Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Cálculo da Distância de Reversão e Construção de Árvores Filogenéticas usando a Ordem dos Genes

José Luis Soncco Álvarez

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Orientador
Prof. Dr. Mauricio Ayala Rincón

Brasília
2017

Ficha Catalográfica de Teses e Dissertações

Está página existe apenas para indicar onde a ficha catalográfica gerada para dissertações de mestrado e teses de doutorado defendidas na UnB. A Biblioteca Central é responsável pela ficha, mais informações nos sítios:

http://www.bce.unb.br
http://www.bce.unb.br/elaboracao-de-fichas-catalograficas-de-teses-e-dissertacoes

Esta página não deve ser inclusa na versão final do texto.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Cálculo da Distância de Reversão e Construção de Árvores Filogenéticas usando a Ordem dos Genes

José Luis Soncco Álvarez

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Prof. Dr. Mauricio Ayala Rincón (Orientador)
CIC / UnB

Profa. Dra. Telma Woerle Lima Soares
Inst. Informática / UFG

Prof. Dr. Marcelo Macedo Brígido
Inst. Biologia / UnB

Prof. Dr. Guilherme Pimentel Telles
Inst. Computação / Unicamp

Prof. Dr. André Costa Drummond
CIC / UnB

Prof. Dr. George Teodoro (Membro Suplente)
CIC / UnB

Profa. Dra. Célia Ghedini Ralha
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 03 de março de 2017

Dedicatória

Dedico este trabalho a toda minha família: avós, tios, primos, irmãos, e especialmente a minha esposa Rosimery e aos meus pais que sempre me apoiaram para continuar meus estudos fora do meu pais.

Agradecimentos

Agradeço a todas a pessoas que contribuíram direta é indiretamente com a elaboração deste trabalho. Aos meus colegas (e amigos) de laboratorio que me acompanharam e presenciaram o esforço que eu fiz dia a dia durante o doutorado. Aos professores do programa de pós-graduação em informática, por compartilhar seus conhecimentos e sua experiência, sem os quais não poderia ter concluído com sucesso este trabalho. E finalmente, a meu orientador Mauricio Ayala Rincón, quem ajudo muito durante todo o processo de doutorado e soube me guiar para realizar uma bom trabalho.

Resumo

O cálculo de distâncias evolutivas, como as distâncias de reversão e *double cut and join*, entre a ordem dos genes de dois organismos é um problema combinatório complexo. Este cenário pode ficar ainda mais complicado se quisermos construir árvores filogenéticas, visto que a maioria das abordagens da literatura primeiro solucionam o problema da mediana de três genomas, o qual foi demonstrado ser $\mathcal{NP}$-Difícil para vários modelos evolutivos. Neste trabalho propomos vários algoritmos evolutivos para o problema de ordenação de permutações (sem sinal) por reversões, cuja saída é a distância de reversão. Estes algoritmos são baseados em um algoritmo genético simples, sobre o qual foram incorporados varias heurísticas como busca local, busca por oposição, e eliminação de pontos de quebra. Experimentos foram realizados usando diferentes dados (permutações) baseados na ordem dos genes, os quais foram gerados artificialmente (de forma aleatória) e também a partir de dados biológicos. Dentre estes algoritmos os que melhores resultados tem para casos práticos, ou seja, permutações de comprimento até 120, são os chamados AMBO e AMBO-Híbrido. Estes resultados foram validados usando testes estatísticos como Friedman e Holm. Adicionalmente, foi implementado um software para construir árvores filogenéticas chamado de `HELPHY`, que toma como entrada dados baseados na ordem dos genes (permutações com sinal). Primeiro foi proposto um algoritmo guloso para o problema da pequena filogenia, cujo objetivo é calcular o custo de uma determinada árvore. Logo, para o problema da grande filogenia foi proposta uma abordagem baseada em busca em vizinhança variável, cujo objetivo é explorar o espaço de soluções de estruturas de árvores. Experimentos mostraram que `HELPHY` conseguiu melhorar o tempo de execução para encontrar árvores com bons escores (distância de reversão) para o dataset *Campanulaceae*; além disso, uma nova árvore tendo o melhor escore (distância *double cut and join*) na literatura foi encontrado para o dataset *Hemiascomycetes*.

Palavras-chave: Algoritmos Genéticos, Distância de Reversão, Combinatória de Permutações, Árvores Filogenéticas

Abstract

Calculating evolutionary distances, such as the reversal distance or the double cut and join distance, between the gene orders of two organisms is a complex combinatory problem. This scenario can be even more complicated if we want to build phylogenetic trees, since most of the approaches in the literature first solves the median problem for three genomes, which was shown to be $\mathcal{NP}$-Hard for various evolutionary models. In this work, we are proposing several evolutionary algorithms for the problem of sorting (unsigned) permutations by reversals, whose output is the reversal distance. These algorithms are based on a simple genetic algorithm, on which were embedded different heuristics such as local search, opposition-based learning, and elimination of breakpoints. Experiments were performed using different types of data (permutations) based on gene orders which were generated artificially (in a random way) and also from biological data. From these algorithms, the ones with the best results for practical cases, that is, permutations of length up to 120, are called as AMBO and AMBO-Híbrido. These results were validated by applying the Friedman and Holm statistical tests. Moreover, a software called `HELPHY` for building phylogenetic trees was implemented, which takes as input data based on gene order (signed permutations). First, an greedy algorithm was proposed for the small phylogeny problem, whose aim is to calculate the cost (score) of a given tree structure. Then, an approach based on variable neighborhood search was proposed for the large phylogeny problem, whose aim is to explore the search space of tree structures. Results of the experiments showed that `HELPHY` improved the execution time for finding good scores (reversal distance) for the dataset *Campanulaceae*; besides, a new tree structure with the best score (double cut and join distance) in the literature was found for the dataset *Hemiascomycetes*.

Keywords: Genetic Algorithms, Reversal Distance, Combinatorics of Permutations, Phylogenetic Trees

Sumário

Lista de Abreviaturas e Siglas

AG Algoritmos Genéticos.

AM Algoritmos Meméticos.

AMBO Algoritmo Memético com Busca por Oposição.

CM Computação Memética.

DCJ *Double Cut and Join.*

EMC problema de Emparelhamento com número Máximo de Ciclos.

EPMC problema de Emparelhamento de Permutações com número Máximo de Ciclos.

HELPHY ***He****uristics for the* ***L****arge and Small* ***Phy****logeny Problem.*

OBL *Opposition Based Learning.*

OPCSR Ordenação de Permutações com Sinal por Reversões.

OPSSR Ordenação de Permutações sem Sinal por Reversões.

PDR Problema da Distância de Reversão.

Capítulo 1

Introdução

O problema de ordenação de permutações por reversões é de grande importância no campo da bioinformática. A distância de reversão é usada para determinar a distância evolutiva entre dois organismos e assim também pode ser usada para construir árvores filogenéticas de vários organismos. A complexidade deste problema pode variar dependendo se os genes foram abstraídos considerando a sua orientação, gerando permutações com sinal ou não. Se a orientação é considerada o problema pertence a classe $\mathcal{P}$; por outro lado, se a orientação dos genes é descartada se consideram somente a ordem dos genes, gerando permutações sem sinal, e neste caso o problema é $\mathcal{NP}$-Difícil.

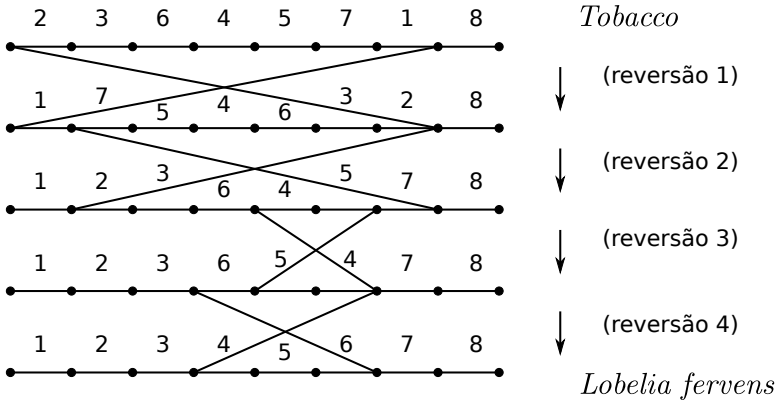

Figura 1.1: Sequência de reversões para transformar a permutação do *Tobacco* na permutação do *Lobelia fervens* (Tomada de [9]).

O problema de ordenação por reversões (usando permutações sem sinal) é um problema de otimização, onde o objetivo é minimizar o número de reversões para transformar um organismo em outro. Os genomas dos organismos são representados como uma sequência de números naturais diferentes, onde cada número representa a ordem de um gene dentro do genoma. Uma reversão é uma operação que inverte uma subsequência de genes contíguos. Por exemplo, a Figura 1.1 mostra uma sequência de reversões que transforma a

permutação do *Tobacco* na permutação do *Lobelia fervens* (exemplo clássico tomado de [9]). Os genes do cloroplasto do *Lobelia fervens* são representados como uma sequência incremental de naturais, e os genes do cloroplasto do *Tobacco* mantêm a mesma representação de números naturais na respectiva ordem neste organismo. A sequência mostrada representa uma solução ótima com só 4 reversões que é a distância de reversão entre estes dois organismos.

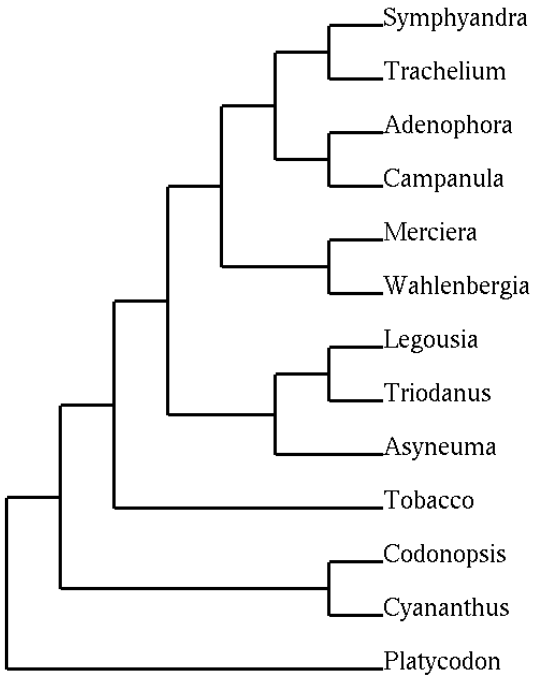

Figura 1.2: Árvore filogenética do dataset *Campanulaceae* encontrada pelo software `MGR` [19].

Tradicionalmente a reconstrução de árvores filogenéticas é baseada na análise de genes individuais. Por outro lado, o rearranjo de genomas é baseado na análise das ordens dos genes, normalmente representadas como permutações com ou sem sinal. Nesse contexto, a reconstrução de árvores filogenéticas está relacionada ao problema de rearranjo de múltiplos genomas, cujo objetivo é construir uma árvore filogenética que minimize o custo total da árvore com respeito a uma métrica (e.g. distância de reversão). Recentemente Yancopoulos [91] propôs uma métrica universal, a operação *Double Cut and Join* (DCJ), que unifica outras operações como reversões, translocações, e intercâmbio de blocos (fissões e fusões). Esta operação corta um cromossomo em duas partes e depois junta os 4 extremos de uma forma diferente à original. A operação pode ser estendida para genomas com múltiplos cromossomos, os quais podem ser lineares e circulares [12]. A Figura 1.2 mostra a árvore filogenética do dataset *Campanulaceae*, a qual contém genomas circulares, esta árvore foi encontrada pelo software `MGR` [19] com um custo total (escore) de 65

reversões. Depois, Kovac et al. [51] usando a mesma estrutura desta árvore encontrou um escore de 59 operações DCJ.

1.1 Motivação

Hannenhalli e Pezner [44] propuseram o primeiro algoritmo exato de tempo polinomial para resolver o problema de ordenação por reversões de permutações (com sinal). Logo, baseados nesses resultados Bader et al. [8] propuseram um algoritmo de tempo linear para calcular a distância de reversão. Depois, como uma extensão natural dessa pesquisa todos esses resultados foram incluídos dentro de um contexto maior que é o da reconstrução de árvores filogenéticas. O cálculo da distância de reversão foi incluído no software de filogenia `GRAPPA` que só usava a distância de pontos de quebra, estendendo dessa maneira a análise filogenética usando uma distância mais significativa desde o ponto de vista evolutivo.

A versão do problema de ordenação por reversões usando permutações sem sinal foi mostrada que é $\mathcal{NP}$-Difícil [21], portanto esta versão do problema é também interessante desde o ponto de vista computacional e combinatório. Assim, foram propostos vários algoritmos aproximados até atingir o raio teórico de 1.375 [14], sendo o algoritmo com raio 1.5 [26] o mais implementado. Também, foram propostos algoritmos evolutivos como o algoritmo genético proposto por Auyeung e Abraham [6].

Depois foram propostas melhorias sobre a abordagem de Auyeung e Abraham como a inclusão de heurísticas usadas por algoritmos de aproximação [76]. Até este ponto foram usados dados gerados artificialmente, logo precisavam ser geradas permutações baseadas em dados biológicos. Também precisavam ser propostas abordagens paralelas e aprimorar as sequenciais usando outras heurísticas.

Com respeito a reconstrução de árvores filogenéticas, este problema pode ser divido em dois problemas: um interno e um externo. O problema interno é conhecido como problema da pequena filogenia que consiste em rotular nós internos com genomas ancestrais. A maioria das abordagens (e.g. `GRAPPA`) para solucionar este problema primeiro solucionavam o problema da mediana de 3 genomas, este problema foi demostrado ser $\mathcal{NP}$-Difícil para diversos modelos evolutivos (reversões, operações DCJ). O problema externo é conhecido como o problema da grande filogenia que consiste em explorar o espaço de soluções de estruturas de árvores, isto foi feito ou explorando exaustivamente o espaço de soluções (e.g. `BPAnalysis`) ou tomando amostras do espaço de soluções (e.g. `GRAPPA`).

Até onde sabemos não foram utilizadas heurísticas para explorar o espaço de soluções quando são usados dados baseados na ordem dos genes. No entanto, no contexto de entradas que são sequências de caracteres (e.g. sequências de DNA) a literatura é extensa

e muitas destas heurísticas podem ser reaproveitadas. Portanto, também precisavam ser propostas abordagens heurísticas para explorar o espaço de soluções de estruturas de árvores no caso de dados baseados na ordem dos genes, e usando como métricas (para avaliar o custo das árvores) as distâncias de reversão e DCJ.

1.2 Resultados Principais

Propuseram-se diversos algoritmos evolutivos para o problema de ordenação por reversões usando dados baseados na ordem dos genes (permutações sem sinal). Desenvolveu-se um software baseado em heurísticas para a reconstrução de árvores filogenéticas usando como entrada dados baseados na ordem dos genes (permutações com sinal). Para avaliar o custo das árvores foram considerados como métricas a distância de reversão e a distância DCJ.

1.3 Resultados Específicos

Foram obtidos os seguintes resultados específicos:

- Desenvolvimento de novos algoritmos baseados no algoritmo genético (proposto em [76]) para problema de ordenação por reversões, o qual é aprimorado usando outras heurísticas como busca local, e busca por oposição (Referências: [77]).
- Desenvolvimento de algoritmos paralelos baseados no algoritmo genético (proposto em [76]) para o problema de ordenação por reversões (Referências: [79, 74]).
- Realizaram-se experimentos utilizando como entrada dados baseados na ordem dos genes, representados neste caso como permutações sem sinal as quais foram geradas de diversas formas: de forma aleatória, ou baseadas em dados biológicos (Referências: [79, 74, 77]).
- Realizou-se uma comparação estatística dos resultados dos experimentos para determinar qual é o melhor algoritmo e se os resultados deste algoritmo tem uma diferença estatisticamente significativa com respeito aos outros algoritmos.
- Desenvolvimento de um algoritmo heurístico, baseado na abordagem de Kovac et al. [51], para o problema da pequena filogenia tomando como dados de entrada um conjunto de genomas baseados na ordem dos genes, e uma estrutura de uma árvore (Referências: [78]).
- Desenvolvimento de uma abordagem baseada em busca em vizinhança variável para o problema da grande filogenia tomando como dados de entrada um conjunto de genomas baseados na ordem dos genes (Referências: [78]).

  - Para avaliar o custo (aptidão) das aŕvores foi usado o algoritmo anterior para o problema da pequena filogenia.

- Realizaram-se experimentos usando datasets importantes da literatura e se compararam os resultados com outras abordagens da literatura para o problema da grande e pequena filogenia.

1.4 Estrutura do Trabalho

Este trabalho está organizado em capítulos. No Capítulo 2, são apresentadas definições e conceitos referentes ao cálculo da distância de reversão, também é apresentado uma revisão da literatura com respeito ao cálculo desta distância.

No Capítulo 3, são apresentadas definições e conceitos referentes ao cálculo da distância *double cut and join*, e a construção de árvores filogenéticas, também é apresentada uma revisão da literatura com respeito a construção de árvores filogenéticas.

No Capítulo 4, são apresentados os novos algoritmos evolutivos propostos para o problema do cálculo da distância de reversão, também são apresentados os resultados dos experimentos, bem como uma comparação estatística destes resultados usando os testes de Friedman e Holm.

No Capítulo 5, são apresentados as abordagens heurísticas para lidar com o problema da pequena e grande filogenia, também são apresentados resultados dos experimentos que consistem em novas árvores filogenéticas para os datasets *Campanulaceae* e *Hemiascomycetes*.

Finalmente, no Capítulo 6 são apresentados as conclusões e trabalhos futuros. É importante salientar que o código fonte dos algoritmos apresentados nesta tese estão no site `http://genoma.cic.unb.br`.

Capítulo 2

Distância de Reversão

2.1 Permutações, Reversões, Distância de Reversão

A seguir serão mostradas algumas definições as quais foram tomadas principalmente das seguintes referências [9], [50], [27], e [45].

Definição 1. *Genes diferentes em um organismo são representadas por números naturais. Assim, a ordem dos genes de um organismo pode ser interpretada, em notação de string, como uma* **permutação sem sinal** $\pi = \pi_1, \pi_2, \ldots, \pi_n$*, que é uma bijeção do conjunto* $\{1, 2, \ldots, n\}$ *sobre si mesmo, onde* n *é considerado o comprimento da permutação* π*.*

Exemplo 2.1. Seja $\pi = 5,\ 3,\ 2,\ 6,\ 4,\ 1$ uma permutação sem sinal de comprimento 6, onde esta sequência de números representa a ordem dos genes de um organismo.

Definição 2. *Dada uma permutação sem sinal* π *de comprimento* n*, pode ser gerada uma* **permutação estendida** *adicionando os pivôs* $\pi_0 = 0$ *e* $\pi_n = n + 1$ *ao inicio e final de* π *respectivamente.*

Definição 3. *Uma* **reversão** *é uma permutação especial, denotada como* $\rho^{i..j}$*, que inverte os elementos de* π *nas posições no intervalo* $[i, j]$*, tal que* $0 < i \leq j < n + 1$*.*

$$\rho_k^{i..j} = \begin{cases} k, \ \textit{if } k < i \ \textit{ or } k > j, \\ j - (k - i), \ \textit{if } i \leq k \leq j. \end{cases}$$

Nesta notação, k representa qualquer posição de uma permutação de comprimento n, e quando k está fora do intervalo $[i, j]$ o seu valor fica invariável; caso contrário, o seu valor é modificado a $j - (k - i)$. De acordo com esta definição, os elementos de qualquer permutação π no intervalo de posições $[i, j]$ são **revertidos** dentro deste intervalo como ação de uma reversão, escrito em notação funcional como $\rho^{i..j} \circ \pi$, onde o símbolo $\circ$ denota a composição de funções.

Exemplo 2.2. Seja $\pi = 0, 5, 3, 2, \underline{6, 4, 1}, 7$ a permutação estendida do exemplo anterior. A permutação resultante depois de aplicar uma reversão $\rho^{4..6}$ a π (i.e. $\rho^{4..6} \circ \pi$) é a seguinte $\pi' = 0, 5, 3, 2, \underline{1, 4, 6}, 7$. Os elementos sublinhados da posição 4 a 5 são aqueles onde a reversão foi aplicada.

A permutação identidade, denotada por $\imath$, e a permutação ordenada em ordem crescente, ou seja $\imath_k = k$ tal que $0 \leq k \leq n + 1$.

