configuração $[(10, -5, 3), (9, 4, -1)]$, cuja sequência de operações é $[\rho_r(1, 2), \rho_t(2, 4, 6), \rho_r(4, 5), \rho_r(3, 4)]$. Ao ser mimetizada para π, esta sequência torna-se $[\rho_r(1, 3), \rho_t(2, 5, 10), \rho_r(7, 9), \rho_r(6, 7)]$. Abaixo encontra-se a aplicação da sequência de operações mimetizada em π.

$$\begin{aligned}
\pi &= (+8\ +9\ -1\ +6\ +2\ +3\ +4\ +5\ -7\ +10) \cdot \rho_r(1, 3) \\
&= (+1\ -9\ -8\ +6\ +2\ +3\ +4\ +5\ -7\ +10) \cdot \rho_t(2, 5, 10) \\
&= (+1\ +2\ +3\ +4\ +5\ -7\ -9\ -8\ +6\ +10) \cdot \rho_r(7, 9) \\
&= (+1\ +2\ +3\ +4\ +5\ -7\ -6\ +8\ +9\ +10) \cdot \rho_r(6, 7) \\
&= (+1\ +2\ +3\ +4\ +5\ +6\ +7\ +8\ +9\ +10) = \iota
\end{aligned}$$

Como após a aplicação de todas as operações da sequência mimetizada a permutação foi ordenada, a heurística para de procurar operações e retorna o número de operações aplicadas.

Considerações gerais

Qualquer sequência de operações encontrada nesta etapa garante uma 2-aproximação. Ao contrário das duas etapas anteriores, esta etapa sempre encontrará uma sequência de operações que deve ser aplicada em π. A Função 17 resume a ordem dos passos seguidos por esta etapa.

É importante notar que o grafo de ciclos de uma permutação σ sem sinais retornado pelo algoritmo de decomposição de ciclos sempre corresponderá ao grafo de ciclos de uma permutação π com sinais, e por este motivo reversões unitárias poderão ser aplicadas no grafo de ciclos. Como reversões unitárias não são permitidas em permutações sem sinais, se a heurística decide aplicar uma reversão unitaria esta reversão será aplicada no grafo de ciclos mas não entrará no cômputo do número de operações para ordenar σ. A Heurística 18 descreve os passos realizados por nossa heurística.

Além disso, a Heurística 18 garante uma 2-aproximação apenas para a permutação com sinais π, que foi obtida a partir do grafo de ciclos $G(\sigma)$ da permutação σ dada como entrada. No entanto, para qualquer permutação σ com sinais temos que $\pi = \sigma$ e, desta forma, a heurística garante uma 2-aproximação também para σ.

Função 17 BuscaBanco($G(\pi)$)

```
1: seq ← ∅
2: ext = 0
3: se G(π) possui apenas 3-ciclos convergentes então
4:     enquanto seq = ∅ faça
5:         Estenda G(π)
6:         seq ← sequência válida do banco EH
7:     fim enquanto
8: fim se
9: se G(π) possui um ciclo par então
10:    enquanto seq = ∅ faça
11:        Estenda G(π)
12:        seq ← sequência válida do banco par
13:    fim enquanto
14: fim se
15: enquanto seq = ∅ e ext < 3 faça
16:    Estenda G(π)
17:    ext ← ext + 1
18:    seq ← sequência válida do banco ímpar
19: fim enquanto
20: enquanto seq = ∅ faça
21:    Estenda G(π)
22:    seq ← sequência válida do banco 2.0
23: fim enquanto
24: return seq
```

No caso em que σ é uma permutação sem sinais, geralmente $\pi \neq \sigma$. Pelo Lema 2.10 temos que $d_{rt}(\sigma) \geq \frac{n+1-c(\sigma)}{2}$ e $d_{\overline{r}t}(\pi) \geq \frac{n+1-c(\pi)}{2}$. Além disso, as arestas que criam ciclos unitários não fazem parte de $G_b(\sigma)$, e o número de ciclos unitários é igual a $(n+1-b_r(\sigma))$. Desta forma, temos que $d_{rt}(\sigma) = \frac{n+1-(n+1-b_r(\sigma)+c_{2+}(\sigma))}{2} = \frac{b_r(\sigma)-c_{2+}(\sigma)}{2}$, onde $c_{2+}(\sigma)$ é o número de k-ciclos, com $k \geq 2$. Como a heurística é uma 2-aproximação para a permutação com sinais π obtida, temos que o fator de aproximação para ordenar permutações sem sinais será $r = \frac{2d_{\overline{r}t}(\pi)}{d_{rt}(\sigma)} = \frac{2(b_r(\sigma)-c_{2+}(\pi))}{b_r(\sigma)-c_{2+}(\sigma)}$.

Um algoritmo l-aproximado para decomposição de ciclos garante que $\frac{b_r(\sigma)-c_{2+}(\pi)}{b_r(\sigma)-c_{2+}(\sigma)} \leq l$. Desta forma, temos que o fator de aproximação $r \leq 2l$. Como utilizamos o algoritmo de Lin e Jiang [31], temos que $l = 1.4193$ e, desta forma, o fator de aproximação da heurística para ordenar permutações sem sinais é $r \leq 2.8386$.

4.4 Resultados Finais

Nesta seção, apresentaremos os resultados obtidos pela nossa heurística, que será denotada por **HEU**, e faremos uma comparação com a heurística Breakpoints, do Capítulo 3, que foi a heurística que produziu os melhores resultados, e com algoritmos existentes na literatura.

De modo a verificar a qualidade de nossa heurística, executamos o mesmo conjunto de permutações criado com o Algoritmo 14 também com nossa heurística, tanto com permu-

Heurística 18 HeuristicaFinal(σ)

```
1: dist ← 0
2: sig ← True
3: se σ é uma permutação sem sinais então
4:     G(σ) ← decomposição em ciclos (Algoritmo de Lin e Jiang [31])
5:     sig ← False
6: senão
7:     G(σ) ← decomposição única de ciclos
8: fim se
9: π ← permutação com sinais obtida a partir de G(σ).
10: enquanto (π ≠ ι) faça
11:     op ← ∅
12:     op ← RemoveBreakpoints(G(π))
13:     se op = ∅ então
14:         op ← AumentaCiclos(G(π))
15:     fim se
16:     se op = ∅ então
17:         op ← BuscaBanco(G(π))
18:     fim se
19:     para cada operação ρ em op faça
20:         π ← π · ρ
21:         se ρ não é uma reversão unitária ou sig = True então
22:             dist ← dist + 1
23:         fim se
24:     fim para
25: fim enquanto
26: return dist
```

tações com sinais, quanto com permutações sem sinais. No total foram gerados 190000 permutações com sinais e a mesma quantidade de permutações sem sinais, com tamanhos de permutações variando entre 10 e 100, em intervalos de tamanho 5. Note que 20 operações de reversões e transposições são suficientes para ordenar qualquer permutação gerada.