Definição 4. *A* **distância de reversão** *entre duas permutações sem sinal π e σ é o mínimo número de reversões para transformar π em σ, e o problema de encontrar esta distância é conhecido como* **Problema da Distância de Reversão** *(PDR). Uma vez que este problema é equivalente a transformar $\sigma^{-1} \circ \pi$ em $\imath$, podemos expressar o PDR como o problema de encontrar a distância de reversão entre uma permutação σ e $\imath$, este problema é conhecido como* **Ordenação de Permutações sem Sinal por Reversões** *(OPSSR).*

Exemplo 2.3. Seja $\rho^{4..6}, \rho^{1..4}, \rho^{4..5}$ uma sequência de 3 reversões aplicadas sobre a permutação sem sinal π do exemplo anterior. De fato o comprimento desta sequência é a mínima possível para transformar π na permutação identidade. Assim, a distância de reversão para π é igual a 3.

$$\begin{aligned}
\pi &= 0, 5, 3, 2, \underline{6, 4, 1}, 7 \quad (\text{Aplicar } \rho^{4..6}) \\
\rho^{4..6} \circ \pi &= 0, \underline{5, 3, 2, 1}, 4, 6, 7 \quad (\text{Aplicar } \rho^{1..4}) \\
\rho^{1..4} \circ \rho^{4..6} \circ \pi &= 0, 1, 2, 3, \underline{5, 4}, 6, 7 \quad (\text{Aplicar } \rho^{4..5}) \\
\rho^{4..5} \circ \rho^{1..4} \circ \rho^{4..6} \circ \pi &= 0, 1, 2, 3, 4, 5, 6, 7
\end{aligned}$$

Definição 5. *Deixe $i \sim j$ denotar a propriedade $|i - j| = 1$. Em uma permutação estendida π, dois elementos π_i e π_j são chamados de consecutivos se $i \sim j$, tal que $0 \leq i, j \leq n+1$. Elementos consecutivos em uma permutação π são chamados de* **adjacentes** *se $\pi_i \sim \pi_j$, logo estes mesmos elementos formam um* **ponto de quebra** *se $\pi_i \nsim \pi_j$. Note que, a única permutação que não tem pontos de quebra é a permutação identidade.*

Definição 6. *Seja ρ uma reversão que transforma π em π', e deixe $b(\pi)$ denotar o* **número de pontos de quebra** *de uma permutação sem sinal π. Então, temos que $b(\pi) - b(\pi') \in \{-2, -1,\ 0,\ 1,\ 2\}$, e que as reversões que eliminam r pontos de quebra simultaneamente são chamadas de r-**reversões**, tal que $r \in \{0, 1, 2\}$.*

Definição 7. *Um* **grafo de pontos de quebra** *(grafo de ciclos) $G(\pi)$ da permutação π é um grafo de arestas coloridas derivado das adjacências e pontos de quebra de π o qual tem $n + 2$ vértices, um vértice para cada elemento de π incluindo os pivôs. Dois vértices π_i e π_j são juntados por uma* **aresta cinza** *(representadas como linhas tracejadas) se não são*

consecutivas e $\pi_i \sim \pi_j$*, por outro lado são juntadas por uma* **aresta preta** *(representadas como linhas continuas) se formam um ponto de quebra, quer dizer, são consecutivas mas* $\pi_i \nsim \pi_j$*.*

Podemos verificar facilmente que a única permutação sem nenhuma aresta é a permutação identidade, uma vez que não tem pontos de quebra e todos seus vértices consecutivos são adjacentes.

Exemplo 2.4. Considere novamente a permutação estendida $\pi = 0, 5, 3, 2, 6, 4, 1, 7$, onde a Figura 2.1 mostra o grafo de pontos de quebra $G(\pi)$. Veja que os elementos 5 e 3 formam um ponto de quebra, assim existe uma aresta preta entre eles. Por outro lado, os elementos 5 e 6 têm uma aresta cinza incidente a ambos.

Definição 8. *Um ciclo em* $G(\pi)$ *é chamado de* **ciclo alternado** *se a cor das arestas é alternado entre cinza e preto.*

Ciclos em $G(\pi)$ aparecem de forma natural, já que cada vértice tem exatamente zero, um, ou duas arestas pretas incidentes, e o mesmo número de arestas cinzas incidentes. De fato, para qualquer ponto de quebra formado por um vértice π_i existe uma aresta preta, e uma aresta cinza que vai até um vértice não consecutivo π_j tal que $\pi_j \sim \pi_i$. Assim, se construirmos um caminho entrando por um vértice usando ou uma aresta preta ou uma aresta cinza, deixa a possibilidade de deixar o vértice usando ou uma aresta cinza ou uma aresta preta respectivamente. Como consequência, $G(\pi)$ pode ser decomposto em ciclos alternados de arestas disjuntas. Pode-se verificar que para um grafo $G(\pi)$, o qual foi gerado a partir de uma permutação sem sinal, existem muitas decomposições em ciclos diferentes.

Definição 9. *Deixe* $c(\pi)$ *denotar o* **número máximo de ciclos** *em qualquer decomposição de ciclos de* $G(\pi)$*.*

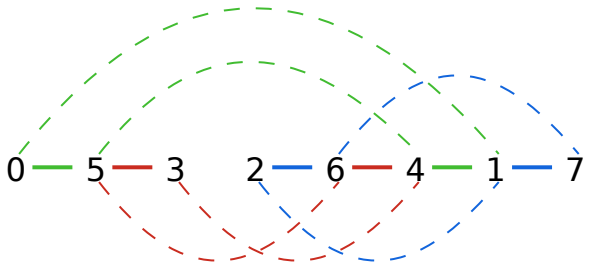

Figura 2.1: Grafo de pontos de quebra $G(\pi)$ e uma decomposição em ciclos. Arestas pretas são representadas como linhas contínuas, e arestas cinzas como linhas tracejadas. Existem 3 ciclos representados pelas cores verde, azul, e vermelho.

Exemplo 2.5. A Figura 2.1 mostra uma decomposição em ciclos do grafo de pontos de quebra da permutação estendida $\pi = 0, 5, 3, 2, 6, 4, 1, 7$. Esta decomposição de ciclos

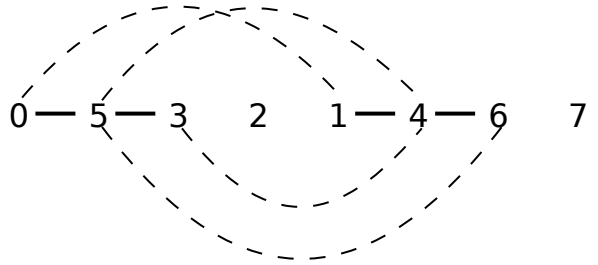

Figura 2.2: Grafo de pontos de quebra resultante depois de aplicar uma 2-reversão ($\rho^{4..6}$) sobre o grafo de pontos de quebra da Figura 2.1.

é formada por três ciclos que são coloridos com verde, azul, e vermelho. A Figura 2.2 mostra a eliminação simultânea de dois pontos de quebra depois de aplicar a 2-reversão $\rho^{4..6}$ sobre a permutação π.

Com respeito a permutações com sinal ($\overrightarrow{\pi}$), os genes são interpretados ou como elementos positivos $+\pi_i$ ou negativos $-\pi_i$. A Figura 2.3 mostra em verde a permutação com sinal $\overrightarrow{\pi} = -2, -5, -1,\ 3, -4, -6$. Assim, para cada permutação sem sinal π podemos construir 2^n permutações com sinal diferentes, isto é feito atribuindo um sinal positivo ou negativo a cada elemento de π. Estas permutações com sinal são chamadas de versões com sinal da permutação π.

Uma reversão $\rho^{i..j}$ aplicada sobre $\overrightarrow{\pi}$, a diferença da versão sem sinal, também muda o sinal dos elementos nas posições dentro do intervalo $[i, j]$. Por exemplo a reversão $\rho^{2..5}$ aplicada sobre

$$\overrightarrow{\pi}$$

gera o seguinte:

$$\begin{aligned} \overrightarrow{\pi} &= -2, \underline{-5, -1,\ 3, -4}, -6 \\ \rho^{2..5} \circ \overrightarrow{\pi} &= -2, 5, 1, -3, 4, -6 \end{aligned}$$

Note que a permutação identidade $\overrightarrow{i}$ é formada só por elementos positivos ordenados em ordem crescente. Neste caso o problema de determinar a distância de reversão entre uma permutação $\overrightarrow{\pi}$ e a permutação identidade é conhecido como **Ordenação de Permutações com Sinal por Reversões** (OPCSR).

A noção de grafo de pontos de quebra pode ser estendida a permutações com sinal transformando uma permutação com sinal em uma permutação sem sinal. Essa transformação é feita mapeando um elemento positivo π_i por $(2\pi_i - 1,\ 2\pi_i)$, e um elemento negativo π_i por $(2\pi_i,\ 2\pi_i - 1)$, e adicionalmente os pivôs inicial e final são dados por $\pi_0 = 0$ e $\pi_{2n+1} = 2n + 1$ respectivamente. Esta transformação leva a permutações cujos grafos de pontos de quebra são de tal forma que cada vértice tem no máximo grau dois, quer dizer, exatamente uma aresta preta e uma aresta cinza (ver Figura. 2.3). Portanto, existe

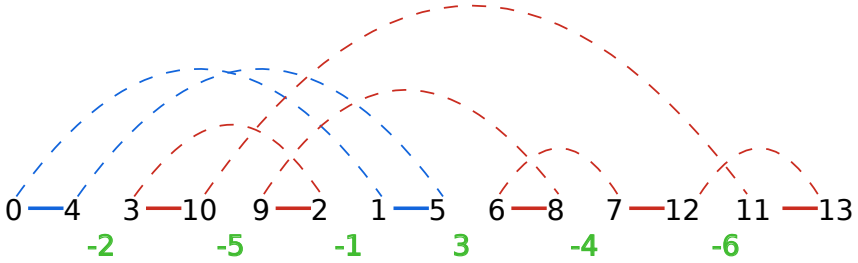

Figura 2.3: Tranformação da permutação com sinal $-2, -5, -1, 3, -4, -6$ na permutação sem sinal $0, 4, 3, 10, 9, 2, 1, 5, 6, 8, 7, 12, 11, 13$.

somente uma única decomposição de ciclos e como consequência o problema OPCSR é mais fácil de ser resolvido que o problema OPSSR.

2.2 Algoritmo Polinomial para o Cálculo da Distância de Reversão (Genes com Sinal)

Suponha que π é a permutação sem sinal que foi obtida a partir da permutação com sinal $\overrightarrow{\pi}$ usando a transformação explicada anteriormente, logo defina $b(\overrightarrow{\pi}) = b(\pi)$ e $c(\overrightarrow{\pi}) = c(\pi)$. Hannenhalli e Pevzner [45] propuseram uma relação simples ($d(\overrightarrow{\pi}) = b(\overrightarrow{\pi}) - c(\overrightarrow{\pi}) + h(\overrightarrow{\pi}) + f(\overrightarrow{\pi})$) para calcular de forma exata a distância de reversão de permutações com sinal, onde $h(\overrightarrow{\pi})$ e $f(\overrightarrow{\pi}) \in \{0, 1\}$ são noções que indicam se uma permutação é difícil de ser ordenada. Esta relação levou ao desenvolvimento de um algoritmo de tempo quadrático ($O(n^2)$) para calcular a distância de reversão de permutações com sinal, e que tem complexidade $O(n^4)$ quando adicionalmente tem que ser calculada a sequência de reversões para ordenar a permutação com sinal.

Logo, Bergeron [10] propôs uma apresentação elementar da teoria de Hannenhalli e Pevzer [45] que age diretamente sobre a permutação com sinal a ser ordenada. Nesta seção será apresentada a abordagem proposta por Bergeron [10].

A ideia para ordenar uma permutação sem sinal $\overrightarrow{\pi}$ usando o número mínimo de reversões é a seguinte: (1) aplicar reversões sobre "pares ordenados", logo no final teremos como resultado uma permutação com todos os elementos positivos; (2) se no passo anterior a permutação não está ordenada, então aplicar reversões sobre "obstáculos"de forma que novos "pares ordenados"sejam criados. Assim podemos ordenar uma permutação aplicando sucessivamente varias vezes os passos (1) e (2).

Agora explicaremos a noções que acabamos de mencionar e apresentaremos os algoritmos de maneria mais formal. Um **par ordenado** (π_i, π_j) de uma permutação com sinal $\overrightarrow{\pi} = 0, \pi_1, \pi_2, \ldots \pi_n\ n+1$, é um par de inteiros consecutivos de sinais opostos tal que $|\pi_i| - |\pi_j| = +1$ ou -1. Este par ordenado indica uma reversão a qual pode ser de dois

formas:

$$\rho^{i..j-1}, \quad \text{se} \quad \pi_i + \pi_j = +1$$
$$\rho^{i+1..j}, \quad \text{se} \quad \pi_i + \pi_j = -1$$

Estas reversões são chamadas de reversões orientadas, as quais depois de serem aplicadas geram inteiros consecutivos. O escore de uma reversão orientada é definido como o número de pares orientados resultantes depois de ser aplicada a reversão.

Exemplo 2.6. A seguinte permutação $\overrightarrow{\pi} = 0,\ 4,\ 2,\ 5,\ 1,\ -3,\ 6$ tem dois pares ordenados $(4, -3)$ e $(2, -3)$. Estes pares ordenados geram as seguintes reversões: $\rho^{1..4}$ e $\rho^{3..5}$ respectivamente.

O escore da reversão $\rho^{1..4}$ é 2 porque depois de ser aplicada sobre $\overrightarrow{\pi}$ temos a seguinte permutação $\rho^{1..4} \circ \overrightarrow{\pi} = 0,\ -1,\ -5,\ -2,\ -4,\ -3,\ 6$ com os seguintes pares ordenados $(0, -1)$ e $(-5, 6)$.

Logo, o escore da reversão $\rho^{3..5}$ é 4 porque depois de ser aplicada sobre $\overrightarrow{\pi}$ temos a seguinte permutação $\rho^{3..5} \circ \overrightarrow{\pi} = 0,\ 4,\ 2,\ 3,\ -1,\ -5,\ 6$ com os seguintes pares ordenados $(0, -1)$, $(2, -1)$, $(4, -5)$, e $(-5, 6)$.

O Algoritmo 1 mostra a estratégia básica para ordenar permutações que tem pares ordenados. Esta estratégia da como resultado uma permutação com todos seus elementos positivos. Em caso de que a permutação não fique ordenada temos que aplicar outra estratégia para lidar com esse problema, que será explicada a continuação.

Algoritmo 1: Estratégia Básica para Ordenar Permutações com Sinal

Entrada: Uma permutação com sinal $\overrightarrow{\pi}$
Saída: Uma permutação com sinal $\overrightarrow{\pi}$ que só tem elementos positivos

```
1 enquanto π⃗ tenha pares orientados faça
2 |  Escolher uma reversão ρ com o máximo escore;
3 |  Aplicar a reversão ρ sobre π⃗;
```

Ordenação de Permutações com Sinal que tem só Elementos Positivos

Seja $\overrightarrow{\pi} = 0,\ \pi_1,\ \pi_2,\ \ldots \pi_n\ n+1$ uma permutação com sinal que tem só elementos positivos, assuma que $\overrightarrow{\pi}$ é reduzida. Uma permutação é **reduzida** se não contém elementos consecutivos ordenados.

Defina um **intervalo emoldurado** na permutação $\overrightarrow{\pi}$ como o intervalo da forma:

$$\mathbf{i}, \pi_{j+1}, \pi_{j+2}, \ldots, \pi_{j+k-1}, \mathbf{i+k}$$

de modo que todos os elementos entre i e $i+k$ pertencem ao intervalo $[i \ldots i+k]$.

Exemplo 2.7. Considere a permutação com sinal $\overrightarrow{\pi} = 0, 3, 5, 4, 6, 2, 1, 7$, note que toda a permutação é um intervalo emoldurado, mas também temos o intervalo emoldurado $3, 5, 4, 6$ que pode ser ordenado como $3, 4, 5, 6$.

Se $\overrightarrow{\pi}$ é uma permutação com sinal reduzida, temos que um **obstáculo** é um intervalo emoldurado que não contém intervalos emoldurados menores. Assim, quando uma permutação tem somente um ou dois obstáculos uma reversão é suficiente para criar suficientes pares ordenados de forma que possamos usar o Algoritmo 1 para ordenar a permutação.

Defina o **corte de um obstáculo** como a reversão sobre o segmento que fica dentro do intervalo i e $i + 1$ de um obstáculo:

$$i, \underline{\pi_{j+1}, \pi_{j+2}, \ldots}, i + 1, \ldots, \pi_{j+k-1}, i + k$$

Logo, defina a **mescla de obstáculos** como a reversão que age sobre os pontos extremos de dois obstáculos:

$$i, \ldots, \underline{i + k, \ldots, i'}, \ldots, i' + k'$$

Exemplo 2.8. Considere a permutação com sinal $\overrightarrow{\pi} = 0, 4, 1, 3, 2, 5$ que tem um único obstáculo $[0, 5]$. A reversão sobre o elemento 4 que fica entre os elementos 0 e 1, corta o obstáculo e gera a seguinte permutação $\overrightarrow{\pi} = 0, -4, 1, 3, 2, 5$ que pode ser ordenada usando o Algoritmo 1.

Exemplo 2.8. Considere a permutação com sinal $\overrightarrow{\pi} = 0, 2, 4, 3, 6, 5, 7, 1, 8$ que tem dois obstáculos: $2, 4, 3, 6, 5, 7$ e $7, 1, 8, 0, 2$. Assim, a reversão que age sobre o elemento 7 mescla os dois obstáculos, e gera a seguinte permutação $\overrightarrow{\pi} = 0, 2, 4, 3, 6, 5, -7, 1, 8$ que pode ser ordenada usando o Algoritmo 1.

Defina um **obstáculo simples** como um obstáculo cujo corte diminui o número de obstáculos. Os obstáculos que não são simples são chamados de **super obstáculos**.

Exemplo 2.9. Considere a permutação com sinal $\overrightarrow{\pi} = 0, 2, 4, 3, 6, 5, 7, 1, 8$ que tem dois obstáculos: $2, 4, 3, 6, 5, 7$ e $7, 1, 8, 0, 2$. Depois de fazer o corte sobre o obstáculo (simples) $2, 4, 3, 6, 5, 7$ e ordenar a permutação resultante, temos a seguinte permutação $0, 2, 3, 4, 5, 6, 7, 1, 8$, a qual depois de ser reduzida da como resultado a permutação $0, 2, 1, 3$ que tem somente um obstáculo.

Exemplo 2.10. Considere a permutação com sinal $\overrightarrow{\pi} = 0, 2, 4, 3, 6, 5, 7, 1, 8, 10, 9, 11$ que tem dois obstáculos $2, 4, 3, 6, 5, 7$ e $8, 10, 9, 11$. Depois de fazer o corte sobre o (super) obstáculo $2, 4, 3, 6, 5, 7$ e ordenar a permutação resultante, temos a seguinte permutação $0, 2, 3, 4, 5, 6, 7, 1, 8, 10, 9, 11$, a qual depois de ser reduzida da como resultado a permutação $0, 2, 1, 3, 5, 4, 6$ que tem ainda dois obstáculos $0, 2, 1, 3$ e $3, 5, 4, 6$.

O Algoritmo 2 mostra a estratégia para lidar com permutações com sinal que só contém elementos positivos. Como mencionado anteriormente, o Algoritmo 1 e o Algoritmo 2 podem ser usados em conjunto para ordenar de forma ótima qualquer permutação com sinal.

Algoritmo 2: Estratégia para Tratar Permutações com Sinal de Elementos Positivos

Entrada: Uma permutação com sinal $\overrightarrow{\pi}$ com só elementos positivos
Saída: Um permutação com sinal $\overrightarrow{\pi}$ com pelo menos um par ordenado

```
1  se uma permutação tem 2k obstáculos, k ≥ 2 então
2      Mesclar dois obstáculos não consecutivos;
3  senão
4      se uma permutação tem 2k + 1 obstáculos, k ≥ 1 então
5          se a permutação tem um obstáculo simples então
6              Cortar o obstáculo;
7          senão
8              se k = 1 então
9                  Mesclar dois obstáculos consecutivos;
10             senão
11                 Mesclar dois obstáculos não consecutivos;
```

2.3 Revisão da Literatura

Caprara [21] demonstrou que OPSSR pertence a classe $\mathcal{NP}$-Difícil, mas antes da complexidade deste problema ser descoberta dois algoritmos foram propostos: um algoritmo de raio de aproximação 2, o qual foi proposto por Kececioglu e Sankoff [50]; e o algoritmo de raio de aproximação 1.75, o qual foi proposto por Bafna e Pevzner [9]. Logo, o raio de aproximação foi melhorado para 1.5 por Christie [27], e depois para 1.375 por Berman et al. [15]. O último algoritmo é de interesse teórico já que a sua implementação é de grande dificuldade, no entanto o algoritmo teórico de raio de aproximação 1.5 tem sido amplamente implementado como mecanismo de controle para comparar a qualidade das soluciones fornecidas por algoritmos heurísticos. De fato, Auyeung e Abraham [5] implementaram este algoritmo, como foi proposto por Christie (chamada de Implementação 1.5-Au), cujos resultados foram também usados por Ghaffarizadeh [43] para fazer comparações. Logo depois, inconsistências teóricas foram encontradas neste algoritmo, as quais foram relatadas e corrigidas em [75] levando a uma nova implementação (chamada de Implementação 1.5-Corrigida), a qual foi usada em [76], e [79].

Para o caso de OPCSR, Hannenhalli e Pevzner [45] mostraram que o problema pertence a classe $\mathcal{P}$, e propuseram um algoritmo de tempo polinomial ($O(n^4)$) para encontrar a sequência de ordenação de reversões, o qual pode rodar em tempo $O(n^2)$ quando se requer somente o cálculo da distância de reversão. Posteriormente, mais melhoras foram propostas. Com respeito ao cálculo da sequência de ordenação, Berman e Hannenhalli [13] propuseram um algoritmo de tempo $O(n^2\ \alpha(n))$, onde $\alpha(n)$ é a inversa da função de Ackermann; logo, Kaplan et al. [49] melhoraram esta complexidade a $O(n^2)$. Logo depois deste último trabalho, Bergeron [10] propôs uma presentação simplificada do método de Hannenhalli e Pevzner, mas com uma complexidade de $O(n^3)$. Mais tarde, melhores algoritmos foram propostos com complexidade de tempo $O(n^{3/2}\sqrt{\log(n)})$ por Tannier et al. [82], e complexidade de tempo $O(n\ \log(n) + kn)$ por Swenson et al. [81], onde k é o número total de reversões "inseguras"realizadas por este algoritmo. Uma permutação com todos seus elementos positivos indica a existência de pelo menos um "componente ruim", e uma reversão é insegura se cria um componente ruim. Neste último algoritmo por cada reversão insegura é realizada uma sequência de correções. Com respeito ao cálculo da distância de reversão apenas, Berman e Hannenhalli [13] melhoraram a complexidade para $O(n\ \alpha(n))$, e finalmente Bader et al. [7] propuseram um algoritmo de tempo linear.

Adicionalmente, para o caso de OPSSR algoritmos evolutivos foram propostos. O primeiro algoritmo genético para o problema de OPSSR foi proposto por Auyeung e Abraham [5], chamado de AG-Au, no qual o espaço de busca é formado por 2^n permutações com sinal que são geradas a partir de uma permutação π sem sinal de tamanho n, atribuindo um sinal positivo ou negativo a cada elemento de π. Este método usa uma população de tamanho n^2 e para o cálculo da aptidão é usado o algoritmo de Kaplan et al. para OPCSR levando a uma complexidade global de $O(n^5)$. Logo, foi proposto um algoritmo evolutivo por Ghaffarizadeh et al. [43] o qual usa uma população de tamanho $n\ \log(n)$, e com uma complexidade global de $O(n^4\ \log^2(n))$. Estes dois algoritmos foram comparados com o algoritmos de raio de aproximação 1.5 (Implementação 1.5-Au) superando seus resultados. Também, o algoritmo de Ghaffarizade et al. relatou ter melhores resultados que o algoritmo AG-Au para permutações até tamanho 110.

Depois, Soncco-Álvarez e Ayala-Rincón propuseram [75] uma abordagem genética, na qual o espaço de busca é formado por reversões que eliminam zero, um, ou dois pontos de quebra. Neste algoritmo a população inicial (de tamanho $n\ \log(n)$) é formada por indivíduos de tamanho zero, e em cada geração os seus comprimentos são incrementados adicionando novas reversões até que uma solução válida é encontrada, a qual ordena a permutação de entrada. A complexidade deste algoritmo é $O(n^3\ \log(n))$, este é o primeiro algoritmo a ser comparado com a nova implementação do algoritmo de raio de aproximação 1.5 (Implementação 1.5-Corrigida), a qual mostrou ter melhores resultados que a

Implementação 1.5-Au, e embora os resultados do algoritmo de Soncco-Álvarez e Ayala-Rincón não melhorem os resultados da Implementação 1.5-Corrigida, tem os resultados mais próximos que outras anteriores abordagens.

Logo depois, Soncco-Álvarez e Ayala-Rincón [76] implementaram uma nova versão do algoritmo genético proposto por Auyeung e Abraham, chamado de AG-SA. Este novo algoritmo tem tamanho de população igual a $n \log(n)$ e para o cálculo da aptidão é usado o algoritmo de tempo linear proposto por Bader et al., levando a uma complexidade global de $O(n^3 \log(n))$. O AG-SA foi o primeiro algoritmo que melhorou os resultados computados pela Implementação 1.5-Corrigida. No mesmo trabalho [76] uma versão melhorada do AG-SA foi proposta chamada de AG-Híbrido, este algoritmo é baseado em AG-SA integrando uma fase de pré-processamento que foi inspirada na heurística de eliminação de 2-reversões usada por muitos algoritmos de aproximação, como em [9] e [27]. Estas 2-reversões eliminam dois pontos de quebra simultaneamente no grafo de pontos de quebra da permutação (sem sinal) inicial até que não é mais possível aplicar uma 2-reversão. Experimentos mostraram que os resultados do AG-Híbrido melhoraram os resultados produzidos pelo AG-SA.

Capítulo 3

Árvores Filogenéticas e a Distância DCJ

3.1 Genes, Genoma, Distância DCJ

As definições apresentadas aqui foram tomadas principalmente do trabalho de Bergeron et al. [12].

Definição 10. *Seja g um* **gene** *que representa uma sequência orientada do DNA. A* **cauda** *de g é denotada por g_t, e sua* **cabeça** *é denotada por g_h; as quais são chamadas de* **extremidades** *do gene.*

Definição 11. *Sejam a e b dois genes consecutivos, uma* **adjacência** *entre a e b pode ser denotada de quatro formas diferentes: $\{a_h, b_t\}, \{a_h, b_h\}, \{a_t, b_t\}, \{a_t, b_h\}$. Uma extremidade que não é adjacente a nenhum outro gene é chamada de* **telômero** *e é denotado por $\{a_t\}$ ou $\{a_h\}$.*

Definição 12. *Um* **genoma** *é um conjunto de adjacências e telômeros, tal que a cabeça ou a cauda de qualquer gene aparece exatamente em uma adjacência ou telômero.*

Definição 13. *Dado um genoma se podem reconstruir os correspondentes cromossomos, isto é o* **grafo do genoma**, *denotando cada adjacência ou telômero por um vértice e depois juntando a cabeça e a cauda de cada gene com uma aresta.*

Exemplo 1. Seja

$$A = \{\{d_h, e_h\}, \{e_t, d_t\}, \{a_t\}, \{a_h, b_t\}, \{b_h, c_h\}, \{c_t, f_h\}, \{f_t\}, \{g_h, h_t\}, \{h_h, g_t\}\}$$

um genoma com 8 genes $\{a, b, c, d, e, f, g, h\}$. O correspondente grafo do genoma é mostrado na Figura 3.1, o qual contém dois cromossomos circulares e um cromossomo linear.

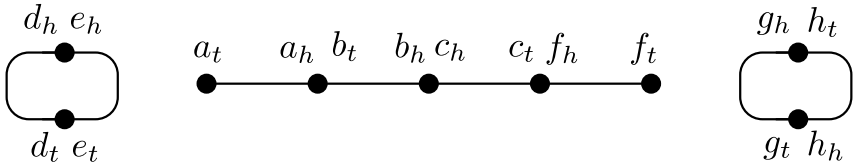

Figura 3.1: Exemplo do grafo de um genoma

Definição 14. *Uma operação* **Double Cut and Join** *(DCJ) age sobre dois vértices u e v de um grafo do genoma na forma seguinte:*

1. *Se $u = \{p, q\}$ e $v = \{r, s\}$ são vértices internos, eles são substituídos pelos vértices $\{p, r\}$ e $\{s, q\}$ ou pelos vértices $\{p, s\}$ e $\{q, r\}$ respectivamente.*

2. *Se $u = \{p, q\}$ é interno e $v = \{r\}$ é externo, eles são substituídos pelos vértices $\{p, r\}$ e $\{q\}$ ou pelos vértices $\{q, r\}$ e $\{p\}$ respectivamente.*

3. *Se $u = \{q\}$ e $v = \{r\}$ são externos, eles são substituídos por $\{q, r\}$ ou vice-versa.*

Exemplo 2. Todos os casos possíveis da definição da operação DCJ são mostrados nas Figuras 3.2, 3.3, e 3.4.

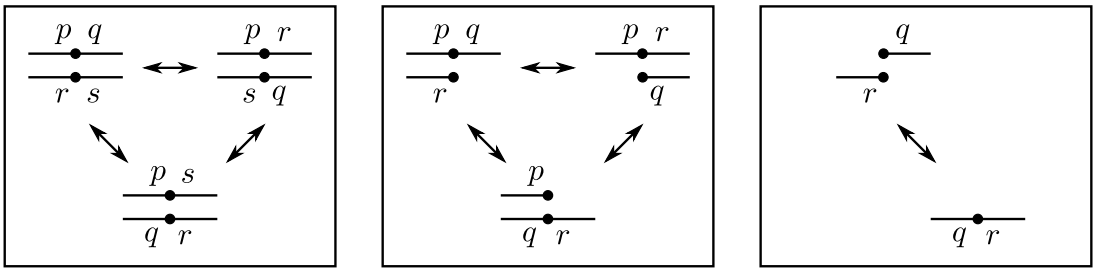

Figura 3.2: A operação DCJ aplicada sobre um ou dois caminhos, produz translocações, fusões, e fissões.

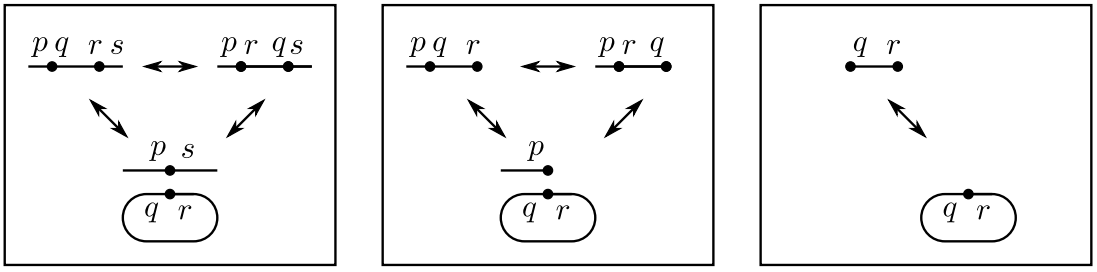

Figura 3.3: A operação DCJ aplicada sobre um único caminho ou sobre um caminho e um ciclo, produz reversões, excisões, integrações, novos ciclos, e novos caminhos.

Definição 15. *Dados dois genomas A e B os quais estão definidos sobre o mesmo conjunto de genes,* **o problema da distância DCJ** *consiste em encontrar a sequência mínima de*

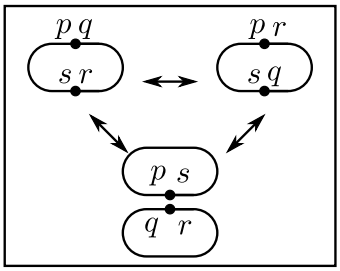

Figura 3.4: A operação DCJ aplicada sobre um único ciclo ou sobre dois ciclos, produz reversões, fusões e fissões de ciclos.

operações DCJ para transformar A em B. O comprimento desta sequência é chamado de **distância DCJ** *entre A e B, e é denotada por* $d_{DCJ}(A, B)$.

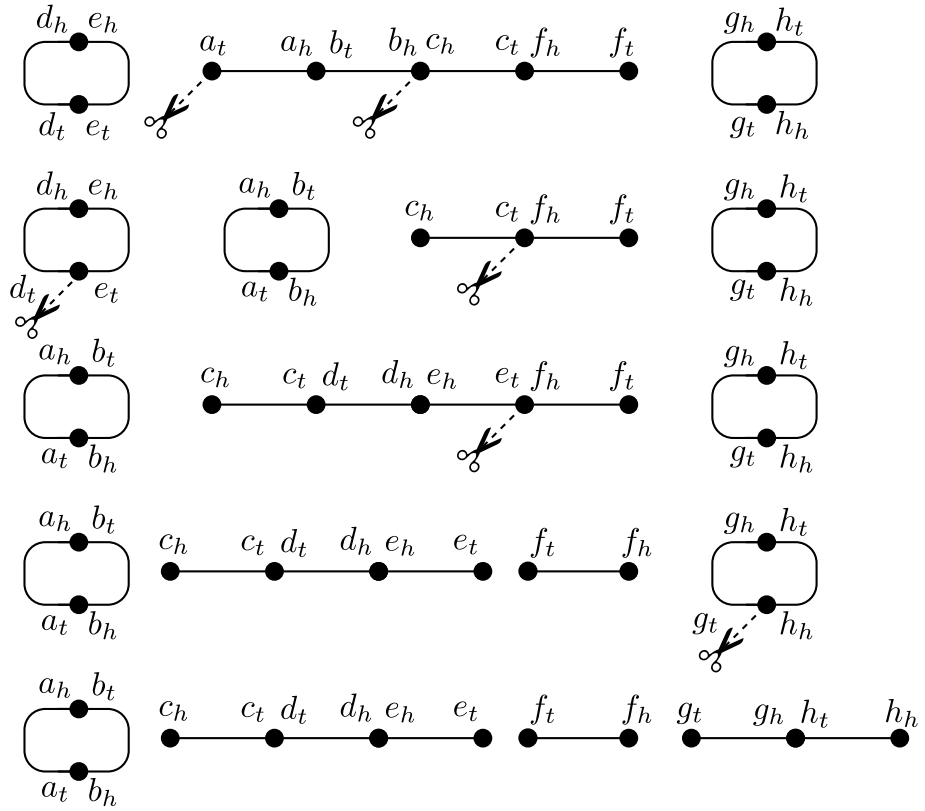

Figura 3.5: Sequência de operações DCJ para transformar o genoma A em B

Exemplo 3. Considere os seguintes genomas que são definidos sobre o conjunto de genes $\{a, b, c, d, e, f, g, h\}$:

$$A = \{\{d_h, e_h\}, \{e_t, d_t\}, \{a_t\}, \{a_h, b_t\}, \{b_h, c_h\}, \{c_t, f_h\}, \{f_t\}, \{g_h, h_t\}, \{h_h, g_t\}\}$$

$$B = \{\{a_h, b_t\}, \{b_h, a_t\}, \{c_h\}, \{c_t, d_t\}, \{d_h, e_h\}, \{e_t\}, \{f_h\}, \{f_t\}, \{g_t\}, \{g_h, h_t\}, \{h_h\}\}$$

Uma sequência de operações DCJ (4 passos) para transformar o genoma A em B é mostrada na Figura 3.5. A distância DCJ entre A e B é $d_{DCJ}(A, B) = 4$ porque o mínimo número de operações DCJ para transformar A em B é 4.

Definição 16. *O* **grafo de adjacência** $G_{Adj}(A, B)$ *é um grafo cujo conjunto de vértices são as adjacências e os telômeros de* A *e* B*. Para cada* $u \in A$ *e* $v \in B$ *existem* $|u \cap v|$ *arestas entre* u *e* v*.*

Exemplo 4. O grafo de adjacência dos genomas

$$A = \{\{d_h, e_h\}, \{e_t, d_t\}, \{a_t\}, \{a_h, b_t\}, \{b_h, c_h\}, \{c_t, f_h\}, \{f_t\}, \{g_h, h_t\}, \{h_h, g_t\}\}$$

$$B = \{\{a_h, b_t\}, \{b_h, a_t\}, \{c_h\}, \{c_t, d_t\}, \{d_h, e_h\}, \{e_t\}, \{f_h\}, \{f_t\}, \{g_t\}, \{g_h, h_t\}, \{h_h\}\}$$

é mostrado na Figura 3.6.

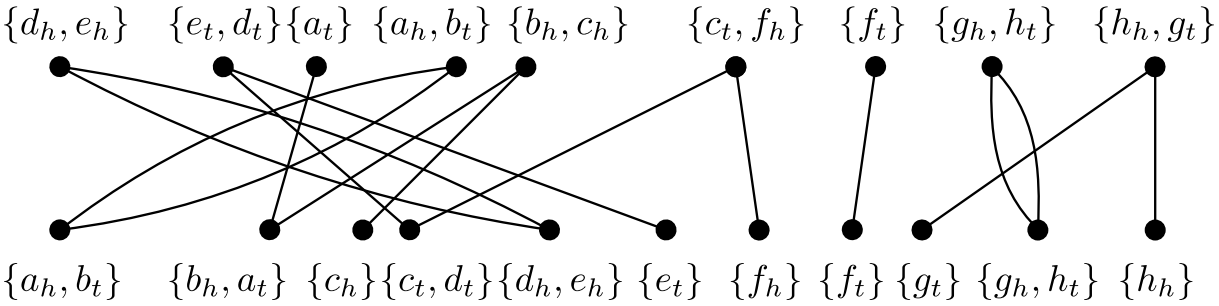

Figura 3.6: Grafo de adjacência dos genomas A e B

Bergeron et al. [12] propuseram um algoritmo ótimo de tempo linear para calcular a distância DCJ usando o grafo de adjacência de dois genomas.

Algoritmo 3: Algoritmo para calcular a distância DCJ

Entrada: Grafo de adjacência dos genomas A e B
Saída: Distância DCJ entre os genomas A e B

```
1 para cada adjacência {p, q} do genoma B faça
2     seja u o elemento do genoma A que contém p;
3     seja v o elemento do genoma A que contém q;
4     se u ≠ v então
5         substituir u e v em A por {p, q} e (u\{p}) ∪ (v\{q});
6 para cada telômero {p} do genoma B faça
7     seja u o elemento do genoma A que contém p;
8     se u é uma adjacência então
9         substituir u em A por {p} e (u\{p})
```

Teorema 1 ([12])**.** *Sejam A e B dois genomas definidos no mesmo conjunto de N genes, temos que*

$$d_{DCJ}(A, B) = N - (C + I/2)$$

onde C é o número de ciclos e I o número de caminhos de comprimento ímpar em $G_{Adj}(A, B)$. Uma sequência ótima de ordenação pode ser encontrada em $O(N)$ pelo Algoritmo 3.

3.2 Árvores Filogenéticas e o Problema da Mediana

Definição 17. *Uma* **árvore binaria sem raiz** *é um grafo conexo sem ciclos, onde cada vértice tem grau um ou três. Os vértices com grau um são chamados de folhas e os vértices de grau três são chamados de nós internos.*

Definição 18. *Uma* **árvore filogenética** *(ou árvore evolutiva) de um conjunto S de n genomas é uma árvore binaria sem raiz com n folhas, onde cada folha é rotulada com um elemento distinto de S, ou seja, a cada folha é atribuído um elemento distinto de S.*

Definição 19. *Dada uma árvore filogenética cujos nós internos foram rotulados (i.e foram atribuídos um genoma), o* **custo da árvore** *é a soma dos custos de todas as suas arestas, onde o* **custo de uma aresta** *é a distância evolutiva entre os genomas rotulando os extremos da aresta.*

Definição 20. *Dado um conjunto S de n genomas e uma distância evolutiva (e.g. distância de reversão), o* **problema da grande filogenia** *consiste em encontrar uma árvore filogenética (estrutura) e uma rotulação dos nós internos tal que o custo da árvore seja mínimo.*

Definição 21. *Dado um conjunto S de n genomas, uma distância evolutiva, e uma estrutura de uma árvore filogenética, o* **problema da pequena filogenia** *consiste em encontrar uma rotulação dos nós internos tal que o custo da árvore seja mínimo.*

Um caso especial do problema da pequena filogenia ($n = 3$) é o problema da mediana para três genomas, o qual foi provado que é $\mathcal{NP}$-Difícil para vários modelos de rearranjo [62, 22, 83, 84].

3.3 O Problema da Mediana para Operações DCJ é $\mathcal{NP}$-Difícil

Tannier et al. [83, 84] demonstraram que o problema da mediana para operações DCJ é $\mathcal{NP}$-Difícil. Nesta seção será apresentada a mesma prova proposta em [83, 84] com alguns exemplos adicionais.

Definição 22. *Dados os genomas A, B e C, o* **problema da mediana (para operações) DCJ** *consiste em encontrar um genoma M tal que a seguinte soma $d(A, M) + d(B, M) + d(C, M)$ é minimizada. Onde d é a distância DCJ.*

Definição 23. *Um grafo é* ***bicolor*** *se todas as suas arestas são coloridas de vermelho ou azul; é* ***equilibrado*** *se só tem vértices de grau 2 ou grau 4, cada vértice é incidente ao mesmo número de arestas vermelhas e azuis, e não existe um ciclo formado por só vértices vermelhos ou só vértices azuis.*

Definição 24. *Seja G um grafo bicolor equilibrado, o* ***problema de decomposição maximal de ciclos alternados*** *(maximum alternating-cycle decomposition) consiste em encontrar uma decomposição em ciclos de arestas alternadas (vermelho-azul) de G tal que o número de ciclos seja maximal.*

Caprara [21] provou que o problema de decomposição maximal de ciclos alternados é $\mathcal{NP}$-Difícil.

Teorema 2 ([83, 84])**.** *O problema da mediana DCJ para genomas com múltiplos cromossomos é $\mathcal{NP}$-Difícil.*

Demonstração. A redução é feita a partir do problema de decomposição maximal de ciclos alternados.

Seja G um grafo bicolor equilibrado com n vértices. Defina o conjunto de genes $\mathcal{G}$ como o conjunto contendo um gene x por cada vértice de grau 2 de G, e dois genes x e y por cada vértice de grau 4 de G.

Construa os genomas Π_1, Π_2, Π_3 na seguinte forma como ilustrado na Figura 3.7.

Seja v um vértice de grau 2 em G. Substituir v por dois vértices que serão rotulados pelas duas extremidades do gene x associado a v, isto é x_t e x_h. A aresta vermelha incidente a v se torna incidente a x_t, e a aresta azul se torna incidente a x_h. Adicionar uma aresta-Π_1 e outra aresta-Π_2 entre x_t e x_h. Agora, seja v um vértice de grau 4 em G. Substituir v por 4 vértices rotulados pelas quatro extremidades dos genes x e y associados a v, isto é x_t, x_h, y_h, y_t. Os vértices vermelhos se tornam incidentes a x_t e y_t, enquanto que os vértices azuis se tornam incidentes a x_h e y_h. Adicionar duas arestas-Π_1 $x_t x_h$ e

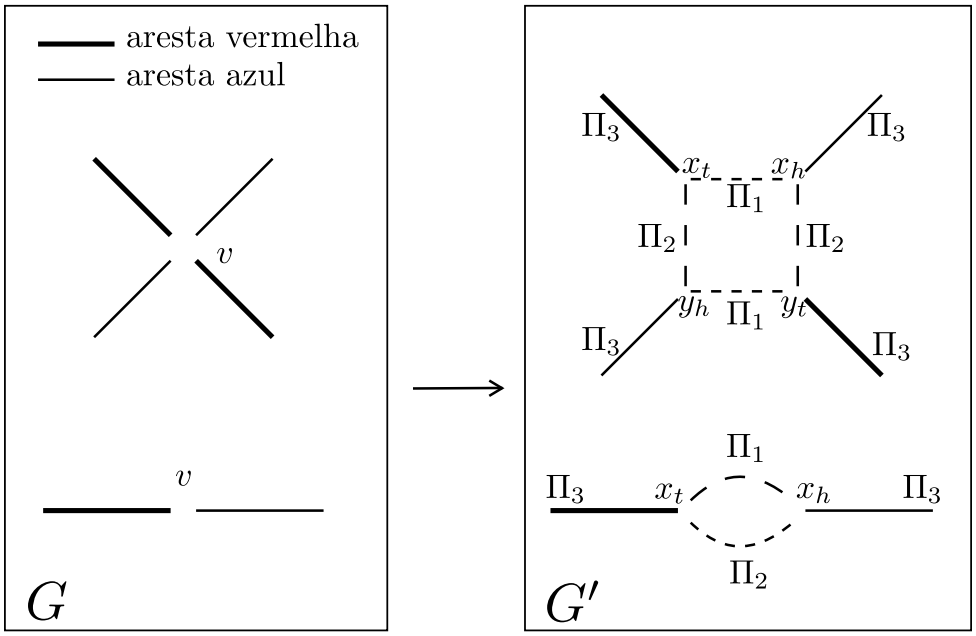

Figura 3.7: Estratégia para reduzir o problema de decomposição maximal de ciclos alternados ao problema da mediana DCJ.