As figuras 4.17 e 4.18 mostram as porcentagens médias de reversões e transposições utilizadas pela heurística para ordenar permutações com sinais e permutações sem sinais, respectivamente. Nossa heurística usa, em média, um pouco mais de transposições do que reversões. Isso se justifica pelo fato de que dois dos três passos da Etapa de Remoção de Breakpoints aplicam transposições e apenas um passo aplica reversões. Além disso, quatro dos cinco passos da Etapa de Análise de Ciclos aplicam transposições e apenas um passo aplica reversões.

As figuras 4.19 e 4.20 mostram as porcentagens médias de operações realizadas por cada etapa da heurística para ordenar permutações com sinais e permutações sem sinais, respectivamente. Note que, em média, mais da metade das operações foram realizadas pela etapa de Remoção de Breakpoints, tanto para permutações com sinais quanto permutações sem sinais. Além disso, a quantidade média de operações nesta etapa aumenta conforme o tamanho das permutações. Para permutações com sinais de tamanho 100, mais de 75%

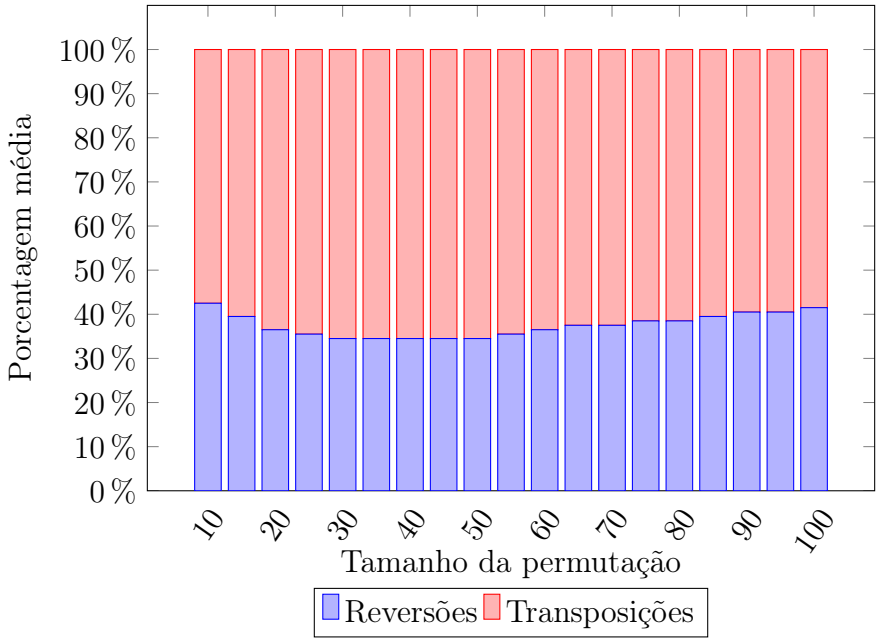


Figura 4.17: Porcentagem média de operações de reversões e transposições utilizadas pela Heurística de Grafo de Ciclos para ordenar permutações com sinais.

das operações em média foram realizadas pela etapa de Remoção de Breakpoints. Para permutações sem sinais de tamanho 100, cerca de 70% das operações em média também foram realizadas por esta etapa.

Para permutações com sinais, a etapa de Análise dos Ciclos possui a segunda maior frequência de operações em permutações pequenas, mas a quantidade média de operações encontradas por esta etapa diminui conforme o tamanho da permutação aumenta. Já para permutações sem sinais, a quantidade média de operações realizadas por esta etapa é menor para permutações pequenas. Uma explicação plausível é que os ciclos destas permutações são obtidos através de um algoritmo de decomposição de ciclos que prioriza sempre ciclos unitários, 2-ciclos e 3-ciclos, nesta ordem. Assim, o número de ciclos unitários em permutações sem sinais é sempre o maior possível. Além disso, o algoritmo de decomposição de ciclos não faz distinção entre ciclos orientados e não orientados, nem entre ciclos divergentes ou convergentes, e, desta forma, ele pode retornar uma decomposição de ciclos onde a maioria dos ciclos não se encaixam nos passos desta etapa.

A etapa de busca nos bancos de configurações estão apresentadas separadamente por tipo de banco de configurações. O *banco EH* foi o banco de configurações menos utilizado em média, tanto para permutações com sinais quanto para permutações sem sinais, o que pode ser explicado pelo fato do grafo da permutação possuir muitas vezes arestas divergentes, que não existem neste banco.

O *banco ímpar* foi, na média, o banco mais utilizado pela heurística. Além disso, conforme o tamanho da permutação aumenta, a quantidade de operações encontradas neste banco diminui de forma muito sutil, tanto para permutações com sinais quanto para permutações sem sinais.

O *banco par* foi o segundo banco mais utilizado pela heurística tanto para permutações

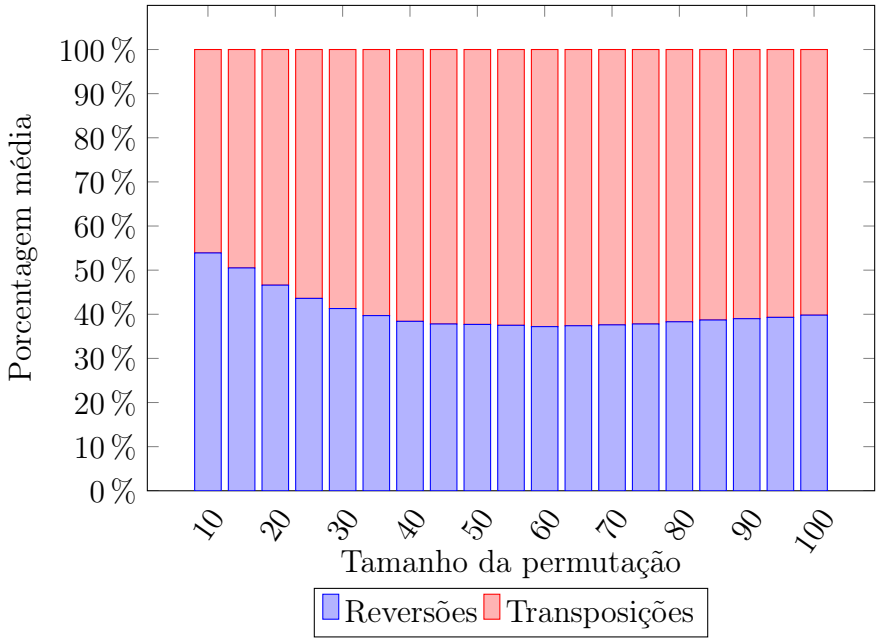


Figura 4.18: Porcentagem média de operações de reversões e transposições utilizadas pela Heurística de Grafo de Ciclos para ordenar permutações sem sinais.

com sinais quanto para permutações sem sinais, sendo mais utilizado pelas permutações sem sinais. Uma explicação plausível, como falado anteriormente, é que o algoritmo de decomposição de ciclos prioriza a criação de ciclos unitários seguido de 2-ciclos. Para permutações sem sinais pequenas este banco foi o que retornou, em média, a maior quantidade de sequências válidas, mas a quantidade de sequências encontradas diminui conforme o tamanho da permutação aumenta.