$y_t y_h$, e duas arestas-Π_2 $x_t y_h$ e $y_t x_h$. Finalmente, as arestas vermelhas e azuis se tornam arestas-Π_3. Chamaremos este grafo final de G'.

Note que Π_1, Π_2, Π_3 definem genomas no conjunto de genes $\mathcal{G}$, e que não tem telômeros. Seja w_2 o número de vértices em G que tem grau 2, e w_4 o número de vértices em G que tem grau 4.

(A seguinte afirmação implica o teorema.)

Afirmação 1. *Existe um genoma M tal que $d(M,\Pi_1) + d(M,\Pi_2) + d(M,\Pi_3) \leq w_2 + 3w_4 - k$, se e somente se existem pelo menos k ciclos de arestas alternadas em G.*

($\Leftarrow$): Suponha que existem k ciclos de arestas alternadas em G. Construiremos a mediana M (genoma M) tal que $d(M,\Pi_1)+d(M,\Pi_2)+d(M,\Pi_3) \leq w_2+3w_4-k$. Primeiro, por cada vértice v de G de grau 2, adicionar a adjacência $\{x^t, x^h\}$ em M. Depois, seja v um vértice de G de grau 4, e seja vw uma aresta azul incidente a v. Em um ciclo alternado, a aresta vw é consecutiva com uma aresta vermelha, isto é uv, como ilustrado na Figura 3.8.

A aresta vw é associada com uma aresta-Π_3, isto é $x^h w^h$. Então, a aresta-Π_3 associada a uv contém como extremidade o gene x^t ou contém a extremidade y^t. No primeiro caso, as adjacências $\{x^h, x^t\}$ e $\{y^t, y^h\}$ são adicionadas a M,e no segundo caso, as adjacências $\{x^h, y^t\}$ e $\{x^t, y^h\}$ são adicionadas a M. Já que o genoma M não contém telômeros, as adjacências definem um genoma circular em $\mathcal{G}$. Na Figura 3.7 pode-se perceber que por

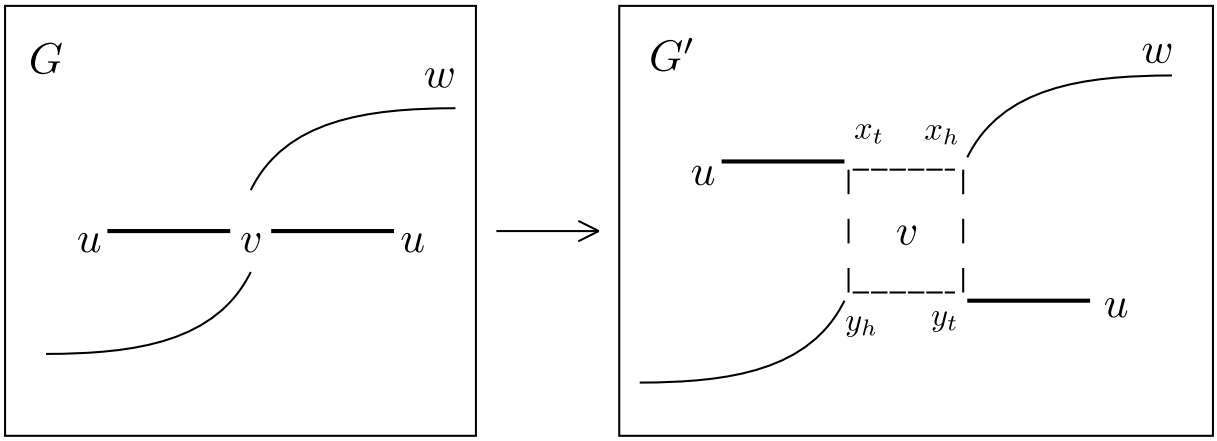

Figura 3.8: Possíveis caminhos em uma decomposição de ciclos.

cada vértice em G de grau 2 temos um gene x, e por cada vértice em G de grau 4 temos dois genes x e y, então o número de genes é $w_2 + 2w_4$. Pelo Teorema 1 temos que

$$\begin{aligned} d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M) &= 3(w_2 + 2w_4) \\ &\quad -(c(\Pi_1, M) + c(\Pi_2, M) + c(\Pi_3, M)) \\ &\quad -(I(\Pi_1, M)/2 + I(\Pi_2, M)/2 + I(\Pi_3, M)/2) \end{aligned}$$

Construindo os grafos $G_{Adj}(\Pi_1, M)$, $G_{Adj}(\Pi_2, M)$, e $G_{Adj}(\Pi_3, M)$, temos que o valor de $I(\Pi_1, M)$, $I(\Pi_2, M)$, e $I(\Pi_3, M)$ é zero nos três casos porque não existem telômeros, e portanto não existem caminhos (de comprimento impar). Nos grafos de adjacências $G_{Adj}(\Pi_1, M)$ e $G_{Adj}(\Pi_2, M)$ pode-se verificar que por cada gene x associado a um vértice de G de grau 2 existe um ciclo, e portanto existe um total de $2w_2$ ciclos. Nesses mesmos grafos pode-se verificar que por cada gene x associado a um vértice de G de grau 4 existem 2 ciclos em $G_{Adj}(\Pi_1, M)$ e um ciclo em $G_{Adj}(\Pi_2, M)$, ou vice-versa, e portanto existem um total de $3w_4$ ciclos. Assim, $c(M, \Pi_1) + c(M, \Pi_2) = 2w_2 + 3w_4$ e $c(M, \Pi_3) = k$, portanto,

$$d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M) = w_2 + 3w_4 - k.$$

($\Rightarrow$) Suponha que M é um genoma tal que $d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M) \leq w_2 + 3w_4 - k$. Suponha que M é escolhido tal que $d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M)$ é mínimo, e entre todos esses genomas, escolher M com um número máximo de arestas paralelas as arestas-Π_1 ou arestas-Π_2. Um genoma circular se diz que é *canônico* se tem somente adjacências que pertencem a Π_1 ou Π_2. Provaremos que M é canônico.

Sub-afirmação. *M é canônico.*

Suponha que M não é canônico. Primeiro, suponha que existe um vértice v de grau 2 em G, tal que M não contém a adjacência $\{x^t, x^h\}$. Já que v é um vértice de grau 2, a adjacência $\{x^t, x^h\}$ está incluída em Π_1 e Π_2, mas não em Π_3. Suponha o primeiro subcaso em que M contém as adjacências $\{x^t, a\}$ e $\{x^h, b\}$, e o segundo subcaso em que M contém os telômeros $\{x^t\}$ e $\{x^h\}$. Então, no primeiro subcaso substituir $\{x^t, a\}$ e $\{x^h, b\}$ pelas adjacências $\{x^h, x^t\}$ e $\{a, b\}$ como mostrado na Figura 3.9. No segundo subcaso substituir os telômeros $\{x^t\}$ e $\{y^h\}$ pela adjacência $\{x^h, x^t\}$ como mostrado na Figura 3.10.

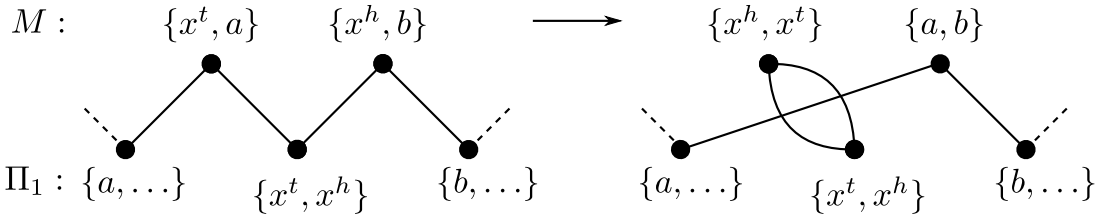

Figura 3.9: Supondo que M não é canônico e existe vértice v de grau 2. Caso onde duas adjacências são substituídas em $G_{Adj}(\Pi_1, M)$(ou $G_{Adj}(\Pi_2, M)$).

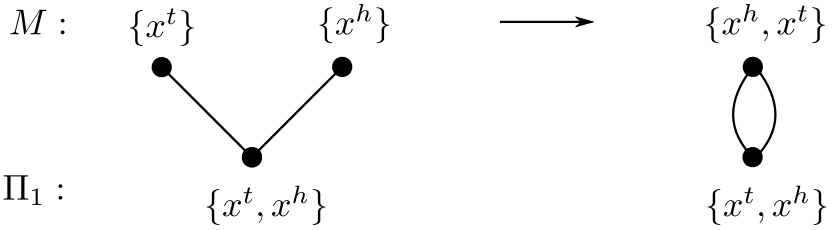

Figura 3.10: Supondo que M não é canônico e existe vértice v de grau 2. Caso onde dois telômeros são substituídos em $G_{Adj}(\Pi_1, M)$(ou $G_{Adj}(\Pi_2, M)$).

Por meio dessa operação o valor de $c(M, \Pi_1)$ e $c(M, \Pi_2)$ é incrementado pelo menos por um (ver Figura 3.9 e 3.10), e o valor de $c(M, \Pi_3)$ diminui no máximo em um (ver Figura 3.11 e 3.12), assim $d(\Pi_1, M)+d(\Pi_2, M)+d(\Pi_3, M)$ diminui em um, o que contradiz a hipótese.

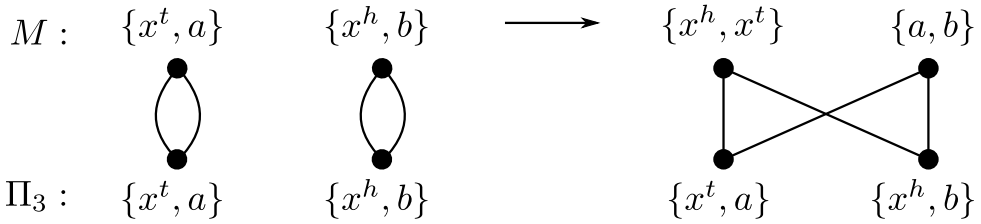

Figura 3.11: Supondo que M não é canônico e existe vértice v de grau 2. Caso onde duas adjacências são substituídas $G_{Adj}(\Pi_3, M)$.

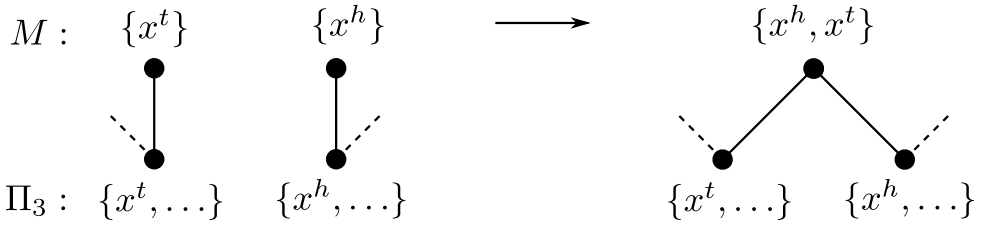

Figura 3.12: Supondo que M não é canônico e existe vértice v de grau 2. Caso onde dois telômeros são substituídos $G_{Adj}(\Pi_3, M)$.

Agora, suponha que existe um vértice v de grau 4 em G tal que M não contém as adjacências $\{x^h, x^t\}$, $\{y^h, y^t\}$, $\{x^h, y^t\}$, e $\{y^h, x^t\}$. Já que v é um vértice de grau 4 as adjacências antes mencionadas estão incluídas em Π_1 e Π_2 mas não em Π_3. Suponha o primeiro subcaso em que M contém as adjacências $\{x^h, a\}$ $\{x^t, b\}$ $\{y^h, c\}$ $\{y^t, d\}$, e o segundo subcaso em que M contém os telômeros $\{x^h\}$, $\{x^t\}$, $\{y^h\}$, e $\{y^t\}$. Então, no primeiro subcaso substituir $\{x^h, a\}$, $\{x^t, b\}$, $\{y^h, c\}$, e $\{y^t, d\}$ (ver Figura 3.13) pelas adjacências $\{x^h, x^t\}$, $\{y^h, y^t\}$, e ($\{a, b\}$, $\{c, d\}$) ou ($\{a, c\}$, $\{b, d\}$) de acordo a combinação que cria o maior número de ciclos em $G_{Adj}(M, \Pi_3)$.

No segundo subcaso substituir os telômeros $\{x^h\}$, $\{x^t\}$, $\{y^h\}$, e $\{y^t\}$ pelas adjacências $\{x^h, x^t\}$ e $\{y^h, y^t\}$, como mostrado na Figura 3.14.

Suponha agora que M contém só uma das seguintes adjacências $\{x^h, x^t\}$, $\{y^h, y^t\}$, $\{x^h, y^t\}$, ou $\{y^h, x^t\}$, digamos que contém a adjacência $\{x^h, x^t\}$ e adicionalmente as adjacências $\{y^t, b\}$ e $\{y^h, c\}$. Então, substituir $\{y^t, b\}$ e $\{y^h, c\}$ pelas adjacências $\{y^h, y^t\}$ e $\{b, c\}$, como mostrado na Figura 3.15.

Todas as operações vistas para caso de um vértice de grau 4 (ver Figuras 3.13, 3.14, e 3.15), diminuem o valor de $d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M)$ ou o mantêm constante,

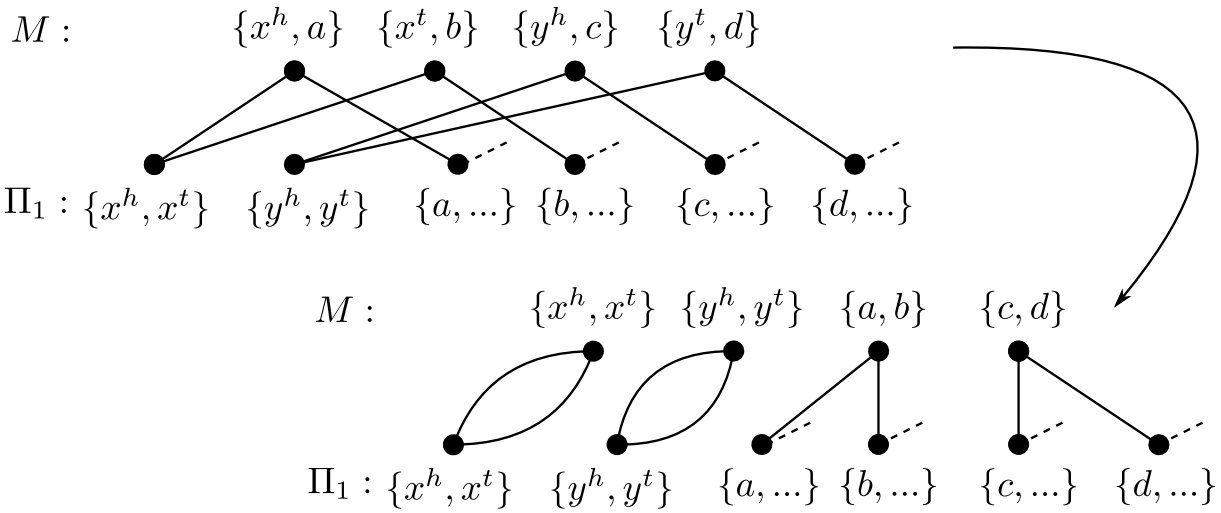

Figura 3.13: Supondo que M não é canônico e existe vértice v de grau 4. Caso onde quatro adjacências são substituídas em $G_{Adj}(\Pi_1, M)$(ou $G_{Adj}(\Pi_2, M)$).

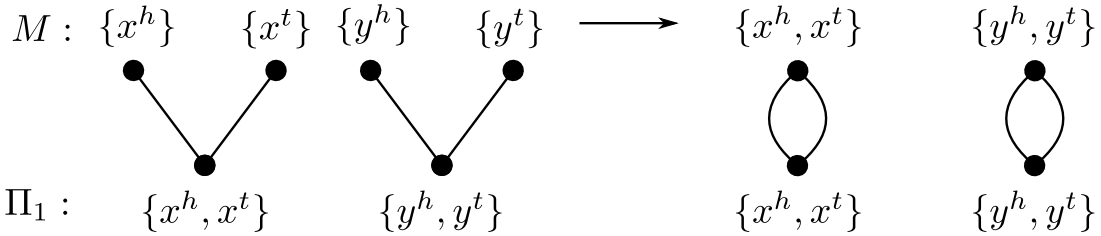

Figura 3.14: Supondo que M não é canônico e existe vértice v de grau 4. Caso onde quatro telômeros são substituídos em $G_{Adj}(\Pi_1, M)$(ou $G_{Adj}(\Pi_2, M)$).

enquanto que incrementam o número de arestas paralelas as arestas- Π_1 e as arestas- Π_2, o que contradiz a hipótese. Assim, a Sub-afirmação está provada.

Finalmente, como M é canônico existem $c(\Pi_3, M)$ ciclos de arestas alternadas em G. Já que uma adjacência de M sempre junta uma cabeça com uma cauda, a aresta correspondente a esta adjacência em G tem uma aresta vermelha adjacente a um de seus vértices e uma aresta azul no outro. Pelo Teorema 1, e já que $I(\Pi_1, M)$, $I(\Pi_2, M)$, e $I(\Pi_3, M)$ tem valor zero nos três casos, temos que

$$\begin{aligned} d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M) &= 3(w_2 + 2w_4) \\ &\quad -(c(\Pi_1, M) + c(\Pi_2, M) + c(\Pi_3, M)) \end{aligned}$$

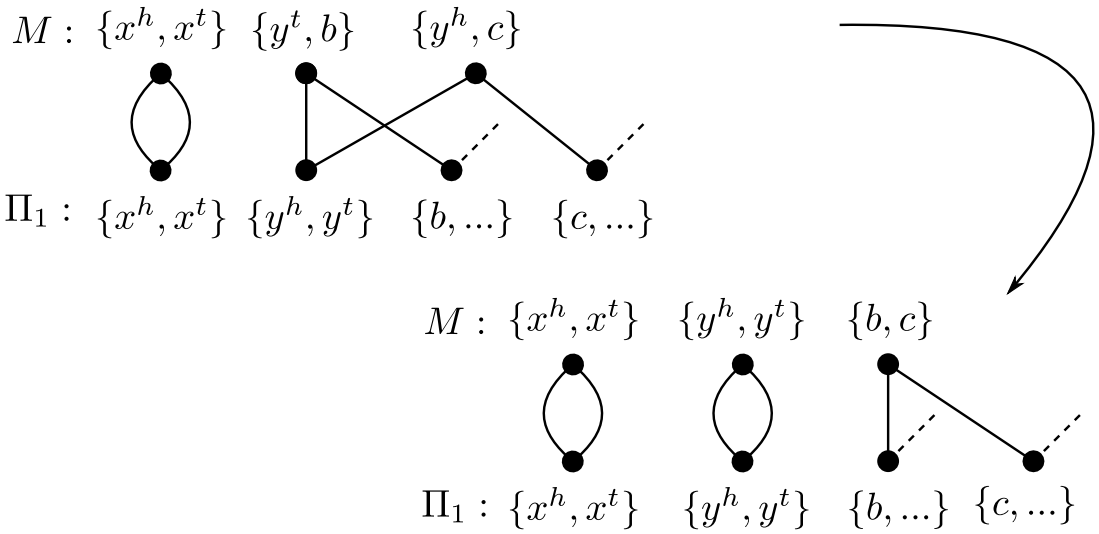

Figura 3.15: Supondo que M não é canônico e existe vértice v de grau 4. Caso onde duas adjacências são substituídas em $G_{Adj}(\Pi_1, M)$(ou $G_{Adj}(\Pi_2, M)$).

Como M é canônico sabemos que $c(M, \Pi_1) + c(M, \Pi_2) = 2w_2 + 3w_4$, e pela hipótese temos que:

$$\begin{aligned} d(\Pi_1, M) + d(\Pi_2, M) + d(\Pi_3, M) &\leq w_2 + 3w_4 - k \\ 3(w_2 + 2w_4) - (c(\Pi_1, M) + c(\Pi_2, M) + c(\Pi_3, M)) &\leq w_2 + 3w_4 - k \\ 3(w_2 + 2w_4) - (c(\Pi_1, M) + c(\Pi_2, M)) - w_2 - 3w_4 + k &\leq c(\Pi_3, M)) \\ 3(w_2 + 2w_4) - (2w_2 + 3w_4) - w_2 - 3w_4 + k &\leq c(\Pi_3, M)) \\ k &\leq c(\Pi_3, M)) \end{aligned}$$

O resultado anterior prova a Afirmação 1. □

3.4 O Problema da Mediana para Reversões é $\mathcal{NP}$-Difícil

Caprara [22] demonstrou que o problema da mediana para reversões é $\mathcal{NP}$-Difícil. Nesta seção será apresentada a mesma prova proposta em [22] com alguns exemplos adicionais. Considere somente permutações com sinal nesta seção (como foram definidas no Capitulo 2).

Definição 25. *Dadas as permutações $\pi^1, \ldots, \pi^q$, $q \geq 3$, o* **problema da mediana para reversões** *consiste em encontrar uma permutação σ tal que $\delta(\sigma) = \sum_{k=1}^{q} d(\sigma, \pi^k)$ é minimizada, onde $d(\sigma, \pi^k)$ é a distância de reversão entre σ e π^k.*

Definição 26. *Dado um conjunto de nós V, o conjunto de arestas $M \subset \{(i,j) : i,j \in V, i \neq j\}$ é um* **emparelhamento** *de V se cada nó de V é incidente a no máximo uma aresta de M. Se cada nó de V é incidente a exatamente uma aresta de M, o emparelhamento é chamado de* **perfeito**.

Definição 27. *Um* **ciclo hamiltoniano** *de V é um ciclo que visita todos os nós de V.*

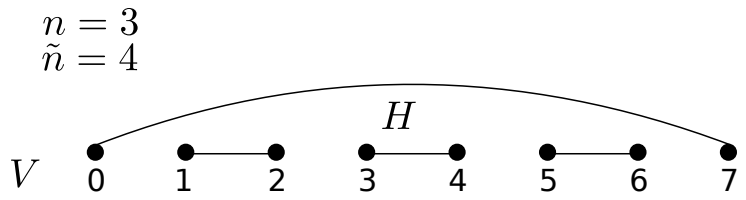

Figura 3.16: Emparelhamento hamiltoniano H para $n = 3$.

Definição 28. *Considere o conjunto de nós $V = \{0, 1, \ldots, 2n, 2n+1\}$, o emparelhamento perfeito $H = \{(2i-1, 2i) : i = 1, \ldots, n\} \cup \{0, 2n+1\}$ é chamando de* **emparelhamento hamiltoniano** *de V, onde n é comprimento de uma permutação π.*

Defina ñ $= n + 1$ como a cardinalidade de qualquer emparelhamento perfeito de V. Na Figura 3.16 aparece o emparelhamento hamiltoniano para $n = 3$.

Definição 29. *Considere o conjunto de nós $V = \{0, 1, \ldots, 2n, 2n+1\}$, um* **emparelhamento de permutação** *é um emparelhamento perfeito de V que junto com H define um ciclo hamiltoniano de V. Em particular, o emparelhamento de permutação $M(\pi)$ associado à permutação π é definido por: $M(\pi) = \{(2|\pi_i| - v(\pi_i), 2|\pi_{i+1}| - 1 + v(\pi_{i+1})) : i \in \{0, \ldots, n\}\}$, onde $\pi_0 = 0$,$\pi_{n+1} = n + 1$ e $v(\pi_i) = 0$ se $\pi_i \geq 0$, $v(\pi_i) = 1$ se $v(\pi_i) < 0$.*

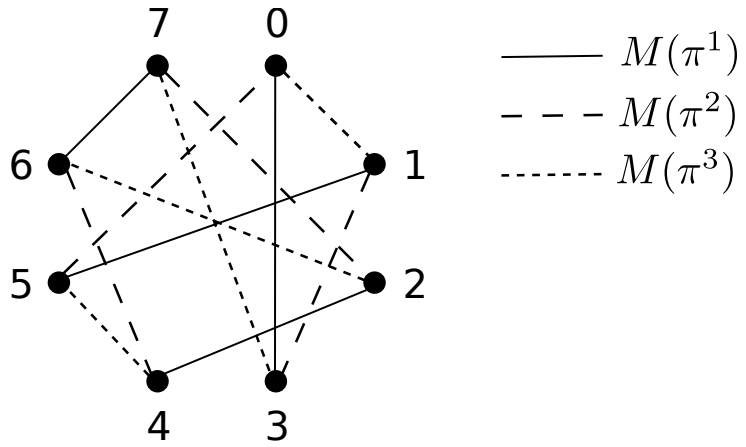

Figura 3.17: Grafo de ponto de quebra múltiplo $G(\pi^1, \pi^2, \pi^3)$ associado com as permutações $\pi^1 = (2 - 1\ 3)$, $\pi^2 = (3 - 2\ 1)$, e $\pi^3 = (1 - 3 - 2)$

Definição 30. *Dadas duas permutações π^1 e π^2, o conjunto de arestas $M(\pi^1 \cup \pi^2)$ define um conjunto de ciclos em V cujas arestas alternam entre $M(\pi^1)$ e $M(\pi^2)$. Denotamos com $c(\pi^1, \pi^2)$ o número destes ciclos.*

Definição 31. *Definimos um* **grafo de ponto de quebra múltiplo** *associado com as permutações $\pi^1, \ldots, \pi^q$ como o grafo $G(\pi^1, \ldots, \pi^q)$ tendo V como conjunto de nós e $M(\pi^1) \cup \ldots \cup M(\pi^k)$ como multi-conjunto de arestas.*

As arestas que estão em $M(\pi^k)$ são chamadas de **k-arestas**, com $k \in Q = \{1, \ldots, q\}$. A Figura 3.17 mostra o grafo de ponto de quebra múltiplo associado às permutações $\pi^1 = (2 - 1\ 3)$, $\pi^2 = (3 - 2\ 1)$, e $\pi^3 = (1 - 3 - 2)$.

Definição 32. *Dada uma instância do problema da mediana para reversões definida pelas permutações $\pi^1, \ldots, \pi^q$, o* **problema de Emparelhamento de Permutações com número Máximo de Ciclos** *(EPMC) consiste em encontrar uma permutação σ tal que $\gamma(\sigma) = \sum_{k=1}^{q} c(\sigma, \pi^k)$ é maximizada. Desde o ponto de vista do grafo de ponto de quebra múltiplo $G(\pi^1, \ldots, \pi^q)$ o problema consiste em encontrar um emparelhamento de permutação T tal que o número total de ciclos definido por $T \cup M(\pi^k)$, $k \in \{1, \ldots, q\}$ é maximizado.*

A correspondência entre as soluções σ e T é dado por $M(\sigma) = T$.

Definição 33. *Considere o grafo $\mathcal{G}$ tendo como conjunto de nós $V = \{0, \ldots, 2n+1\}$ cujo conjunto de arestas é particionado em três emparelhamentos perfeitos de V, digamos M_1, M_2, M_3 o* **problema de Emparelhamento com número Máximo de Ciclos** *(EMC) em $\mathcal{G}$ consiste em encontrar um emparelhamento perfeito T de V tal que $\sum_{k=1}^{3} c(T, M_k)$ é maximizada. Note que quando $\mathcal{G}$ é uma grafo de ponto de quebra múltiplo, a solução T não precisa ser um emparelhamento de permutação.*

A continuação, primeiro vamos a demonstrar que o problema EPMC é $\mathcal{NP}$-Difícil, e a partir desse resultado mostrar que o problema da mediana para reversões é $\mathcal{NP}$-Difícil.

Começamos mostrando uma redução polinomial do problema de decomposição maximal de ciclos ao problema EPMC.

Dado um grafo bicolor equilibrado $\mathcal{E} = (W, R \cup B)$ (R e B são as arestas vermelhas e azuis respectivamente) construiremos um grafo de ponto de quebra múltiplo $G(\mathcal{E})$. Denotamos como W_2 e W_4 o conjunto de nós de $\mathcal{E}$ com grau 2 e 4 respectivamente. Cada aresta $(i, j) \in R \cup B$ é substituída por uma 1-aresta. Logo, cada nó $i \in W_2$ é substituído por dois nós i_1 e i_2 e duas arestas paralelas (i_1, i_2) (2-aresta e 3-aresta). Cada nó $i \in W_4$ é substituído por 4 nós i_1, i_2, i_3, i_4, duas 2-arestas (i_1, i_2) (i_3, i_4), e duas 3-arestas (i_2, i_3)

(i_1, i_4). A redução é mostrada na Figura 3.18. Assim, uma solução ótima para o problema EPMC contém arestas paralelas às 2-arestas e 3-arestas, e define junto com as 1-arestas o mesmo número de ciclos que existem em uma solução ótima do problema de decomposição maximal de ciclos.

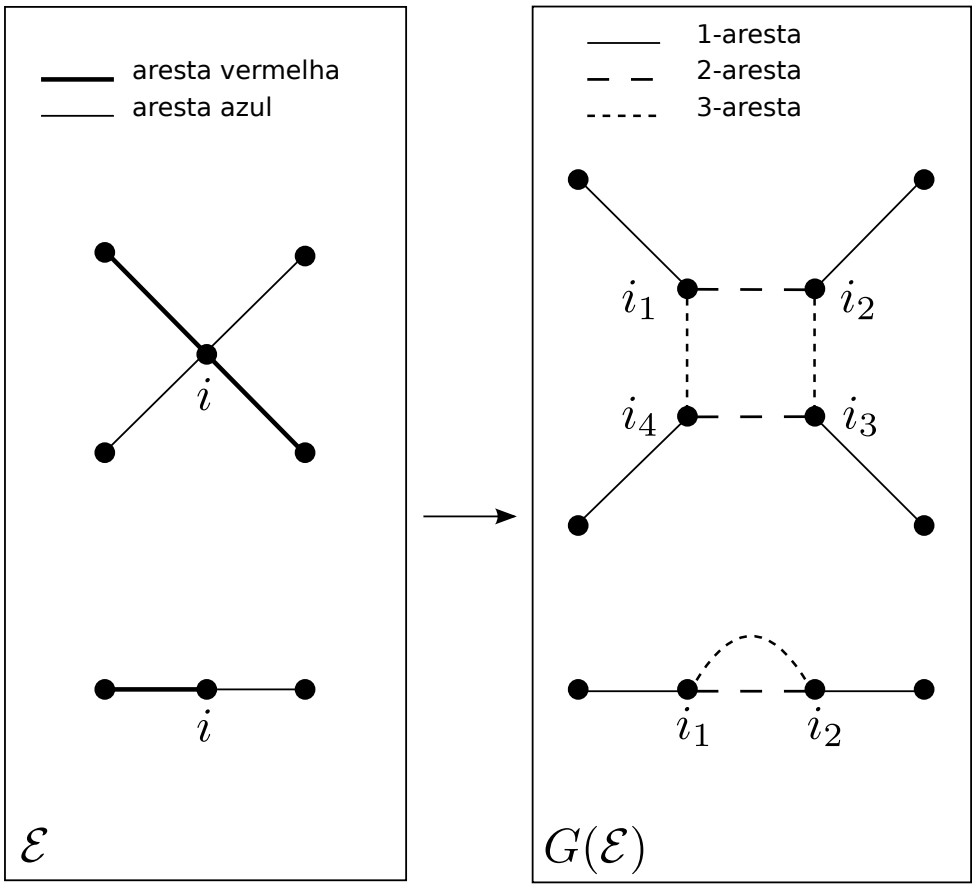

Figura 3.18: Estratégia para reduzir o problema de decomposição maximal de ciclos alternado ao problema EPMC.

Lema 1 ([22])**.** *$G(\mathcal{E})$ é um grafo de ponto de quebra múltiplo com um emparelhamento hamiltoniano H.*

Demonstração. Omitida. □

Agora vamos a mostrar que a partir de solução ótima para o problema EPMC em $G(\mathcal{E})$ podemos derivar uma solução ótima para o problema de decomposição maximal de ciclos em $\mathcal{E}$. Para isso precisamos de duas definições adicionais.

Um **grafo de ciclos pequenos** é um grafo $\mathcal{G}$ com conjunto de nós $V = \{0, \ldots, 2n+1\}$ cujo conjunto de arestas está particionado em 3 emparelhamentos perfeitos de V, M_1, M_2, M_3 (chamados de 1-arestas, 2-arestas, e 3-arestas), tal que cada ciclo definido por $M_2 \cup M_3$ tem comprimento 2 ou 4, e nenhuma aresta em M_1 é paralela a alguma aresta em $M_2 \cup M_3$.

Deixe $F(\mathcal{G})$ denotar o grafo contendo um nó por cada ciclo definido por $M_2 \cup M_3$, e uma aresta entre cada par de nós tal que os correspondentes ciclos em $\mathcal{G}$ estão conectados por uma 1-aresta. As arestas de $F(\mathcal{G})$ são coloridas azul e vermelho de modo a garantir que é um grafo bicolor equilibrado e que nenhuma 2-aresta e 3-aresta conecta em $\mathcal{G}$ duas 1-arestas correspondente a arestas da mesma cor em $F(\mathcal{G})$.

Deixe $t(\mathcal{G})$ denotar o número de ciclos de comprimento 2 definido por $M_2 \cup M_3$, e $\Gamma(F(\mathcal{G}))$ denotar o valor de uma solução ótima para o problema de decomposição maximal de ciclos em $F(\mathcal{G})$.

Lema 2 ([22]). *Uma solução ótima para o problema EMC em um grafo de ciclos pequenos $\mathcal{G}$ tem valor* $\tilde{\mathrm{n}} + \frac{\tilde{\mathrm{n}} - t(\mathcal{G})}{2} + t(\mathcal{G}) + \Gamma(F(\mathcal{G}))$ *e existe uma solução ótima que contém só arestas paralelas a* 2*-arestas e* 3*-arestas.*

Demonstração. Qualquer solução para o problema de decomposição maximal de ciclos em $F(\mathcal{G})$ define uma solução T para o problema EMC em $\mathcal{G}$ com valor $\tilde{\mathrm{n}} + \frac{\tilde{\mathrm{n}} - t(\mathcal{G})}{2} + t(\mathcal{G}) + \Gamma(F(\mathcal{G}))$. A solução T contém todas suas arestas paralelas a 2-arestas e 3-arestas conectando 1-arestas correspondentes a arestas consecutivas em um ciclo de uma solução para o problema de decomposição maximal de ciclos. Assim, T define $\Gamma(F(\mathcal{G}))$ ciclos com as 1-arestas, e $\tilde{\mathrm{n}} + \frac{\tilde{\mathrm{n}} - t(\mathcal{G})}{2} + t(\mathcal{G})$ ciclos com as 2-arestas e 3-arestas. Esta é a melhor solução para o problema EMC a qual pode ser obtida usando somente arestas paralelas a 2-arestas e 3-arestas, já que qualquer solução define $\tilde{\mathrm{n}} + \frac{\tilde{\mathrm{n}} - t(\mathcal{G})}{2} + t(\mathcal{G})$ ciclos com as 2-arestas e 3-arestas, e um máximo de $\Gamma(F(\mathcal{G}))$ ciclos com as 1-arestas.

Agora suponha que $\mathcal{G}$ é um contra-exemplo. Em particular, existe uma solução ótima T ao problema EMC tendo um valor estritamente maior que $\tilde{\mathrm{n}} + \frac{\tilde{\mathrm{n}} - t(\mathcal{G})}{2} + t(\mathcal{G}) + \Gamma(F(\mathcal{G}))$, e portanto essa solução tem alguma aresta não paralela a 2-arestas e 3-arestas. Assumimos por minimidade que T contém só arestas que não são paralelas a 2-arestas ou 3-arestas.

Dados três emparelhamentos perfeitos T, S, R, de V, podemos verificar facilmente que a seguinte relação se cumpre: $c(T,R) + c(R,S) \leq \tilde{\mathrm{n}} + c(T,S)$. Na Figura 3.19 é apresentando um exemplo especifico mostrando a relação. Por definição de $\mathcal{G}$ temos que M_1 e M_2 não tem arestas paralelas portanto o valor de $c(M_1, M_2)$ é no máximo $\Gamma(F(\mathcal{G}))$, ou seja $c(M_1, M_2) \leq \Gamma(F(\mathcal{G}))$. Assim, a partir dar ultimas duas relações temos que: $c(T, M_1) + c(T, M_2) \leq \tilde{\mathrm{n}} + c(M_1, M_2)$, e que $\tilde{\mathrm{n}} + c(M_1, M_2) \leq \tilde{\mathrm{n}} + \Gamma(F(\mathcal{G}))$, por conseguinte $c(T, M_1) + c(T, M_2) \leq \tilde{\mathrm{n}} + \Gamma(F(\mathcal{G}))$. Adicionalmente, temos que $c(T, M_3) \leq \frac{\tilde{\mathrm{n}}}{2}$, já que assumimos que T não tem nenhuma aresta paralela a 3-arestas e portanto $T \cup M_3$ não tem nenhum ciclo de comprimento 2. Finalmente, somando as relações $c(T, M_1) + c(T, M_2) \leq$

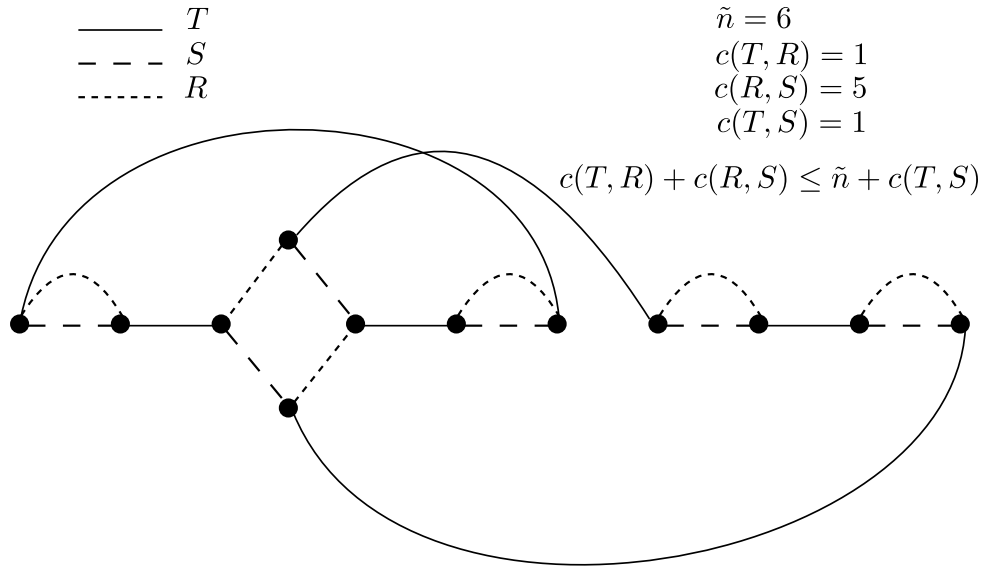

Figura 3.19: Exemplo mostrando a relação $c(T,R) + c(R,S) \leq \tilde{\text{n}} + c(T,S)$.

$\tilde{\text{n}} + \Gamma(F(\mathcal{G}))$ e $c(T, M_3) \leq \frac{\tilde{\text{n}}}{2}$ temos que:

$$\begin{aligned} c(T,M_1) + c(T,M_2) + c(T,M3) &\leq \tilde{\text{n}} + \tfrac{\tilde{\text{n}}}{2} + \Gamma(F(\mathcal{G})) \\ \textstyle\sum_{k=1}^{3} c(T,M_k) &\leq \tilde{\text{n}} + \tfrac{\tilde{\text{n}}}{2} + \Gamma(F(\mathcal{G})) \\ &\leq \tilde{\text{n}} + \tfrac{\tilde{\text{n}} - t(\mathcal{G})}{2} + t(\mathcal{G}) + \Gamma(F(\mathcal{G})) \end{aligned}$$

o que é uma contradição. □

Lema 3 ([22])**.** *$G(\mathcal{E})$ é uma grafo de ciclos pequenos e $F(G(\mathcal{E}))$ é isomórfico a $\mathcal{E}$.*

Demonstração. Pela definição de $G(\mathcal{E})$ segue que é um grafo de ciclos pequenos. Logo, pela definição de $F(.)$ temos que existe uma correspondência biunívoca entre os vértices de $F(G(\mathcal{E}))$ e $\mathcal{E}$, e também existe uma correspondência biunívoca entre as arestas(e correspondentes vértices) de $F(G(\mathcal{E}))$ e $\mathcal{E}$. Portanto, $F(G(\mathcal{E}))$ é isomórfico a $\mathcal{E}$. □

Lema 4 ([22])**.** *Qualquer solução para o problema EMC em $G(\mathcal{E})$ contendo somente arestas paralelas a 2-arestas e 3-arestas é uma solução viável ao problema EPMC.*

Demonstração. Devemos demonstrar que a solução T para o problema EMC é um emparelhamento de permutação, ou seja que junto com o emparelhamento perfeito H definem um ciclo hamiltoniano. Primeiro considere os nós i_1 e i_2 que se correspondem com um nó $i \in W_2$, neste caso $T \cup H$ sempre vai definir um emparelhamento de permutação (Veja parte esquerda da Figura 3.20). Logo, considere os nós j_1, j_2, j_3, j_4 que se correspondem com um nó $j \in W_4$, veja na parte direita da Figura 3.20 que a aresta (j_1, j_3) de H garante um ciclo hamiltoniano, já que T vai ter as arestas (j_1, j_2) e (j_3, j_4) ou vai ter as arestas (j_2, j_3) e (j_1, j_4). □

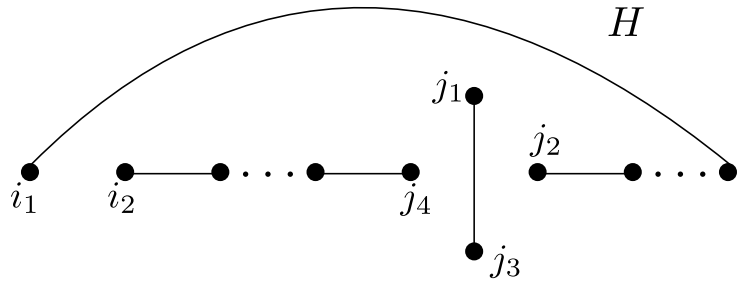

Figura 3.20: Emparelhamento perfeito H.