Por fim, o *banco 2.0* foi um dos bancos menos utilizados em média, com quantidade de operações aplicadas maior apenas que do *banco EH*. Além disso, conforme o tamanho da permutação aumenta, a quantidade de operações obtidas por este banco diminui, tanto para permutações com sinais quanto permutações sem sinais. Para permutações com sinais e sem sinais de tamanho 100, por exemplo, menos de 1.3% das operações foram obtidos pelo banco 2.0 em média.

A nossa heurística foi comparada com a heurística Breakpoints, que será denotada por **BP**, e também com quatro algoritmos já existentes na literatura. O primeiro é o algoritmo apresentado por Walter, Dias e Meidanis [39], que será denotado por **Walter**, que ordena por reversões e transposições permutações com sinais com fator de aproximação 2.0, utilizando para isso o grafo de ciclos, e permutações sem sinais com fator de aproximação 3, utilizando apenas breakpoints.

O segundo é o algoritmo apresentado por Rahman, Shatabda e Hasan [35], que será denotado por **Rahman**, que ordena permutações sem sinais por reversões e transposições com fator de aproximação 2k, onde k é o fator de aproximação do algoritmo utilizado pela decomposição de ciclos. Nós adaptamos este algoritmo para permutações com sinais, com fator de aproximação 2.0, de modo a aproveitar este algoritmo em nossa comparação de permutações com sinais também.

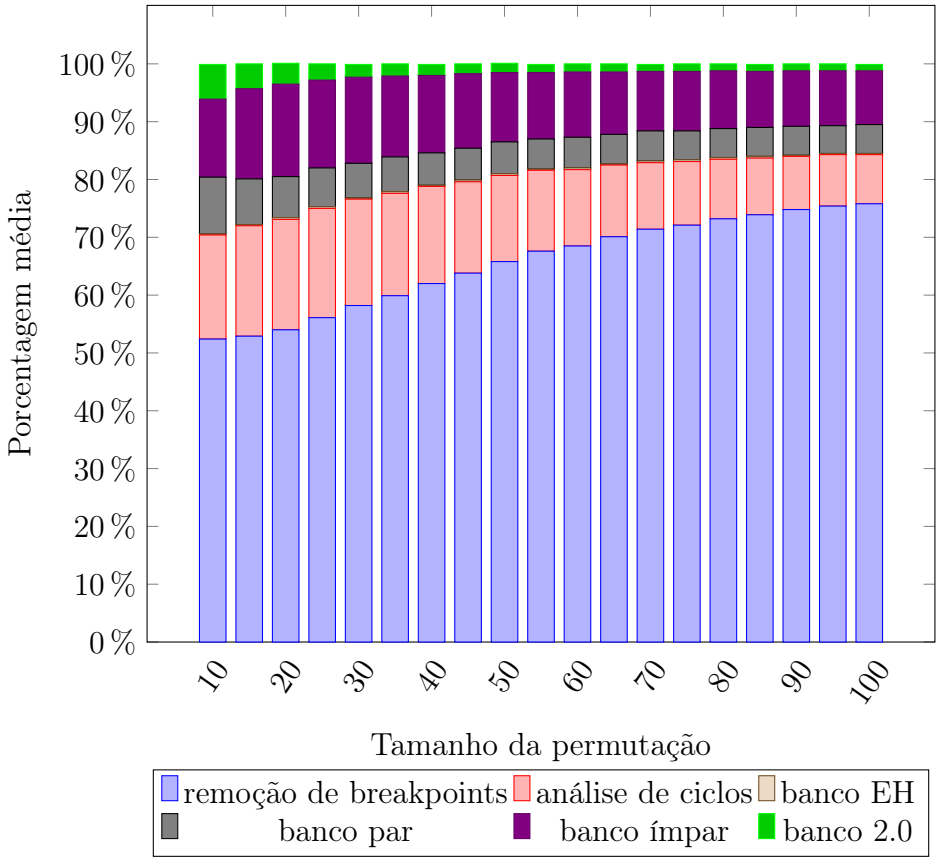


Figura 4.19: Porcentagem média de operações realizadas por cada etapa da heurística para ordenar permutações com sinais.

O terceiro algoritmo será o algoritmo que ordena permutações por reversões, e será denotado por **REV**. O programa *GRIMM* [38] fornece a solução ótima para o problema de ordenar permutações com sinais por reversões. Note que aplicar uma transposição $\rho_t(i,j,k)$ é equivalente à aplicar a sequência de três reversões $\rho_r(i,k-1)$, $\rho_r(i,j-1)$, $\rho_r(j,k-1)$. Deste modo, os resultados de **REV** fornecem uma aproximação para nosso problema: dado π e uma sequência ótima de ordenações por reversões $d_r(\pi)$, $d_{rt}(\pi) \geq \frac{d_r(\pi)}{3}$. De modo a comparar também os resultados para permutações sem sinais, utilizamos a decomposição de ciclos de Lin e Jiang [31], cujo fator de aproximação é de 1.4193, e para cada permutação π, obtivemos a permutação com sinais σ a partir da decomposição do grafo de ciclos, como explicado no início do Capítulo 4. Uma vez que nosso problema aceita tanto reversões quanto transposições, uma sequência que aplica apenas reversões é uma solução válida.

Da mesma forma que estamos utilizando um algoritmo de ordenação de permutações por reversões, o quarto algoritmo será o algoritmo que ordena permutações por transposições apresentado por Bafna e Pevzner [3], e será denotado por **TRANS**. Entretanto, o problema de ordenação por transposições é definido apenas para permutações sem sinais. Neste caso, para toda permutação π com sinais, identificamos sequências maximais de elementos negativos e aplicamos uma reversão nestas sequências gerando, desta forma, uma permutação π' cujos elementos são todos positivos. Assim, a permutação pode ordenar