Teorema 3 ([22])**.** *O problema EPMC é $\mathcal{NP}$-Difícil, inclusive para 3 permutações.*

Demonstração. Dos lemas 1, 2, 3, e 4, temos que uma solução ótima para o problema EMC em $G(\mathcal{E})$ é a mesma solução ótima para o problema EPMC, e que a partir dessa solução podemos produzir uma solução ótima para o problema de decomposição maximal de ciclos. □

Definição 34. *Dado um emparelhamento de permutação $M(\pi^1)$, um* **emparelhamento dirigido** $\vec{M}(\pi^1)$ *é obtido substituindo cada aresta* $(2|\pi_i|-v(\pi_i), 2|\pi_{i+1}|-1+v(\pi_{i+1})) : i \in \{0,\ldots,n\}$ *por um arco dirigido que vai de* $2|\pi_i|-v(\pi_i)$ *até* $2|\pi_{i+1}|-1+v(\pi_{i+1})$.

Definição 35. *Um ciclo formado por arestas e arcos é chamado* **ciclo dirigido** *se tem pelo menos comprimento 4 e é possível percorrer cada arco na direção da sua orientação, caso contrário é um* **ciclo não dirigido**. *Note que por definição um ciclo de comprimento 2 não é dirigido.*

Um corolário do resultado principal proposto por Hannenhalli e Pevzner [44] reapresentado de acordo com a notação usada nesta seção é mostrado a continuação.

Teorema 4 ([44])**.** *Se cada ciclo definido por $\vec{M}(\pi^1) \cup M(\pi^2)$ não é dirigido, então $d(\pi^1,\pi^2) = \tilde{n} - c(\pi^1,\pi^2)$ (onde d é a distância de reversão).*

Agora vamos a mostrar com usar o problema da mediana para reversões para solucionar uma instância do problema EPMC em um grafo de ponto de quebra múltiplo $G(\mathcal{E})$ definido a partir do grafo $\mathcal{E}$ de uma instância genérica do problema de decomposição maximal de ciclos alternados. A partir do grafo $G(\mathcal{E})$ geramos o grafo $G'(\mathcal{E})$ tal que existe uma correspondência entre as soluções ótimas do problema EPMC em $G(\mathcal{E})$ e $G'(\mathcal{E})$. Adicionalmente, existe uma solução ótima T ao problema EPMC em $G'(\mathcal{E})$ com $\tau = M^{-1}(T)$ tal que $M(\tau) \cup \vec{M}(\pi^k)$ define só ciclos não dirigidos, $k \in \{1,2,3\}$, onde π^k é uma permutação associada a uma k-aresta de $G'(E)$. Neste caso, τ é também uma solução ótima a uma instância do problema da mediana para reversões definida pelas permutações π^1, π^2, π^3, e devido ao Teorema 4 e o Lema 5.

Lema 5 ([22]). *Se uma permutação τ é uma solução ótima a uma instância do problema EPMC definida pelas permutações $\pi^1, \ldots, \pi^q$, e $d(\tau, \pi^k) = \tilde{n} - c(\tau, \pi^k)$, $k \in \{1, \ldots, q\}$, então τ é uma solução ótima à instância do problema da mediana para reversões definida pelas permutações $\pi^1, \ldots, \pi^q$.*

Demonstração. Dada uma permutação arbitraria σ, e seja $Q = \{1, \ldots, q\}$, temos que $\delta(\sigma) = \sum_{k \in Q} d(\sigma, \pi^k) \geq q\tilde{n} - \sum_{k \in Q} c(\sigma, \pi^k)$. Logo, já que τ é uma solução ótima ao problema EPMC temos que $q\tilde{n} - \sum_{k \in Q} c(\sigma, \pi^k) \geq q\tilde{n} - \sum_{k \in Q} c(\tau, \pi^k) = \sum_{k \in Q} d(\tau, \pi^k) = \delta(\tau)$. □

Lema 6 ([22]). *Dado um grafo de ponto de quebra múltiplo G associado com um emparelhamento hamiltoniano H e a permutação π^k associada às k-arestas de G, $k \in 1, \ldots, q$, o ciclo hamiltoniano definido por $H \cup \vec{M}(\pi^k)$ é dirigido.*

Demonstração. O emparelhamento hamiltoniano H tem uma aresta $\{0, 2n+1\}$ e arestas $(2i-1, 2i)$ para $i \in \{1, \ldots, n\}$. Seja π uma permutação com emparelhamento dirigido $\vec{M}(\pi)$, suponha que existe a seguinte aresta dirigida $(2|\pi_{i-1}| - v(\pi_{i-1}), 2|\pi_i| - 1 + v(\pi_i))$, cuja segunda componente pode ter dois possíveis valores (dependendo do sinal de π_i), ou $2|\pi_i|-1$ ou $2|\pi_i|$. Logo, a próxima aresta dirigida começa em $2|\pi_i|$ ou em $2|\pi_i|-1$, respectivamente. Claramente podemos ver que o fim de uma aresta dirigida é ligado ao inicio da próxima aresta dirigida por uma aresta de H, isso é ou $(2|\pi_i|-1, 2|\pi_i|)$ ou $(2|\pi_i|, 2|\pi_i|-1)$. Aplicando o mesmo procedimento a todas arestas dirigidas de $\vec{M}(\pi)$ prova o lema. □

Seja H o emparelhamento hamiltoniano associado com $G(\mathcal{E})$, como definido anteriormente, e sejam π^1, π^2, e π^3 as permutações associadas com as 1-arestas, 2-arestas, e 3-arestas de $G(\mathcal{E})$ respectivamente. Agora, considere em $\mathcal{E}$ todos os nós $i \in W_4$, sabemos que H contém todas as arestas da forma (i_1, i_3), e devido ao Lema 6 os arcos correspondentes às 2-arestas (i_1, i_2) e (i_3, i_4) são orientados da seguinte forma ou $i_2 \to i_1$ e $i_3 \to i_4$ ou $i_1 \to i_2$ e $i_4 \to i_3$ (veja a Figura 3.21).

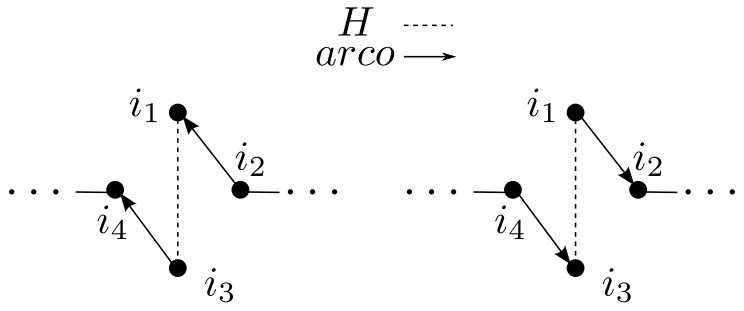

Figura 3.21: Dois possíveis orientações dos arcos correspondentes às 2-arestas (i_1, i_2) e (i_3, i_4).

Devido ao Lema 2 uma solução ótima T ao problema EPMC em $G(\mathcal{E})$ só contém arestas paralelas a 2-arestas e 3-arestas. Portanto todos os ciclos definidos por $T \cup \vec{M}(\pi^2)$ são não dirigidos. No caso das 2-arestas associadas a nós $i \in W_2$, T sempre formará ciclos de comprimento 2. No caso das 2-arestas associadas a nós $i \in W_4$, se podem dar dois casos (veja a Figura 3.22) nos quais ou $T \cup \vec{M}(\pi^2)$ forma ciclos de comprimento 2 ou forma um ciclo não dirigido de comprimento 4. O mesmo raciocínio vale também para 3-arestas, ou seja, todos os ciclos definidos por $T \cup \vec{M}(\pi^3)$ não são dirigidos.

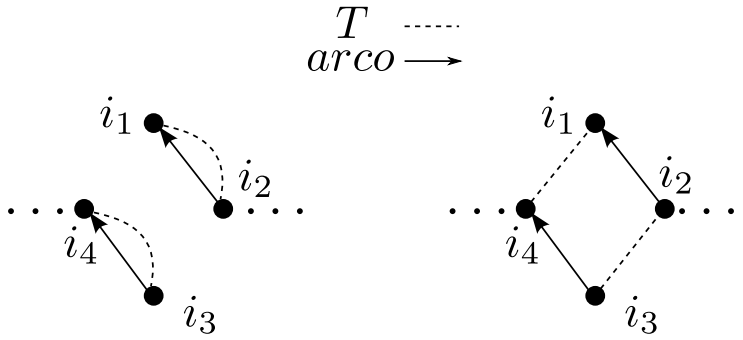

Figura 3.22: Dois possíveis casos de ciclos entre as arestas de T e os arcos das 2-arestas (i_1, i_2) e (i_3, i_4) associadas a um nó $i \in W_4$.

Assim, o único motivo pelo qual uma permutação τ não seria uma solução ótima ao problema da mediana para reversões é que $T \cup \vec{M}(\pi^1)$ possa conter ciclos dirigidos. Para resolver este inconveniente é definido um novo grafo de ponto de quebra múltiplo $G'(\mathcal{E})$ a partir de $G(\mathcal{E})$ da seguinte forma.

No começo, o grafo $G'(\mathcal{E})$ é igual ao grafo $G(\mathcal{E})$, os emparelhamentos hamiltonianos associados H' e H também são iguais. Para cada 1-aresta $e = (i, j)$ em $G(\mathcal{E})$ modifique $G'(\mathcal{E})$ e H' dividindo a aresta e da seguinte forma (veja a Figura 3.23). Eliminar a aresta e, e adicionar os seguintes nós x_e, y_e, z_e, e w_e, logo adicionar as 1-arestas (i, x_e), (y_e, z_e), e (w_e, j), finalmente adicionar as 2-arestas e 3-arestas paralelas (x_e, y_e) e (z_e, w_e). Ademais, escolher uma aresta f em H' que não seja do tipo (i_1, i_3) $(i \in W_4)$, e que não compartilhe vértices com a aresta e. Deixe $f = (a, b)$ tal que os nós i e a formam um caminho de i até a, similarmente, nós j e b formam um caminho de b ate j. Logo, eliminar a aresta $f(a, b)$ e adicionar as seguintes arestas ao emparelhamento H', (x_e, w_e), (a, y_e), e y_e, z_e. A Figura 3.23 mostra a definição de $G'(\mathcal{E})$.

Lema 7 ([22])**.** *$G'(\mathcal{E})$ é um grafo de ponto de quebra múltiplo com emparelhamento hamiltoniano H'.*

Demonstração. A prova será por indução. Inicialmente, H' é igual ao emparelhamento hamiltoniano H. Logo, se H' é um emparelhamento hamiltoniano para uma configuração parcial de $G'(\mathcal{E})$, temos que cada divisão da aresta e (1-aresta) garante que que o novo H'

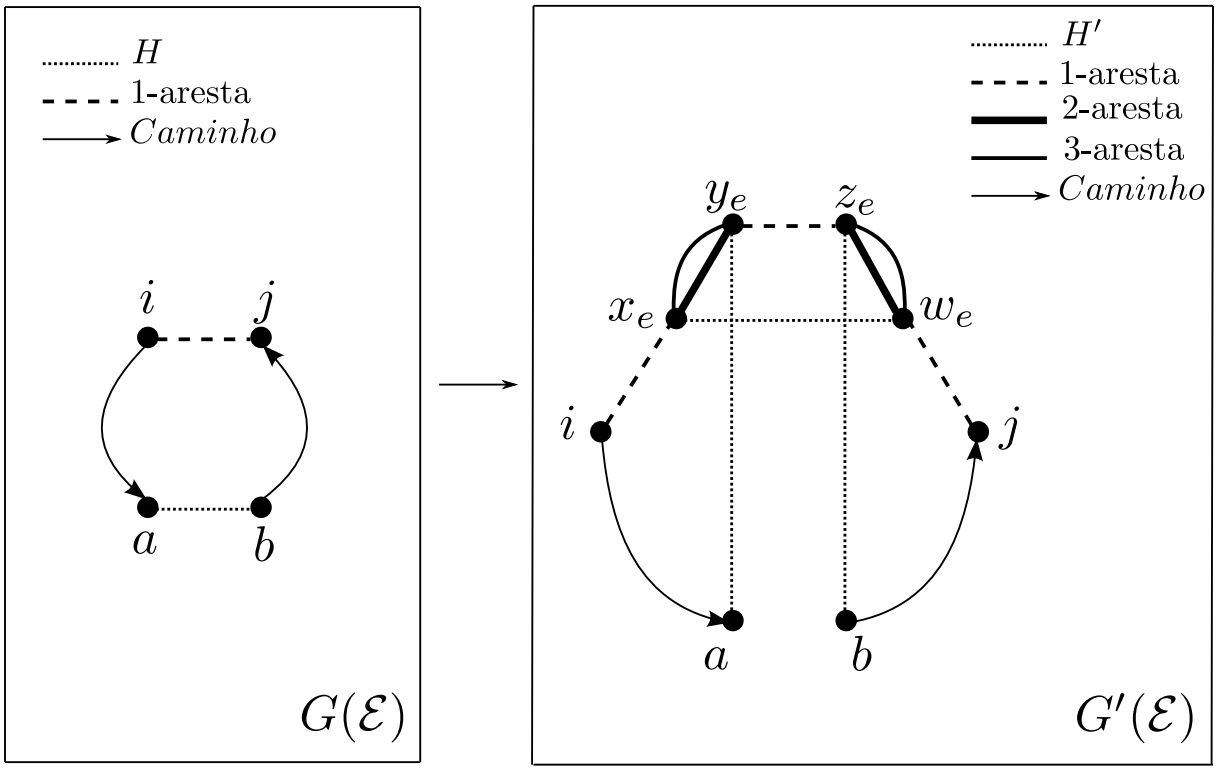

Figura 3.23: Definição de $G'(\mathcal{E})$ a partir de $G(\mathcal{E})$

continua sendo um emparelhamento hamiltoniano do novo grafo $G'(\mathcal{E})$. Isto é, no ciclo hamiltoniano de $G'(\mathcal{E})$ formado por H' e por 1-arestas, a aresta $f = (a, b)$ é substituida pelo caminho (a, y_e), (y_e, z_e), e (z_e, b); e a aresta $e = (i, j)$ é substituída pelo caminho (i, x_e), (x_e, w_e), e (w_e, j). Agora, nos ciclos hamiltonianos de $G'(\mathcal{E})$ formados por H' e 2-arestas, e por H' e 3-arestas, a aresta $f = (a, b)$ é substituida pelo caminho (a, y_e), (y_e, x_e), (x_e, w_e), (w_e, z_e), e (z_e, b). □

Podemos considerar as instâncias do problema EPMC e do problema da mediana para reversões associadas com o grafo $G'(\mathcal{E})$. Seja T uma solução ótima ao problema EPMC em $G(\mathcal{E})$ contendo só arestas paralelas a 2-arestas e 3-arestas, e $T' = T \cup \{(x_e, y_e), (z_e, w_e) : e \text{ é uma 1-aresta de } G(\mathcal{E})\}$.

Lema 8 ([22])**.** *T' é uma solução ótima ao problema EPMC em $G'(\mathcal{E})$, cujo valor é igual a $T + 4\tilde{n}$, onde $\tilde{n}$ é o número de 1-arestas de $G(\mathcal{E})$.*

Demonstração. Note que o grafo $G'(\mathcal{E})$ é um grafo de ciclos pequenos, e que a diferença entre os grafos $F(G(\mathcal{E}))$ e $F(G'(\mathcal{E}))$ é que as arestas azuis e vermelhas de $F(G(\mathcal{E}))$ se correspondem com caminhos de arestas alternadas de comprimento 3 com nós intermédios de grau 2 em $F(G'(\mathcal{E}))$. Logo, o valor da solução ótima para o problema de decomposição maximal de ciclos alternados coincide nestes dois grafos. O anterior juntamente com o

Lema 2 implica que T' é uma solução ótima ao problema EPMC em $G'(\mathcal{E})$. Ademais, as soluções T e T' definem o mesmo número de ciclos com as 1-arestas, enquanto que T' define o mesmo número de ciclos que T define com 2-arestas e 3-arestas, e 4ñ ciclos adicionais de comprimento 2 com as novas 2-arestas e 3-arestas que foram criadas na transformação de $G(\mathcal{E})$ a $G'(\mathcal{E})$. □

Sejam π^1, π^2, π^3 as permutações associadas às 1-arestas, 2-arestas, e 3-arestas do grafo $G'(\mathcal{E})$.

Lema 9 ([22]). *Todos os ciclos definidos por $T' \cup \vec{M}(\pi^k)$, $k \in \{1, 2, 3\}$, são não dirigidos.*

Demonstração. Da discussão sobre a orientação dos ciclos formados por T e as 2-arestas e 3-arestas em $G(\mathcal{E})$, e devido a que todas as arestas da forma (i_1, i_3) em H também estão presentes em H', temos que todos os ciclos definidos por $T' \cup \vec{M}(\pi^2)$ e $T' \cup \vec{M}(\pi^3)$ não são dirigidos. Logo, cada ciclo definido por $T' \cup \vec{M}(\pi^1)$ contém as seguintes arestas (i, x_e), (x_e, y_e), (y_e, z_e), (z_e, w_e), (w_e, j), devido ao Lema 6, e a estrutura resultante de H' (depois de dividir a aresta e) temos que os arcos correspondentes as 1-arestas são orientados em $\vec{M}(\pi^1)$ da seguinte forma, ou $i \to x_e$, $z_e \to y_e$, e $w_e \to j$, ou ao contrário $x_e \to i$, $y_e \to z_e$, e $j \to w_e$ (veja a Figura 3.24).

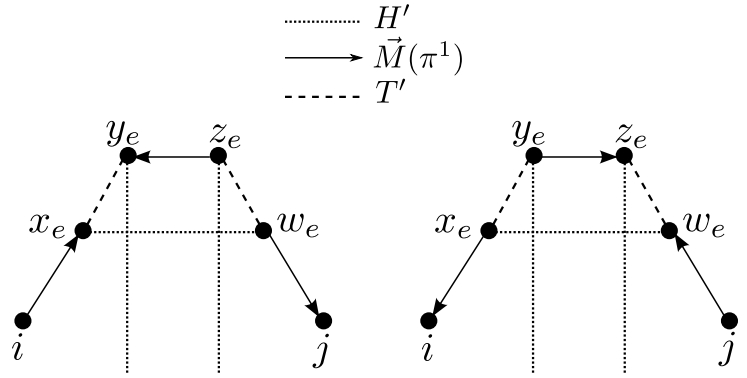

Figura 3.24: Dois possíveis casos de orientação dos arcos de $\vec{M}(\pi^1)$

Esta orientação das 1-arestas é sempre mantida mesmo dividindo outras 1-arestas, já que cada aresta é substituída por caminhos no ciclo hamiltoniano definido por $H' \cup \vec{M}(\pi^1)$. Portanto, todos os ciclos definidos por $T' \cup \vec{M}(\pi^1)$ não são dirigidos (veja a Figura 3.24). □

Teorema 5 ([22]). *O problema da mediana para reversões é $\mathcal{NP}$-Difícil, inclusive para 3 permutações.*

Demonstração. Pelo Teorema 4 e pelos Lemas 5 e 9, temos que o valor de uma solução ótima a uma instância do problema da mediana para reversões definido pelas permutações

π^1, π^2, π^3 produz o valor de uma solução ótima ao problema EPMC em $G'(E)$, portanto, pelos Lemas 2 e 8, o valor de uma solução ótima para o problema de decomposição maximal de ciclos alternados. □

3.5 Revisão da Literatura

Um dos métodos mais usados para solucionar o problema da pequena filogenia usando dados baseados na ordem dos genes é o método presentado por Blanchette et al. [17, 18], o qual foi adaptado a partir do método iterativo proposto em [71]. Neste método o custo de uma árvore é otimizado (minimizado) solucionado iterativamente o problema da mediana de três nós (genomas) até que a árvore não possa ser mais melhorada (estado de convergência). O problema da mediana para *breakpoints* é reduzido a uma instância do problema do caixeiro viajante (PCV), assim o primeiro problema foi solucionado usando um algoritmo exato para PCV. Esta abordagem foi implementada em um software chamado `BPAnalysis` para solucionar o problema da grande filogenia, explorando o espaço de busca de forma exaustiva. Este fato faz deste método extremamente lento para grandes conjuntos de dados (como o dataset *Campanulaceae*) como foi relatado em [58].

Moret et al. [58] fizeram uma reimplementação do software `BPAnalysis` para melhorar o tempo de execução, o novo software foi chamado de `GRAPPA`. Este novo software explora uma amostra do espaço de busca, também usa uma técnica chamada de condensação de genomas para reduzir o comprimento dos genomas. Adicionalmente, foram implementados algoritmos gulosos e exatos para PCV e foi relatado que são mais rápidos que aqueles usados por `BPAnalysis`.

O algoritmo linear para o cálculo da distância de reversão (permutações com sinal) [8] foi incluído na análise executada pelo software `GRAPPA`. Experimentos foram executados usando o dataset *Campanulaceae* e retornaram 216 árvores com um escore igual a 67 reversões. Logo, Moret et al. [56], incluíram em `GRAPPA` o algoritmo para solucionar o problema da mediana para reversões, o qual foi proposto por Caprara [23]. Experimentos foram realizados usando o dataset *Campanulaceae*, e retornaram muitas árvores com um escore de 64 reversões. Os experimentos foram executados em uma única estação de trabalho (*workstation*) e demoraram somente uma hora em finalizar [56].

Bourque e Pevzner [19] propuseram uma abordagem diferente para solucionar o problema da mediana para reversões de três genomas (digamos G_1, G_2, G_3). Neste método é usada uma heurística que consiste em aplicar **boas reversões** sobre um genoma G_1, ou seja, reversões que levam G_1 a ficar mais perto aos outros dois genomas G_2 e G_3. Esta heurística é aplicada também aos genomas G_2 e G_3 até que convergir a um ancestral comum. Logo, para construir uma árvore filogenética, um novo genoma é adicionado ite-

rativamente em alguma aresta da árvore parcial sendo construída. Esta aresta é aquela que forma uma instância do problema da mediana com o novo genoma sendo inserido e que leva a um escore mínimo. Este algoritmo foi implementado no software chamado `MGR` para solucionar o problema da grande filogenia. Experimentos foram realizados com o dataset *Campanulaceae* e retornaram uma árvore com um escore de 65 reversões.

Larget et al. [53] propuseram um abordagem bayesiana a qual foi implementada em um software chamado `BADGER` com o qual foram encontradas 180 árvores diferentes com um escore de 64 reversões para o dataset *Campanulaceae*.

Adam e Sankoff [1] implementaram um algoritmo para o problema da pequena filogenia, o qual iterativamente soluciona o problema da mediana, mas usando como métrica a distância DCJ. Esta distância foi proposta inicialmente por Yancopoulos et al. [91] e redefinida por Bergeron et al. [12]. A heurística usada em [1] para solucionar o problema da mediana para operações DCJ é similar àquela usada no software `MGR`, ou seja, para 3 genomas que formam uma instância do problema da mediana, a heurística consiste em levar um genoma mais perto dos outros dois genomas. Experimentos foram realizados tomando como entrada o dataset *Campanulaceae* e a árvore encontrada pelo software `MGR` [19]. Como resultado destes experimentos foram encontrados os seguintes escores: 59 operações DCJ e 64 operações DCJ com restrições. As restrições para o dataset *Campanulaceae* consistem em considerar somente genomas com um cromossomo circular nos nós internos.

Até este ponto, o problema da mediana foi o problema mais importante a ser resolvido para poder solucionar o problema da pequena filogenia, foi nesse sentido que os esforços foram dirigidos a resolver o problema da mediana. Xu e Sankoff [89] propuseram um algoritmo exato *branch-and-bound* para solucionar o problema da mediana para operações DCJ, o qual foi implementado em um software chamado `ASMedian`. Depois, Gao et al. [41] propuseram um algoritmo genético para o problema da mediana para operações DCJ. Experimentos mostraram que esta abordagem retorna resultados competitivos com respeito a `ASMedian`, e também melhora o tempo de execução quando usadas instâncias do problema difíceis de ser resolvidas.

Kováč et al. [51] propuseram uma abordagem diferente para solucionar o problema da pequena filogenia, a qual foi implementada em um software chamado `PIVO`. Esta abordagem consiste na geração de candidatos por cada nó interno. Cada candidato é gerado aplicando uma operação DCJ sobre o genoma que está rotulando um nó interno. O seguinte passo desta abordagem é calcular o escore dos candidatos usando um algoritmo de programação dinâmica, depois os candidatos com os melhores escores são escolhidos para rotular os nós internos. Este processo é repetido até que o escore total da árvore não possa ser mais melhorado. Experimentos foram realizados usando como entrada o

dataset *Campanulaceae* e a estrutura da árvore encontrada pelo software `MGR` [19]. Como resultado destes experimentos foi encontrado um escore de 59 operações DCJ, e um escore de 64 operações DCJ para o caso com restrições. Um experimento adicional foi realizado tomando como entrada o dataset *Hemiascomycetes* e a estrutura da árvore encontrada usando o software `MrBayes` [68], como resultado deste experimento foi encontrado um escore de 78 operações DCJ com restrições. As restrições para o dataset *Hemiascomycetes* consistem em considerar somente genomas com um cromossomo circular ou genomas com um ou mais cromossomos lineares nos nós internos.

Mais tarde, Herencsár e Brejová [48] propuseram outras heurísticas, como a busca tabu, para melhorar o software `PIVO`. Estas melhoras foram implementadas no software `PIVO2`. Experimentos mostraram que o software `PIVO2` retornou melhores escores com respeito a `PIVO` tomando como entrada os datasets *Campanulaceae* e *Hemiascomycetes*, estes resultados aparecem resumidos na Tabela 5.1, onde são comparados com aqueles obtidos no presente trabalho.

Capítulo 4

Algoritmos Evolutivos para o Cálculo da Distância da Reversão

4.1 Algoritmo Genéticos

Em 1975, Holland desenvolveu os primeiros Algoritmos Genéticos (AG) em seu livro "Adaptation in natural and artificial systems", nos quais foram aplicados os princípios da evolução natural para resolver problemas de otimização [73].

Auyeung e Abraham [5] propuseram o primeiro AG para tratar o problema OPSSR. Em esta abordagem o espaço de busca é formado por 2^n versões com sinal da permutação sem sinal que precisa ser ordenada pelo AG. Por exemplo: Se a permutação sem sinal que vai ser ordenada é $\pi = 3, 1, 2$ então o espaço de busca estaria composto pela família de oito permutações com sinal $\{\pm 3, \pm 1, \pm 2\}$, onde $\pm$ representa ou o sinal $+$ ou $-$. Note que as permutações $\overrightarrow{\pi}' = -3, +1, -2$ e $\overrightarrow{\pi}'' = +3, -1, -2$ representam dois indivíduos diferentes, e em geral os indivíduos no espaço de busca são diferentes somente nos sinais e não nos elementos.

Esta abordagem é baseada principalmente em duas observações:

- Uma sequência de reversões para qualquer das versões com sinal de uma determinada permutação sem sinal π é também uma sequência válida de ordenação para a permutação π.
- Uma das sequências de ordenação por reversões de uma das versões com sinal de uma determinada permutação sem sinal π é uma sequência de ordenação ótima para a permutação π.

Baseada nas observações anteriores a função de aptidão de um indivíduo (permutação com sinal) é definida como a distância de reversão para o problema OPCSR, para o qual existe um algoritmo de tempo linear ([7]). Assim, o algoritmo genético buscará a

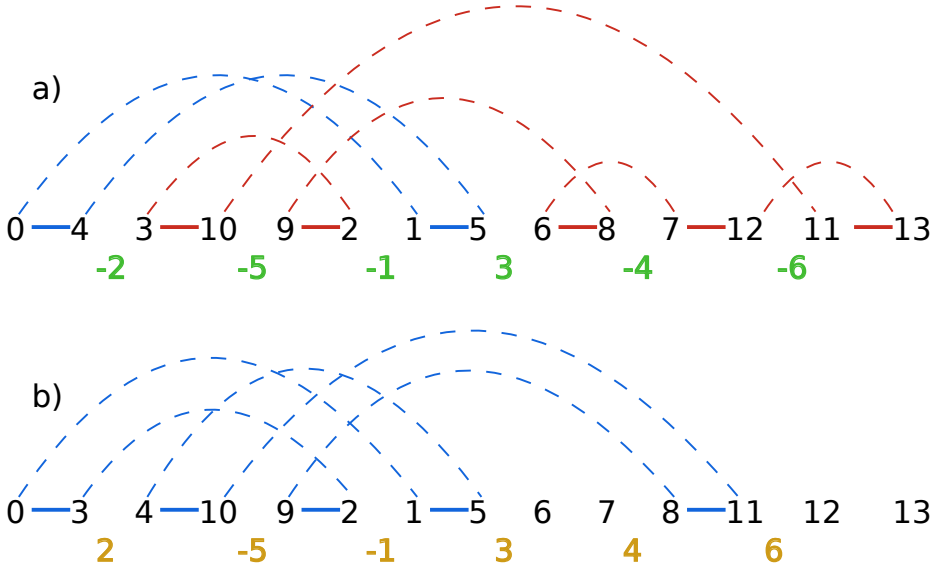

Figura 4.1: Duas permutações com sinal $\overrightarrow{\pi}^a$ e $\overrightarrow{\pi}^b$, e seus correspondentes grafos de ponto de quebra e decomposições em ciclos.

menor distância de reversão no espaço de busca composto de permutações com sinal, esta distância no pior caso será um número válido de reversões para ordenar a permutação sem sinal tomada como entrada do AG.

Como exemplo, considere os seguintes dois indivíduos (permutações com sinal): $\overrightarrow{\pi}^a = -2, -5, -1,\ 3, -4, -6$ e $\overrightarrow{\pi}^b = 2, -5, -1,\ 3,\ 4,\ 6$ cuja decomposição em ciclos é mostrada nas Figuras 4.1 a) e b), respectivamente. Note que $b(\overrightarrow{\pi}^a) = 7$, $c(\overrightarrow{\pi}^a) = 2$, $b(\overrightarrow{\pi}^b) = 5$ e $c(\overrightarrow{\pi}^b) = 1$. Neste caso, a distância de reversão é exatamente $b(\pi) - c(\pi)$ [1], quer dizer, $d(\overrightarrow{\pi}^a) = 5$ e $d(\overrightarrow{\pi}^b) = 4$; estes valores representam a aptidão dos indivíduos $\overrightarrow{\pi}^a$ e $\overrightarrow{\pi}^b$.

A seleção dos melhores indivíduos para a etapa de crossover é feita ordenando todos os indivíduos por seu valor de aptidão. O operador de crossover é aplicado sobre dois individuos usando um único ponto, quer dizer, um ponto é escolhido aleatoriamente e os elementos depois desse ponto são trocados entre dois indivíduos. Por exemplo, as Figuras 4.2 a) e b) mostram a descendência resultante $\overrightarrow{\sigma}^a$ e $\overrightarrow{\sigma}^b$, respectivamente, depois de aplicar o operador de crossover sobre as permutações $\overrightarrow{\pi}^a$ e $\overrightarrow{\pi}^b$, onde o ponto de crossover está entre os elementos 1 e 3. A distância de reversão do primeiro descendente é $d(\overrightarrow{\sigma}^a) = 3$. Este novo indivíduo tem grandes chances de ser considerado na próxima geração do AG uma vez que o valor de aptidão foi melhorado. A distância de reversão do segundo descendente é $d(\overrightarrow{\sigma}^b) = 6$; assim, este indivíduo será ignorado na próxima geração

[1] Está informação é consistente com o limite inferior $d(\pi) \geq b(\pi) - c(\pi)$ encontrado por Bafna e Pevzner [9]. Para o cálculo da distância de reversão de permutações com sinal existe um relação mais exata $d(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi)$, encontrada por Hannenhalli e Pevzner, onde $h(\pi) \geq 0$ e $f(\pi) \in \{0, 1\}$ representam um obstaculo (*hurdle*) e uma fortaleza (*fortresses*) as quais são noções que indicam se uma permutação é difícil de ser ordenada.

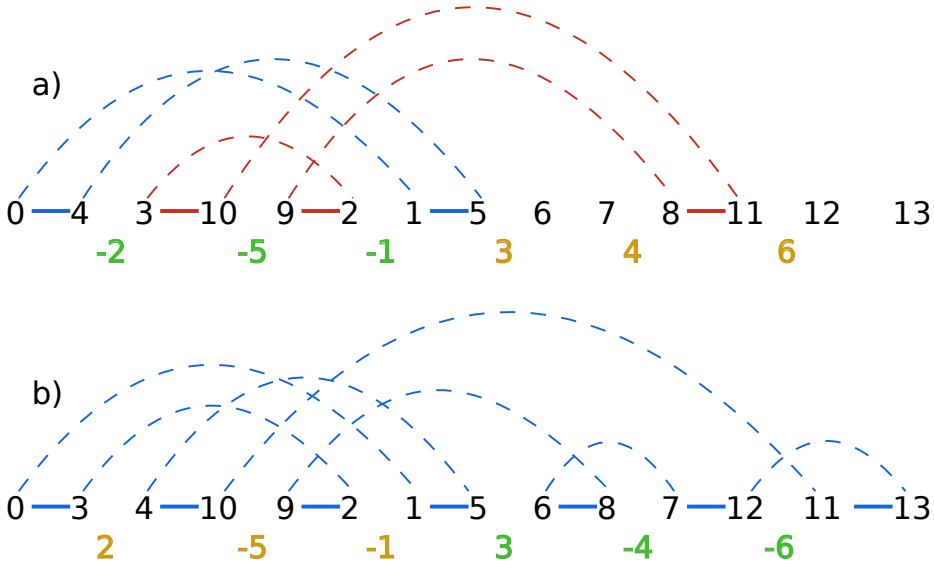

Figura 4.2: Descendência resultante ($\overrightarrow{\sigma}^a$ e $\overrightarrow{\sigma}^b$) depois de aplicar o operador de crossover sobre as permutações $\overrightarrow{\pi}^a$ e $\overrightarrow{\pi}^b$ da Figura 4.1, onde o ponto de crossover está entre os elementos 1 e 3.

do AG.

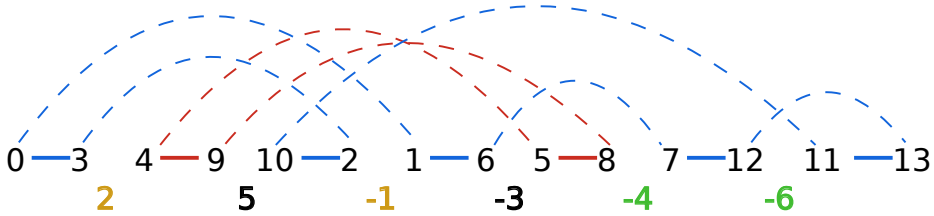

Figura 4.3: Indivíduo resultante ($\overrightarrow{\sigma}^c$) depois de aplicar o operador de mutação sobre a permutação $\overrightarrow{\sigma}^b$ da Figura 4.2, onde os elementos modificados são 5 e 3.

A mutação é realizada modificando o sinal de um elemento de positivo a negativo ou vice-versa. Este processo é aplicado para todos os elementos de um indivíduo com uma determinada probabilidade. Por exemplo, a Figura 4.3 mostra o indivíduo resultante $\overrightarrow{\sigma}^c$ depois de aplicar o operador de mutação sobre o indivíduo $\overrightarrow{\sigma}^b = 2, -5, -1,\ 3, -4, -6$, onde os elementos 5 e 3 são aqueles com um sinal modificado. Neste caso, a distância de reversão para $\overrightarrow{\sigma}^c$ é 5, a qual melhora o valor de aptidão de $\overrightarrow{\sigma}^b$ ($d(\overrightarrow{\sigma}^b) = 6$).

Seguindo a abordagem de Auyeung e Abraham (AG-Au), um AG padrão foi implementado em [76] (AG-SA), mas em vez de usar o algoritmo de Kaplan et al. (com complexidade $O(n^2)$) para o cálculo da aptidão, AG-SA usa o algoritmo de tempo linear de Bader et al; adicionalmente, para ordenar a população na etapa de seleção foi usado o algoritmo *counting sort*. Nos experimentos AG-SA será usada como a implementação predefinida do AG padrão. O Algoritmo 4 mostra o pseudocódigo do AG padrão para o problema OPSSR.

Algoritmo 4: AG Padrão para o Problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
1 Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
2 Avaliar a aptidão dos indivíduos da população inicial;
3 para cada i = 2 até numeroGerações faça
4     Seleção dos melhores indivíduos;
5     Aplicar operador de Crossover;
6     Aplicar operador de Mutação;
7     Avaliar a aptidão da descendência;
8     Substituição dos piores indivíduos;
```

Também, uma melhoria sobre AG-SA foi proposta em [76] que inclui uma etapa de pré-processamento na qual 2-reversões são aplicadas até que não seja mais possível eliminar dois pontos de quebra. Esta heurística de eliminação de 2-reversões foi aplicada por muitos algoritmos de aproximação como [9], e [27]. O Algoritmo 5 mostra o pseudocódigo do AG-Híbrido.

Algoritmo 5: AG-Híbrido para o Problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
1 Aplicar 2-reversões sobre π até que este tipo de operação não possa ser mais
    aplicada, depois atualizar π;
2 Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
3 Avaliar a aptidão dos indivíduos da população inicial;
4 para cada i = 2 até numeroGerações faça
5     Seleção dos melhores indivíduos;
6     Aplicar operador de Crossover;
7     Aplicar operador de Mutação;
8     Avaliar a aptidão da descendência;
9     Substituição dos piores indivíduos;
```

Os parâmetros usados por AG-SA e AG-Híbrido são os mesmos, os quais são apresentados a seguir:

- *Probabilidade crossover* que descreve com que frequência a operação de crossover será aplicada sobre dois indivíduos pais. Este parâmetro é usado na etapa das linhas 5 e 6 dos Algoritmos 4 e 5, respectivamente.
- *Probabilidade de mutação* que descreve com que frequência o operador de mutação será aplicado sobre cada elemento de um indivíduo. Este parâmetro é usado nas etapas das linhas 6 e 7 dos Algoritmos 4 e 5, respectivamente.

- *Número de pontos de crossover* que indica quantos pontos serão usados para a operação de crossover. Este parâmetro é usado na etapa das linhas 5 e 6 dos Algoritmos 4 e 5, respectivamente.
- *Porcentagem de seleção* que indica quantos dos melhores indivíduos serão selecionados para o crossover e a mutação. Este parâmetro é usado nas etapas das linhas 4 e 5 dos Algoritmo 4 e 5, respectivamente.
- *Porcentagem de substituição* que indica quantos dos piores indivíduos serão substituídos por uma nova descendência. Este parâmetro é usado nas etapas das linhas 8 e 9 dos Algoritmos 4 e 5, respectivamente.

4.2 Algoritmos Genéticos Aprimorados por Heurísticas

4.2.1 Algoritmos Meméticos

Algoritmos Meméticos (AM) são técnicas de otimização que combinam algoritmos evolutivos com uma o mais fases de busca local, e até podem incluir métodos exatos e algoritmos de aproximação ([59, 52]); nesse sentido, os AM mostraram potencial aplicabilidade para tratar problemas de otimização discreta ([47]). A aplicação de busca local sobre um indivíduo envolve a geração de soluções vizinhas promissórias; este processo é repetido um determinado número de vezes. Esta técnica vai acelerar o descobrimento de novas boas soluções e portanto vai melhorar a população inteira ([60, 52]).

É importante salientar que os algoritmos meméticos podem ser considerados como um conjunto de um domínio maior conhecido como Computação Memética (CM) ([25, 61]). Esta disciplina estuda estruturas algorítmicas compostas por muitos operadores, chamados de *memes*, que interagem e evoluem para solucionar problemas de otimização. Um projeto automático de estruturas de CM, para problemas de otimização continua, foi proposto por Caraffini et al. [24], onde a seleção dos operadores é feita baseada na análise das características do problema sendo tratado.

Neste trabalho, entre outras coisas, estamos propondo um algoritmo memético para o problema OPSSR. Este algoritmo é baseado no algoritmo AG-SA, no qual foi embebido um procedimento de busca local. O problema OPSSR pode ser visto como um problema de otimização discreta, já que lidamos com um espaço de busca formado de permutações com sinal. O procedimento de busca local proposto é baseado na mutação de uma posição aleatória de uma permutação com sinal, quer dizer, na troca do sinal de um elemento. O Algoritmo 6 mostra o pseudocódigo da busca local.

Algoritmo 6: Busca Local sobre uma permutação com sinal

Entrada: Uma permutação com sinal $\overrightarrow{\pi}$

```
1  melhorAptidão = Calcular a aptidão da permutação π⃗;
2  para i = 1 até numeroIterações faça
3      Gerar uma posição aleatória j em π⃗;
4      Modificar o sinal do elemento na posição j;
5      aptidão = Calcular a aptidão de π⃗;
6      se aptidão < melhorAptidão então
7          Atualizar o novo valor de aptidão de π⃗;
8          break;
9      senão
10         Recuperar estado anterior de π⃗;
```

De acordo com a classificação proposta em [36], a busca local (mostrada no Algoritmo 6) é classificada como:

- Estocástica, porque a geração de um novo indivíduo de teste é feito em forma aleatória.
- Solução-Única, porque só uma solução é processada.
- Gulosa, porque a substituição é feita assim que uma solução que melhore a solução atual for encontrada.

Este procedimento de busca local foi incluído nas seguintes etapas do AM:

- Geração da população inicial.
- Reinício da população.

Adicionalmente, uma nova etapa de apenas busca local foi incluída depois do ciclo de reprodução do AM. Neste algoritmo, a população é reiniciada em caso de que atinga um estado degenerado (quer dizer, indivíduos com alta similaridade). Como medida de similaridade foi usada a **entropia de Shannon** como definida em [72]. O Algoritmo 7 mostra o pseudocódigo do AM.

O AM para o problema OPSSR compartilha os mesmos parâmetros usados por AG-SA e AG-Híbrido, ademais inclui os seguintes novos parâmetros:

- *Porcentagem de busca local*, que indica quantos dos melhores indivíduos serão selecionados para aplicar o operador de buscar local. Este parâmetro é usado na etapa da linha 10 do Algoritmo 7.