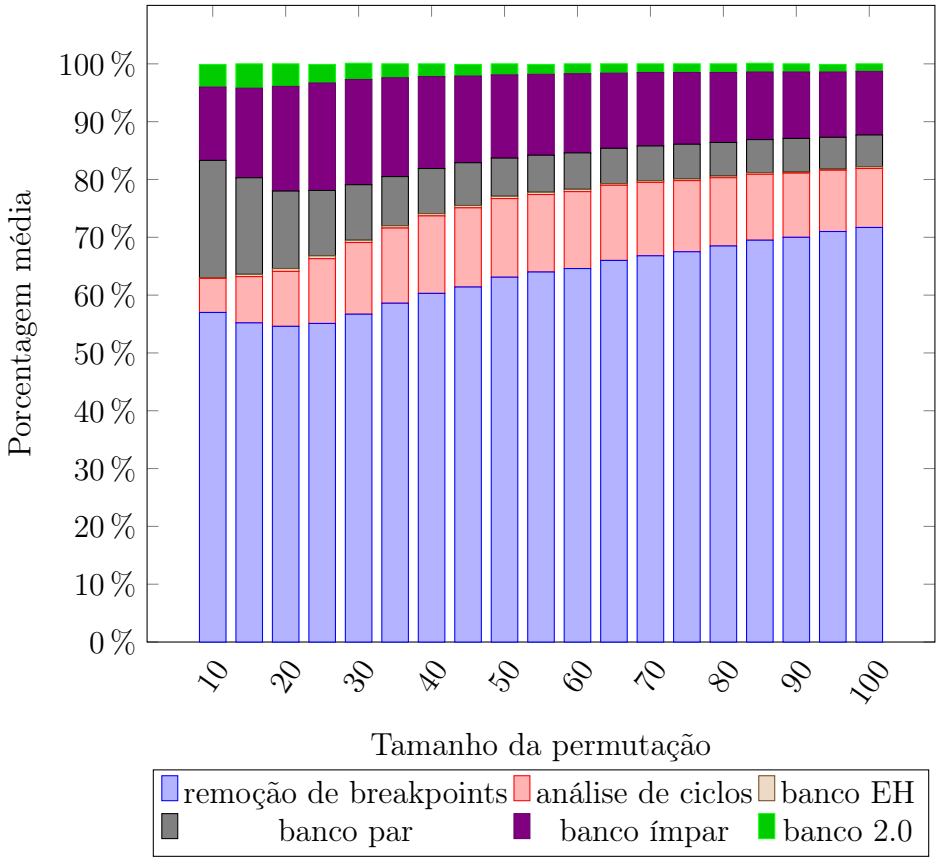


Figura 4.20: Porcentagem média de operações realizadas por cada etapa da heurística para ordenar permutações sem sinais.

π' uma vez que transposição não troca os sinais dos elementos.

Para os quatro algoritmos acima, os mesmos conjuntos de permutações com sinais e sem sinais foram fornecidos como entrada, e as distâncias médias de cada heurística e de cada algoritmo encontram-se nas tabelas 4.2 e 4.3.

Para permutações com sinais, a Heurística de Grafo de Ciclos retornou na média os menores valores para todos os tamanhos de permutações utilizadas dos testes. Além disso, dado que qualquer permutação utilizada nos testes pode ser ordenada com no máximo 20 operações, **HEU** obteve resultados muito bons, cuja maior distância média foi de 20.23, para permutações de tamanho 100. As segundas menores distâncias médias foram obtidas por **BP**, uma heurística simples que remove o maior número de breakpoints de forma gulosa. Comparado com os algoritmos existentes na literatura que ordenam permutações com sinais por reversões e transposições, **HEU** retornou resultados pelo menos 10% menores que **Rahman** e **Walter** em média, para qualquer tamanho de permutação. Além disso, para permutações maiores que 20, nossa heurística retornou resultados em média 30% menores que **Walter**. Os resultados obtidos por **Walter** foram um pouco piores também que os resultados obtidos por **TRANS**, que só permite transposições. Entretanto, é importante lembrar que no caso específico de permutações com sinais foram aplicadas reversões em elementos consecutivos negativos antes do algoritmo **TRANS** ordená-las, o que certamente colaborou com os resultados obtidos. Os piores resultados foram obtidos

n	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
BP	6.90	10.03	12.59	14.59	16.17	17.38	18.34	19.07	19.66	20.08	20.50	20.74	20.94	21.18	21.26	21.42	21.52	21.60	21.68
Walter	8.38	12.75	16.51	19.58	22.00	23.82	25.25	26.28	27.00	27.53	27.94	28.25	28.45	28.66	28.77	28.88	28.98	29.06	29.13
Rahman	7.26	10.87	13.89	16.32	18.24	19.60	20.62	21.30	21.71	22.00	22.19	22.35	22.37	22.47	22.49	22.52	22.58	22.61	22.66
REV	8.76	13.17	16.95	20.06	22.50	24.32	25.75	26.78	27.51	28.08	28.48	28.77	28.97	29.18	29.29	29.41	29.49	29.54	29.62
TRANS	7.38	10.85	13.81	16.28	18.32	19.95	21.36	22.52	23.54	24.34	25.11	25.67	26.29	26.78	27.13	27.64	27.93	28.22	28.55
HEU	6.37	9.16	11.48	13.38	14.85	16.01	16.91	17.60	18.13	18.54	18.91	19.18	19.39	19.61	19.75	19.89	20.01	20.13	20.23

Tabela 4.2: Distâncias médias para permutações com sinais.

n	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
BP	4.48	7.07	9.45	11.53	13.26	14.73	16.00	17.03	17.93	18.73	19.31	19.86	20.33	20.74	21.06	21.34	21.55	21.75	21.95
Walter	5.31	9.08	12.73	15.94	18.71	21.01	23.06	24.74	26.24	27.55	28.62	29.58	30.44	31.22	31.89	32.49	32.92	33.45	33.85
Rahman	4.98	7.87	10.61	13.09	15.21	17.00	18.55	19.77	20.69	21.49	22.09	22.57	22.91	23.12	23.32	23.38	23.46	23.48	23.43
REV	5.76	9.21	12.49	15.49	17.97	20.10	21.93	23.42	24.62	25.65	26.46	27.12	27.65	28.05	28.40	28.65	28.83	29.00	29.15
TRANS	4.66	7.16	9.49	11.71	13.76	15.72	17.63	19.45	21.22	23.01	24.76	26.50	28.14	29.89	31.56	33.39	34.92	36.67	38.36
HEU	4.74	7.36	9.66	11.64	13.28	14.65	15.79	16.73	17.50	18.14	18.67	19.01	19.45	19.74	20.00	20.18	20.30	20.40	20.54

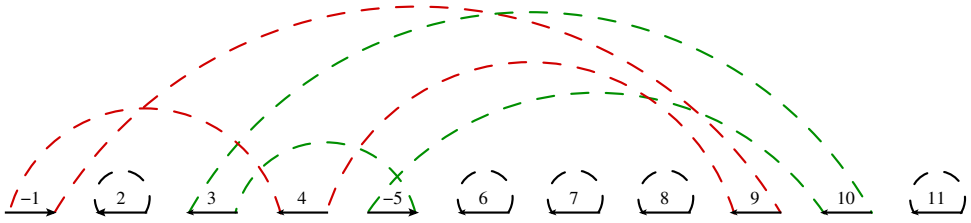
Tabela 4.3: Distâncias médias para permutações sem sinais.

pelo algoritmo **REV**, que retorna sempre a distância exata da ordenação por reversões. Os resultados de **REV** foram em média 50% maiores que os resultados obtidos por nossa heurística.

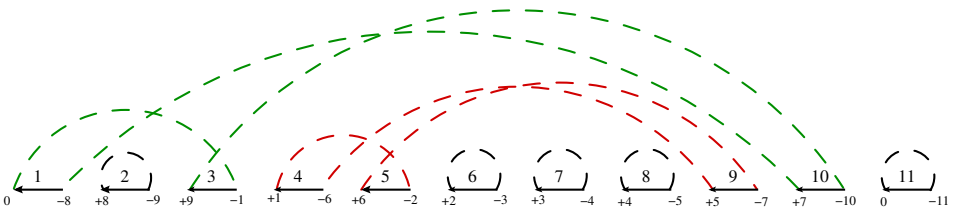
Para permutações sem sinais, a Heurística de Grafo de Ciclos retornou os melhores resultados para permutações maiores que 30. Para permutações menores os resultados de **HEU** foram no máximo 6% maiores que **BP** e **TRANS**. Para as permutações utilizadas nos testes com tamanho até 30, o número médio de breakpoints em uma permutação era maior que 75% do número máximo possível, o que certamente colaborou para o bom desempenho da heurística **BP** e do algoritmo **TRANS**.

Uma outra explicação plausível para **HEU** obter distâncias médias para permutações sem sinais pequenas maiores que **BP** deve-se ao fato do algoritmo utilizado para decomposição em ciclos não distinguir características dos ciclos criados, como orientado ou convergente, e apenas retornar um conjunto com a maior quantidade de ciclos unitários, 2-ciclos e 3-ciclos. Por exemplo, para a permutação sem sinais $\sigma = (8\ 9\ 1\ 6\ 2\ 3\ 4\ 5\ 7\ 10)$, o algoritmo de Lin e Jiang [31] retorna o grafo de ciclos da Figura 4.21(a), que possui $c(\sigma) = 7$. Os ciclos $C_1 = (10, -5, 3)$ e $C_2 = (9, 4, -1)$ não são ciclos removidos pelas etapas de Remoção de Breakpoints e Análise dos Ciclos. Além disso, a sequência de operações que ordena estes ciclos possui aproximação 2.0, e a sequência de operações necessárias para ordenar σ com base neste grafo de ciclos é $\rho_r(1,3), \rho_t(2,5,10), \rho_r(7,9)$ e $\rho_r(6,7)$.

Entretanto, a Figura 4.21(b) exibe outra decomposição de σ com $c(\sigma) = 7$, e com os ciclos $C_1 = (10, 1, 3)$ e $C_2 = (9, 4, 5)$, que são ciclos orientados com duas distâncias ímpares, removidos pela etapa de Remoção de Breakpoints. Desta forma, ordenar σ com base no grafo de ciclos da Figura 4.21(b) requer a aplicação de metade do número de operações necessárias pelo grafo de ciclos Figura 4.21(a), bastando aplicar as transposições $\rho_t(1,3,10)$ e $\rho_t(2,3,7)$.



(a) Grafo de ciclos criado pelo algoritmo.



(b) Grafo de ciclos contendo apenas ciclos unitários e ciclos orientados convergentes.

Figura 4.21: Duas possíveis decomposições do grafo de breakpoints $G_b(\sigma)$ com 7 ciclos da permutação sem sinais $\sigma = (8\ 9\ 1\ 6\ 2\ 3\ 4\ 5\ 7\ 10)$.

O algoritmo **TRANS** obteve bons resultados para permutações pequenas, mas obteve os piores resultados para as permutações maiores que 80, com distâncias médias quase duas

vezes maiores que o necessário. Isso se deve ao fato de que ordenar por transposições um trecho invertido (causado por uma reversão) requer mover cada elemento individualmente. Já o algoritmo **REV** obteve resultados medianos em todos os conjuntos de permutações testadas, inclusive nos conjuntos de permutações grandes. Isso se explica pelo fato de que, ao contrário de uma transposição, uma reversão consegue simular uma transposição com 3 operações apenas. Comparado com os algoritmos existentes na literatura que ordenam permutações sem sinais por reversões e transposições, **HEU** retornou sempre os menores resultados. Comparado com **Rahman**, os resultados de **HEU** foram em média 15% menores para permutações maiores que 30. Comparado com **Walter**, os resultados de **HEU** foram em média 50% menores para permutações maiores que 50. Além disso, dado que qualquer permutação utilizada nos testes pode ser ordenada com no máximo 20 operações, **HEU** obteve resultados também muito satisfatórios para permutações sem sinais, cuja maior distância média foi de 20.54, para permutações de tamanho 100.

Além da distância média, foram geradas também para cada algoritmo o fator de aproximação médio, o fator de aproximação máximo e a porcentagem de permutações do conjunto de testes com menor distância dentre os algoritmos. Para o cálculo do fator de aproximação médio e máximo do conjunto de permutações com sinais foi utilizado a fórmula dada pelo Teorema 2.12, que considera o número de ciclos do grafo de ciclos da permutação. Para o cálculo do fator de aproximação médio e máximo do conjunto de permutações sem sinais foi utilizado a fórmula dada pelo Teorema 2.13, que considera o número de breakpoints de reversão sem sinal da permutação.

Os resultados do fator de aproximação médio, fator de aproximação máximo e porcentagem de permutações com distância mínima para permutações com sinais estão apresentados nas figuras 4.22, 4.23 e 4.24, respectivamente. A heurística **HEU** obteve sempre o menor fator de aproximação médio, ficando este fator sempre abaixo de 1.4, e, para permutações maiores que 50, o fator de aproximação médio ficou abaixo de 1.2. A heurística **BP** também obteve bons resultados, ficando em média com um fator de aproximação 0.1 maior que da heurística **HEU**. Os algoritmos **REV** e **Walter** ficaram com os fatores de aproximação médios mais altos, sempre acima de 1.6.

Pela Figura 4.23 podemos observar que o fator de aproximação máximo da heurística **HEU** também foi o menor para todos os tamanhos de permutação sem sinais, com fator 2.0 para permutações com tamanho 10, e este fator diminuiu gradativamente conforme o tamanho das permutações aumentaram. Note também que **HEU** garante o fator de aproximação máximo 2.0 para ordernar a permutação com sinais. Os fatores de aproximação máximos de **Walter** e **Rahman** foram sempre menores ou iguais a 2.0. Os fatores de aproximação de **TRANS** ficaram sempre acima de 2.0, e de **REV** sempre ficaram exatamente em 2.0.

Para cada permutação com sinais utilizada nos testes foi tabulado qual algoritmo retornou o melhor resultado. Estas informações foram computadas e a Figura 4.24 apresenta a porcentagem de permutações para cada algoritmo com o melhor resultado do conjunto. A heurística **HEU** obteve também a maior porcentagem de permutações com menor distância de ordenação, com valores estáveis entre 80% e 90%. **REV**, **TRANS** e **Walter** ficaram na média com as menores quantidades de permutações com menor distância de ordenação, com valores muito próximos de 0%.

Os resultados do fator de aproximação médio, fator de aproximação máximo e porcentagem de permutações com a menor distância obtida para permutações sem sinais estão apresentadas nas figuras 4.25, 4.26 e 4.27, respectivamente. A heurística **HEU** obteve o menor fator de aproximação médio para todas as permutações com tamanho igual ou maior que 35. Além disso, o fator de aproximação médio ficou sempre entre 1.5 e 1.6, para todos os tamanhos de permutações. A heurística **BP** também obteve fatores de aproximações baixos, principalmente para permutações de tamanhos menores ou iguais a 30, onde os fatores de aproximação médios foram os menores dentre os algoritmos analisados. **Rahman** obteve fatores de aproximação medianos, com valores entre 1.5 e 1.9. **TRANS** obteve o segundo menor fator de aproximação para permutações pequenas, entretanto, para permutações grandes, o fator de aproximação ultrapassou 2.8.

Dentre os maiores fatores de aproximação obtidos, como mostra a Figura 4.26, **HEU** obteve os melhores resultados, ficando com fator de aproximação acima de 2.0 apenas para permutações de tamanho 30, onde obteve o fator de aproximação de 2.142. Note que este fator ficou abaixo do fator de aproximação teórico de **HEU** para permutações sem sinais, que é de 2.8386. **BP** ficou como o segundo melhor resultado, obtendo o maior fator de aproximação igual a 2.5 para permutações de tamanho 10. **Rahman**, **REV** e **Walter** ficaram com fatores de aproximação entre 2.5 e 3.0. Já **TRANS** obteve os maiores fatores de aproximação, alcançando fatores de aproximação acima de 4.5 para permutações grandes.

Para cada permutação sem sinais utilizada nos testes foi tabulado qual algoritmo retornou o melhor resultado. Estas informações foram computadas e a Figura 4.27 apresenta a porcentagem de permutações para cada algoritmo com o melhor resultado do conjunto. A heurística **HEU** obteve também a maior porcentagem de permutações com menor distância de ordenação para permutações maiores que 30, chegando a valores acima de 80% do total. **TRANS** e **BP** obtiveram a maior quantidade de melhores distâncias para permutações pequenas, mas para permutações grandes apenas **BP** obteve bons resultados, com porcentagem de melhores resultados entre 40% e 60%. **Rahman**, **Walter** e **REV** foram os algoritmos que conseguiram a menor quantidade de melhor distância de ordenação.

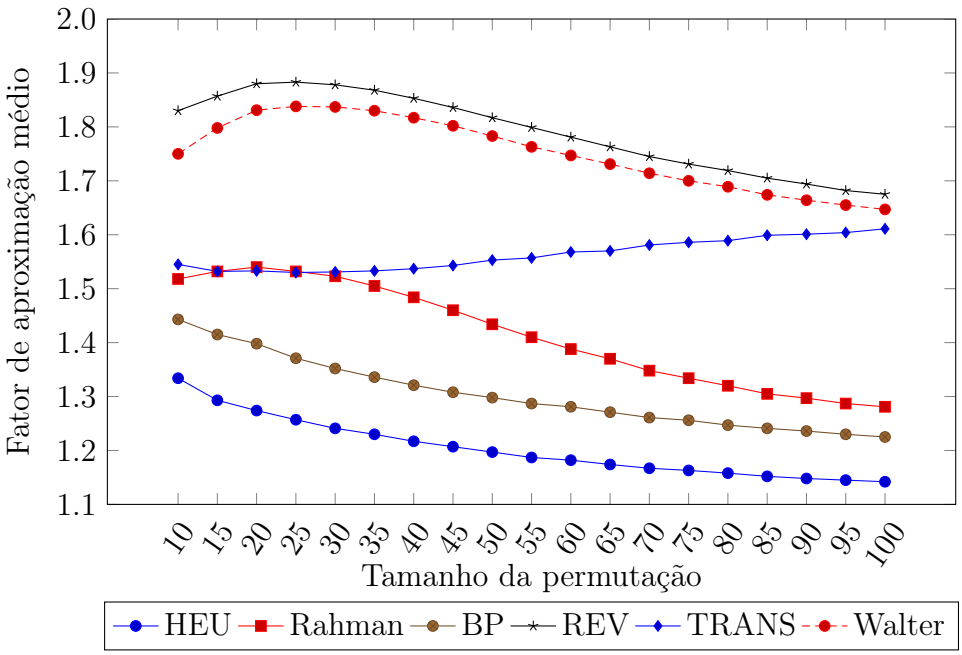


Figura 4.22: Fator de aproximação médio para permutações com sinais.

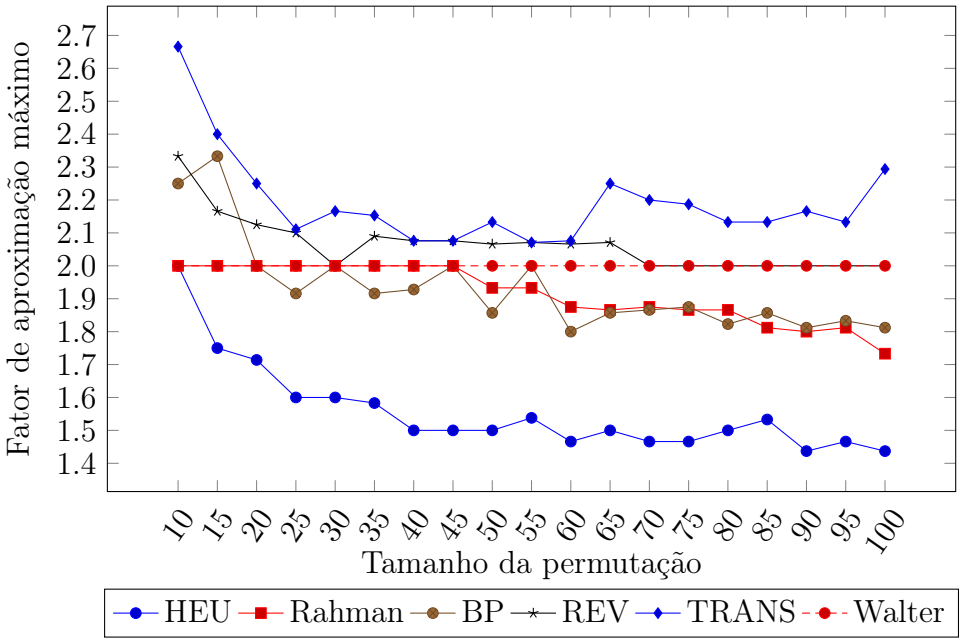


Figura 4.23: Fator de aproximação máximo para permutações com sinais.

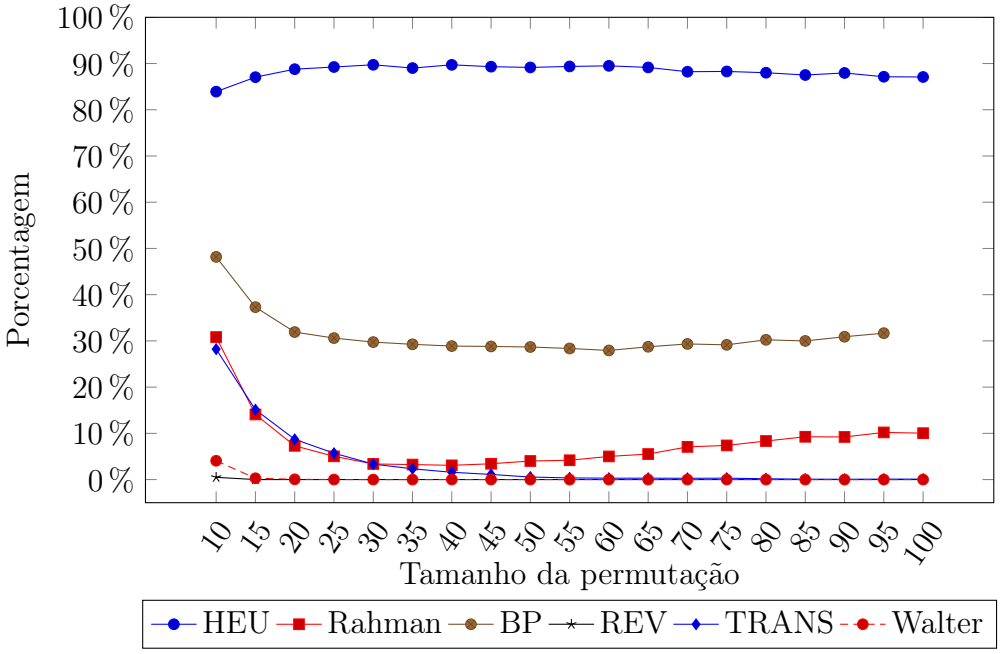


Figura 4.24: Porcentagem de permutações com sinais com menor distância dentre os algoritmos analisados.

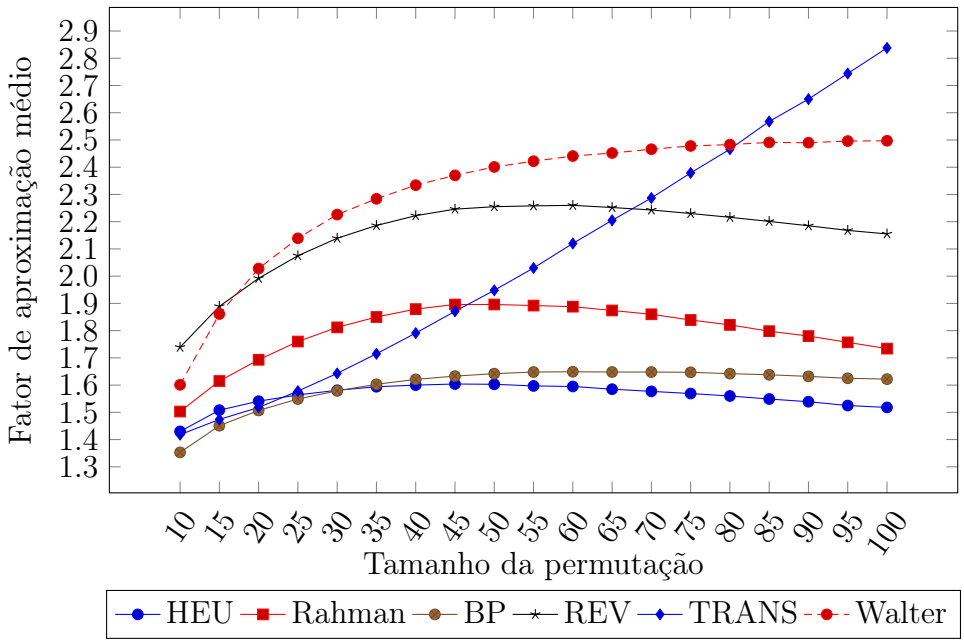


Figura 4.25: Fator de aproximação médio para permutações sem sinais.

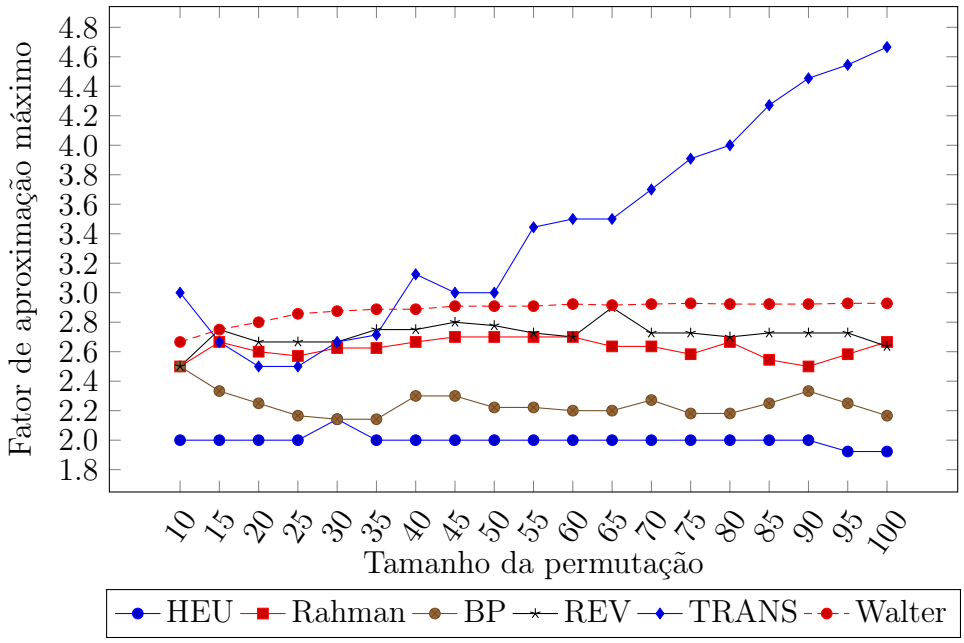


Figura 4.26: Fator de aproximação máximo para permutações sem sinais.

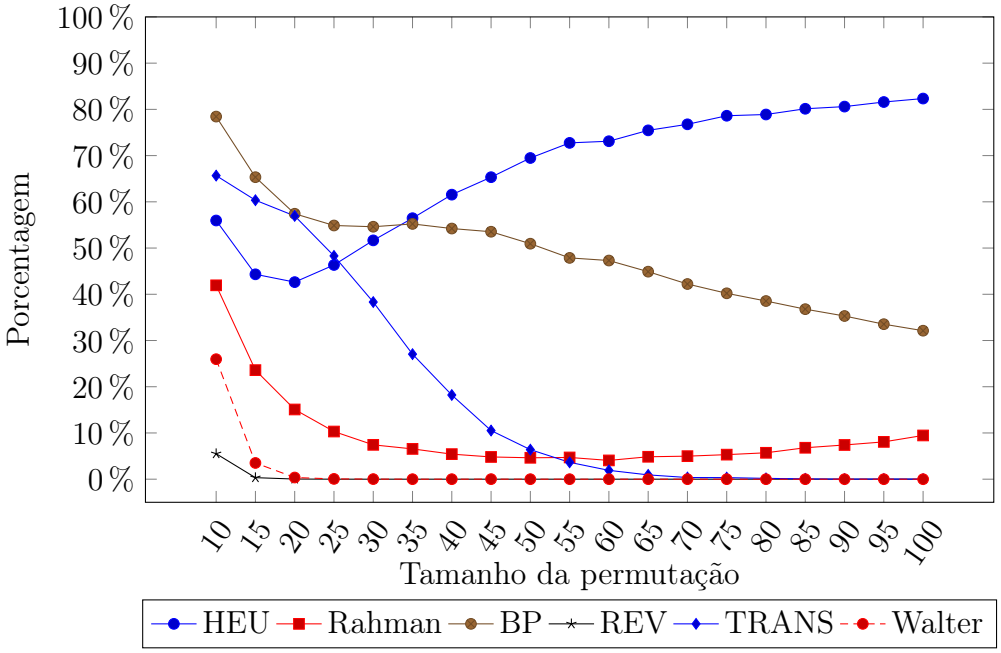


Figura 4.27: Porcentagem de permutações sem sinais com menor distância dentre os algoritmos analisados.

Capítulo 5

Conclusões

Este trabalho apresentou um estudo do Problema da Ordenação de Permutações por Reversões e Transposições. Foram criadas diversas heurísticas, apresentadas no Capítulo 3, de forma a verificar o comportamento de métricas existentes na literatura com o problema proposto. Essas heurísticas foram posteriormente testadas com uma grande quantidade de permutações com sinais e permutações sem sinais.

Além disso, apresentamos uma heurística utilizando o grafo de ciclos, chamada de Heurística de Grafo de Ciclos, em conjunto com a métrica breakpoints, que foi a heurística que obteve os melhores resultados na análise do Capítulo 3.

Os resultados obtidos pela Heurística de Grafo de Ciclos foram comparados com diversos algoritmos existentes na literatura.

Para permutações com sinais, a heurística retornou em média as menores distâncias e os menores fatores de aproximação médios para todos os tamanhos de permutações testadas. Além disso, o maior fator de aproximação da heurística ocorreu em permutações de tamanho 10, cujo fator foi de 2.0, sendo este o fator teórico garantido pela heurística para permutações com sinais. Para permutações maiores, o fator de aproximação ficou sempre abaixo de 1.8. Por fim, dentre todos os algoritmos testados, a heurística retornou em média pelo menos 80% dos melhores resultados.

Para permutações sem sinais, a heurística retornou em média as menores distâncias e os melhores fatores de aproximação médios para as permutações testadas com tamanho maior que 30. Além disso, o maior fator de aproximação da heurística ocorreu em permutações de tamanho 30, cujo fator foi de 2.142. Este fator ficou abaixo do fator teórico da heurística para ordenar permutações sem sinais, que é de 2.8386. Para os demais tamanhos de permutações, os maiores fatores de aproximação ficaram sempre iguais a 2.0. Por fim, dentre todos os algoritmos testados, a heurística retornou em média pelo menos 60% dos melhores resultados para as permutações testadas com tamanho maior ou igual a 40.

Uma versão preliminar da heurística descrita no Capítulo 4 foi apresentada em Setembro de 2014 na *5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB)*, na cidade de Newport Beach, California, e publicada no anais do evento [16]. A versão preliminar não considerava a etapa de remoção de breakpoints. Além disso, na versão preliminar, os bancos de configurações foram gerados através de regras considerando open gates, e não caminhos hamiltonianos.

Como trabalhos futuros, pode-se fazer uma investigação sobre as configurações que não

conseguiram uma sequência válida com aproximação 1.8, e verificar formas não proibitivas computacionalmente de diminuir estes fatores. Além disso, pode ser feita uma análise do limitante inferior da distância $d_{\overline{r}t}(\pi)$, uma vez que se $G(\pi)$ possui pelo menos uma das configurações de ciclos utilizadas nos bancos de configurações, sabemos que serão necessárias pelo menos $d_{\overline{r}t}(\pi) + 1$ operações para ordenar π.

O Problema de Ordenação de Permutações por Reversões e Transposições ainda não possui complexidade conhecida para as versões que consideram permutações com sinais e permutações sem sinais. Trabalhos futuros podem também investigar essas complexidades.

Referências Bibliográficas

[1] 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*, 8:483–491, 2001.

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

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

[4] 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.

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

[6] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). In *Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209. Springer Berlin Heidelberg, 1999.

[7] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Journal of Computational Biology*, 172:11–17, 1996.

[8] L. Bulteau, G. Fertin, and I. Rusu. Pancake flipping is hard. In *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin Heidelberg, 2012.

[9] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012.

[10] 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.

[11] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[12] X. Chen. On sorting unsigned permutations by double-cut-and-joins. *Journal of Combinatorial Optimization*, 25(3):339–351, 2013.

[13] D. A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'1998)*, pages 244–252, San Francisco, California, USA, 1998.

[14] D. A. Christie. *Genome rearrangement problems*. PhD thesis, Glasgow University, 1998.

[15] D. S. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Theoretical Computer Science*, 61(2):105–120, 1995.

[16] U. Dias, A. R. Oliveira, and Z. Dias. An improved algorithm for the sorting by reversals and transpositions problem. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB'2014)*, pages 400–409, Newport Beach, California, USA, 2014.

[17] Z. Dias and U. Dias. Sorting by prefix reversals and prefix transpositions. *Discrete Applied Mathematics*, 181:78–89, 2015.

[18] Z. Dias and J. Meidanis. Sorting by prefix transpositions. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, pages 65–76, London, UK, 2002.

[19] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

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

[21] J. Fischer and S. W. Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. In *Proceedings of the 13th Annual European Conference on Algorithms (ESA'2005)*, pages 415–425, Palma de Mallorca, Spain, 2005.

[22] W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27:47–57, 1979.

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

[24] S. A. Guyer, L. S. Heath, and J. P. Vergara. Subsequence and run heuristics for sorting by transpositions. Technical report, Virginia Polytechnic Institute & State University, 1997.

[25] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'1995)*, pages 581–592, Milwaukee, Wisconsin, USA, 1995.

[26] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[27] T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005.

[28] L. S. Heath and J. P. C. Vergara. Sorting by bounded block-moves. *Discrete Applied Mathematics*, 88(1-3):181–206, 1998.

[29] M. H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25(1):67–94, 1997.

[30] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13:180–210, 1995.

[31] G. Lin and T. Jiang. A further improved approximation algorithm for breakpoint graph decomposition. *Journal of Combinatorial Optimization*, 8(2):183–194, 2004.

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

[33] C. N. Lintzmayer and Z. Dias. On the diameter of rearrangement problems. In *Algorithms for Computational Biology*, volume 8542 of *Lecture Notes in Computer Science*, pages 158–170. Springer International Publishing, 2014.

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

[35] A. Rahman, S. Shatabda, and M. Hasan. An approximation algorithm for sorting by reversals and transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.

[36] M. Sharmin, R. Yeasmin, M. Hasan, A. Rahman, and M. S. Rahman. Pancake flipping with two spatulas. *Electronic Notes in Discrete Mathematics*, 36:231–238, 2010.

[37] E. Tannier, A. Bergeron, and M.-F. Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

[38] G. Tesler. Grimm: genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

[39] 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.

[40] 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.