Algoritmo 7: AM para o problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
1  Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
2  Avaliar a aptidão dos indivíduos da população inicial;
3  Melhorar os indivíduos aplicando o procedimento de busca local (Alg.6);
4  para i = 2 até numeroGerações faça
5      Seleção dos melhores indivíduos;
6      Aplicar operador de Crossover;
7      Aplicar operador de Mutação;
8      Avaliar a aptidão da descendência;
9      Substituição dos piores indivíduos;
10     Aplicar busca local à população atual;
11     se parâmetro limiteEntropiaMinimo é atingido então
12         Reiniciar a população;
13         Melhorar os indivíduos aplicando busca local;
```

- *Número de passos de busca local*, que indica o número de iterações do procedimento de busca local aplicado a uma permutação. Este parâmetro é usado na linha 2 do Algoritmo 6.
- *Porcentagem de preservação*, que indica quantos dos melhores indivíduos serão preservados. A parte restante da população será reiniciada. Este parâmetro é usado na etapa da linha 12 do Algoritmo 7.
- *Limite de entropia mínima*, que indica o mínimo valor da entropia de Shannon que a população pode alcançar. Este parâmetro é usado na linha 11 do Algoritmo 7.

4.2.2 Algoritmos Meméticos e Busca por Oposição

A ideia intuitiva por trás da busca por oposição (*Opposition Based Learning* (OBL)) é que no pior caso de um processo de busca um indivíduo estaria no lado oposto de uma solução ótima no espaço de busca, assim, se aplicássemos o conceito de busca por oposição poderíamos obter bons resultados sempre que os indivíduos estivessem no lado oposto de uma solução ótima, ou com uma valor de aptidão ruim. Tizhoosh e Ventesca definiram o conceito de número oposto e o conceito de pontos opostos do tipo-I e tipo-II ([86], [2]).

- Seja x um número real no intervalo $[a, b]$. O **número oposto** $\breve{x}$ é definido da seguinte forma:

$$\breve{x} = a + b - x. \tag{4.1}$$

- Seja $P = (x_1, x_2, \ldots, x_N)$ um ponto N-dimensional com $x_i \in \Re$ e $x_i \in [a_i, b_i]$, então um **ponto oposto do tipo-I** é definido por $\breve{P} = (\breve{x_1}, \breve{x_2}, \ldots, \breve{x_N})$, onde

$$\breve{x_i} = a_i + b_i - x_i \qquad i = 1, 2, \ldots, N. \tag{4.2}$$

- Seja f uma função arbitrária $\mathbb{R}^n \to \mathbb{R}$ com imagem no intervalo $[y_{min}, y_{max}]$. Para cada ponto N-dimensional $P = (x_1, \ldots, x_n)$, um **ponto oposto do tipo-II** é definido por

$$\breve{f}(x_1, \ldots, x_n) = y_{min} + y_{max} - f(x_1, \ldots, x_n). \tag{4.3}$$

No contexto de algoritmos de otimização enquanto a busca por oposição do tipo-I é baseada nos pontos opostos no espaço de busca, a busca por oposição do tipo-II corresponde ao oposto do valor de aptidão. Também, é importante salientar que a oposição do tipo-I requer conhecer o espaço de busca para poder aplicar a definição de oposição, enquanto que a oposição do tipo-II requer conhecer primeiro a função de aptidão ([69]). Motivado por estas razões e por simplicidade, neste trabalho usamos o operador de oposição do tipo-I.

Por exemplo, a Figura 4.4 mostra a aplicação do operador de oposição do tipo-I sobre uma permutação com sinal $\overrightarrow{\pi^a} = -2, -5, \ 1, -3, -4, -6$. A permutação resultante depois de aplicar este operador é $\overrightarrow{\pi^b} = 2, \ 5, -1, \ 3, \ 4, \ 6$, onde todos os seus elementos tem seus sinais trocados de acordo com a definição de número oposto. Onde o intervalo é $[0, 1]$, o sinal negativo é representado por 0, e o sinal positivo é representado por 1, assim o oposto de um número positivo é um número negativo e vice-versa. Também, neste caso o valor de aptidão (distância de reversão) foi melhorada de $d(\overrightarrow{\pi^a}) = 5$ para $d(\overrightarrow{\pi^b}) = 4$.

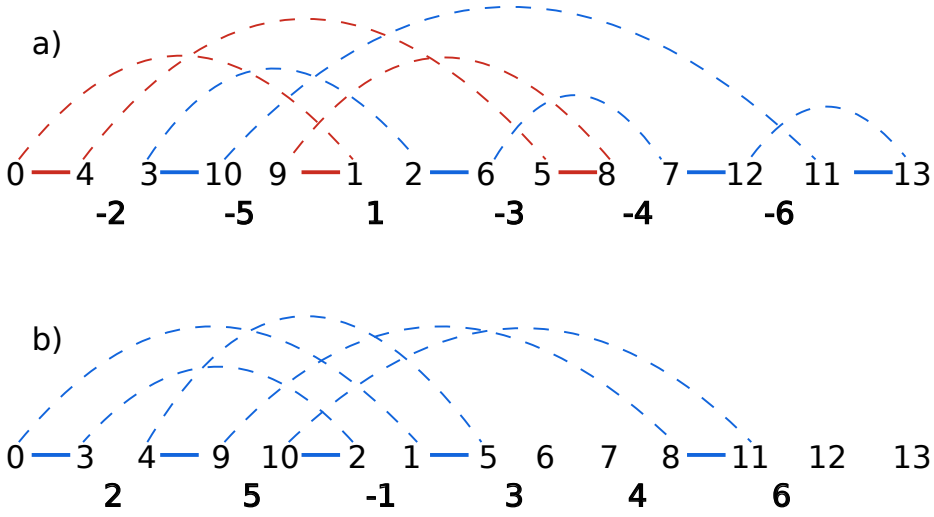

Figura 4.4: Uma permutação com sinal $\overrightarrow{\pi^a}$ e sua permutação com sinal oposta $\overrightarrow{\pi^b}$.

Inicialmente, OBL foi proposto em [85] para estender os algoritmos genéticos, a aprendizagem reforçada (*reinforcement learning*), e as redes neurais. Depois, OBL foi aplicada em muitos algoritmos de otimização ([90]) como a evolução diferencial (*differential evolution - DE*), otimização por enxame de partículas, otimização baseada em biogeografia, busca de harmonia (*harmony search*), sistema de colônia de formigas (*ant colony system*), colônia de abelhas artificiais (*artificial bee colony*). Destes algoritmos, a DE é a que tem maior predomínio, talvez pelos resultados obtidos no trabalho clássico proposto por Rahnamayan et al. [63], onde a DE foi melhorada mediante a inclusão de OBL na etapa da geração da população inicial e na etapa de busca de novas soluções (*generation jumping*).

O operador de OBL foi aplicado dentro do AM (Alg. 7) para melhorar os indivíduos da população inicial e também depois de reiniciar a população. OBL foi combinado com a busca local como explicado a seguir:

- primeiramente, OBL é aplicado a uma permutação com sinal $\overrightarrow{\pi}$;
- se o valor de aptidão da permutação com sinal $\overrightarrow{\pi}$ é melhorado então $\overrightarrow{\pi}$ é substituído pela permutação gerada por OBL;
- caso contrário o procedimento de busca local é aplicado sobre a permutação $\overrightarrow{\pi}$.

Algoritmo 8: OBL sobre uma permutação com sinal

Entrada: Uma permutação com sinal $\overrightarrow{\pi}$

```
1 melhorAptidão = Calcular a aptidão de π⃗;
2 para i = 1 até numeroElementos de π⃗ faça
3     Modificar o sinal do elemento na posição i;
4 aptidão = Calcular a aptidão de π⃗;
5 se aptidão < melhorAptidão então
6     Atualizar a nova aptidão de π⃗;
7 senão
8     Recuperar estado anterior de π⃗;
```

O Algoritmo 8 mostra a aplicação de OBL sobre uma permutação com sinal. O algoritmo que combina OBL com AM é chamado de Algoritmo Memético com Busca por Oposição (AMBO), pode ser visto que este algoritmo aproveita a busca local para explorar novas soluções e a busca por oposição para descobrir novas soluções no espaço de busca. O Algoritmo 9 mostra o pseudocódigo do AMBO.

O algoritmo AMBO compartilha os mesmos parâmetros usados por AG-SA, AG-Híbrido, e AM, mas também inclui um novo parâmetro:

Algoritmo 9: AMBO para o problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
1  Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
2  Avaliar a aptidão dos indivíduos da população inicial;
3  Melhorar os indivíduos aplicando os procedimentos de OBL (Alg. 8) e de busca
     local (Alg.6);
4  contador ← 0;
5  para i = 2 até numeroGerações faça
6      Seleção dos melhores indivíduos;
7      Aplicar operador de Crossover;
8      Aplicar operador de Mutação;
9      Avaliar a aptidão da descendência;
10     Substituição dos piores indivíduos;
11     Aplicar busca local à população atual;
12     se a melhor aptidão atual é igual à melhor aptidão anterior então
13         contador ← contador + 1;
14     se (parâmetro limiteEntropiaMinimo é atingido) ou (contador é igual ao
         parâmetro numeroVezesAptidãoInvariante) então
15         contador ← 0;
16         Reiniciar a população;
17         Melhorar os indivíduos aplicando os procedimentos de OBL e de busca
             local;
```

- *Número de vezes aptidão invariante*, que indica o máximo número de gerações que o melhor indivíduo é permitido a ficar invariante antes de reiniciar a população. Este parâmetro é usado na linha 14 do Algoritmo 9.

4.2.3 Algoritmos Meméticos, Busca por Oposição, e Heurística de Eliminação de Pontos de Quebra

Como foi visto em outros trabalhos (ver [76]) o passo de pré-processamento que consiste em aplicar reversões que eliminam simultaneamente dois pontos de quebra, deu bons resultados quando foi aplicado sobre o algoritmo genético padrão (AG-SA); da mesma forma, esta abordagem é aplicada sobre o algoritmo AMBO com o objetivo de melhorar a qualidade dos resultados. O Algoritmo 10 mostra o pseudocódigo do AMBO-Híbrido, o qual compartilha os mesmos parâmetros que AMBO.

Algoritmo 10: AMBO-Híbrido para o problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
1  Aplicar 2-reversões sobre π até que este tipo de operação não possa ser mais
     aplicada, depois atualizar π;
2  Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
3  Avaliar a aptidão dos indivíduos da população inicial;
4  Melhorar os indivíduos aplicando os procedimentos de OBL (Alg. 8) e de busca
     local (Alg.6);
5  contador ← 0;
6  para i = 2 até numeroGerações faça
7      Seleção dos melhores indivíduos;
8      Aplicar operador de Crossover;
9      Aplicar operador de Mutação;
10     Avaliar a aptidão da descendência;
11     Substituição dos piores indivíduos;
12     Aplicar busca local à população atual;
13     se a melhor aptidão atual é igual à melhor aptidão anterior então
14         contador ← contador + 1;
15     se (parâmetro limiteEntropiaMinimo é atingido) ou (contador é igual ao
         parâmetro numeroVezesAptidãoInvariante) então
16         contador ← 0;
17         Reiniciar a população;
18         Melhorar os indivíduos aplicando os procedimentos de OBL e de busca
             local;
```

4.3 Algoritmos Genéticos Paralelos

Neste trabalho também propomos dois algoritmos paralelos baseados no AG apresentado no Algoritmo 4 (AG-SA), os quais já foram apresentados nas seguintes referências [79, 74].

De acordo com a classificação proposta por Cantú-Paz [20] os modelos dos algoritmos paralelos deste trabalho são: AG Mestre-Escravo de População-Única e AG Múltiplas-Populações. Ademais, de acordo com a classificação proposta por Sudholt [80] o segundo algoritmo paralelo se corresponde com o modelo de Execuções Independentes (*Independent Runs Model*), e neste trabalho será chamado de AG-Paralelo.

Como já foi mencionado no anterior paragrafo o primeiro algoritmo paralelo usa o modelo Mestre-Escravo, no qual um processo mestre mantém uma única população e executa as etapas de reprodução do AG (seleção, crossover, mutação, e substituição). A avaliação da função de aptidão é realizada em paralelo pelos processos escravos. O pseudocódigo deste primeiro algoritmo é apresentado no Algoritmo 11.

Algoritmo 11: AG com cálculo em paralelo da função de aptidão para o problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π

```
// Para o processo mestre:
Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
Avaliar a aptidão dos indivíduos da população inicial em paralelo (Alg. 12);
para cada i = 2 até numeroGerações faça
    Seleção dos melhores indivíduos;
    Aplicar operador de Crossover;
    Aplicar operador de Mutação;
    Avaliar a aptidão da descendência em paralelo (Alg. 12);
    Substituição dos piores indivíduos;
// Para o processo escravo:
enquanto True faça
    Receber uma permutação do processo mestre;
    Avaliar a aptidão;
    Enviar o resultado ao processo mestre;
```

Algoritmo 12: Avaliar a aptidão em paralelo

```
para todos os processos escravos faça
    Enviar a um processo escravo uma permutação com sinal da população;
enquanto exista um indivíduo sem aptidão avaliada faça
    Receber resultado de um processo escravo;
    Enviar para um processo escravo uma permutação com sinal da população;
para todos os processos escravos faça
    Receber resultado de um processo escravo;
```

O segundo algoritmo paralelo usa o modelo de Múltiplas-Populações, onde cada processo escravo mantém a sua própria população e executa uma instância do AG, ou seja, todo o processo de reprodução. O processo mestre recebe os melhores valores de aptidão encontrados pelos processos escravos em cada etapa do AG, e depois encontra a melhor aptidão. O pseudocódigo desta abordagem que usa múltiplas populações (AG-Paralelo), é mostrado no Algoritmo 13.

Dos experimentos realizados nas referências [79, 74] temos que o primeiro algoritmo paralelo em efeito melhorou o tempo de execução do AG-SA e manteve a mesma acurácia (número de reversões); logo, o segundo algoritmo paralelo (AG-Paralelo) mostrou ter os melhores resultados com respeito ao número de reversões, mas usando uma população maior que o AG-SA, ou seja, 23 vezes a população do AG-SA (23*n log n). Este último fato indica uma comparação não justa no sentido que o AG-Paralelo usa mais recursos

Algoritmo 13: AG paralelo com múltiplas populações para o problema OPSSR

Entrada: Uma permutação sem sinal π
Saída: Um número de reversões para ordenar π
`// Para o processo mestre:`
para *cada geração* **faça**
  para *todos os processos escravos* **faça**
    Receber a melhor aptidão de um escravo;
    se *solução* $<$ *melhorAptidão* **então**
      solução = melhorAptidão;
`// Para o processo escravo:`
Inicializar a população inicial gerando aleatoriamente versões com sinal de π;
Avaliar a aptidão dos indivíduos da população inicial;
para cada $i = 2$ *até* numeroGerações **faça**
  Seleção dos melhores indivíduos;
  Aplicar operador de Crossover;
  Aplicar operador de Mutação;
  Avaliar a aptidão da descendência;
  Substituição dos piores indivíduos;
  Enviar ao processo mestre a melhor aptidão encontrada;

que o AG-GA e portanto era esperado ter melhores resultados. Na seção da discussão deste capítulo é mostrada a real contribuição do AG-Paralelo.

4.4 Experimentos e Resultados

Todos os algoritmos propostos neste trabalho foram implementados na linguagem `C` e executados nas seguintes plataformas: `OSX`, com um processador Intel Core I7 operando a 3.4GHz; e `Ubuntu Linux`, com dois processadores Intel Xeon E5-2620 operando a 2.4 GHz. Os algoritmos sequenciais foram executados na primeira plataforma e os paralelos na segunda. O fato de usar duas plataformas diferentes não afeta o resultado dos experimentos já que os algoritmos usam os mesmos recursos computacionais e são comparados pelo número de reversões.

Posteriormente na subseção de discussão um experimento adicional foi realizado para comparar os tempos de execução de AG-SA e AG-Paralelo, neste caso foi usada a mesma plataforma (`Ubuntu Linux`).

O Algoritmo 14 mostra o pseudocódigo do gerador aleatório de permutações com sinal que foi usado no primeiro e segundo experimento.

Os parâmetros de AG-SA (e AG-Híbrido) são os mesmos que foram estabelecidos em [76]. Também os parâmetros para o AM (para OPSSR) são os mesmos estabelecidos em

Algoritmo 14: Gerador aleatório de permutações sem sinal.

Entrada: Comprimento da permutação a ser gerada n
Saída: Permutação aleatória sem sinal π

```
Criar uma permutação vazia π;
Criar uma lista A de números de 1 até n;
para i = 1 to n faça
    Gerar aleatoriamente um inteiro x de 1 até o comprimento da lista A;
    Adicionar em π um elemento na posição x da lista A;
    Eliminar um elemento na posição x da lista A;
```

[77]. Para AMBO (e AMBO-Híbrido) os parâmetros foram estabelecidos na mesma forma que na ultima referência, fazendo uma analise de sensibilidade na seguinte forma:

- Primeiro, cada parâmetro é convertido em um conjunto discreto, quer dizer, $\{0.1; 0.2; \ldots; 0.9\}$.
- Então, o conjunto discreto para cada parâmetro é reduzido executando AMBO de forma exaustiva (cem vezes por cada elemento do conjunto). O conjunto reduzido de um parâmetro representa aqueles valores que têm os melhores resultados em promédio para AMBO.
- Finalmente, os elementos destes conjuntos reduzidos são permutados para encontrar a melhor configuração de parâmetros executando AMBO de forma exaustiva (cem vezes por cada configuração). A configuração de parâmetros que tem o melhor resultado em promédio é escolhida como os ajustes dos parâmetros.

Tabela 4.1: Configuração dos parâmetros para AG-SA, AM, e AMBO.

Parameter	**SA-GA** [2]	**AM** [3]	**AMBO**
Probabilidade crossover	0.90	0.98	0.98
Probabilidade de mutação	0.02	0.01	0.01
Num. de pontos de crossover	1	1	1
% de seleção	0.60	0.96	0.96
% de substituição	0.60	0.60	0.60
% de busca local	-	0.94	0.94
# passos busca local	-	2	3
% de preservação	-	0.98	0.40
Limite de entropia mínima	-	0.20	0.15
# de vezes aptidão invariante	-	-	14

[2]Parâmetros tomados de [76]
[3]Parâmetros tomados de [77]

A Tabela 4.1 mostra todos os ajustes de parâmetros para os algoritmos AG-SA, AG-Híbrido, AM, AMBO, e AMBO-Híbrido. Note que AG-Híbrido e AMBO-Híbrido têm os mesmos parâmetros que AG-SA e AMBO, respectivamente. É importante salientar que o número de passos de busca local para AMBO (e AMBO-Híbrido) foi escolhido para ser 3 baseado numa analise de sensibilidade. Para mais de três passos o decremento do número de reversões não tinha uma diferença significativa com respeito a usar três passos. Assim, três foi a melhor escolha, visto que incrementar o número de passos também incrementa o número de avaliações da função de aptidão.

Três tipos diferentes de entradas foram usadas nos experimentos:

a) Conjuntos de permutações geradas aleatoriamente de diferentes comprimentos, cada conjunto contendo cem permutações sem sinal do mesmo comprimento. O Algoritmo 14 foi usado para gerar estas permutações.

b) Permutações sem sinal simples previamente propostas na literatura como benchmarks.

c) Permutações criadas a partir de dados biológicos.

4.4.1 Experimentos usando Conjuntos de Cem Permutações sem Sinal

Este experimento foi realizado de forma similar a alguns trabalhos da literatura (veja [5, 75, 76, 79, 77]). O número de indivíduos e avaliações da função de aptidão foram fixados para ter uma comparação justa dos algoritmos AG-SA, AG-Híbrido, AM, AMBO, AMBO-Híbrido, e AG-Paralelo. Assumindo que o comprimento da permutação de entrada é n, o tamanho da população foi fixado a $n \log n$ para todos os algoritmos. Para o caso do AG-Paralelo, o tamanho da população foi divido por 23, que é o número de processos escravos. A configuração deste experimento segue a continuação:

- Para cada conjunto de cem permutações (geradas aleatoriamente) de comprimento i, com $i \in \{10,\ 20,\ \ldots,\ 150\}$:
  - Inicialmente, escolher uma permutação de comprimento i, executar AMBO uma única vez por 200 gerações e salvar o número de avaliações da função de aptidão. Este número de avaliações, digamos k, serão usados como critério de parada para todos os algoritmos quando forem usadas permutações de comprimento i.
  - Depois, para cada permutação em um determinado conjunto:

* Executar todos os algoritmos 50 vezes para a permutação atual. Em cada execução, parar os algoritmos quando alcançarem k avaliações da função de aptidão.
* Calcular a média de 50 saídas (número de reversões) para cada algoritmo. Este valor, para um algoritmo, representa o resultado da permutação atual.

– Finalmente, para cada algoritmo calcular a média e o desvio padrão dos resultados de cem permutações de comprimento i.

Tabela 4.2: Média (e desvio padrão) do número de reversões do experimento usando conjuntos de cem permutações sem sinal.

Cmp.	**AG-SA**		**AG-H.**		**AM**		**AMBO**		**AMBO-H.**		**AG-P.**	
	Méd.	D.P.	Méd.	D.P.	Méd.	D.P.	Méd.	D.P.	Méd.	D.P.	Méd.	D.P.
10	5.74	0.861	5.77	0.849	**5.73**	0.863	**5.73**	0.863	5.76	0.851	**5.73**	0.863
20	13.18	1.064	13.22	1.07	13.13	1.054	**13.12**	1.047	13.17	1.055	13.16	1.068
30	20.73	1.105	20.73	1.101	20.59	1.091	**20.55**	1.084	20.59	1.079	20.77	1.084
40	28.6	1.237	28.61	1.232	28.38	1.211	**28.34**	1.207	28.38	1.189	28.75	1.247
50	36.9	1.245	36.89	1.252	36.6	1.214	**36.52**	1.186	36.53	1.175	37.26	1.268
60	45.26	1.444	45.26	1.455	44.91	1.448	44.8	1.429	**44.78**	1.423	45.83	1.377
70	53.55	1.573	53.54	1.575	53.19	1.571	53.07	1.538	**53.03**	1.558	54.34	1.514
80	62.05	1.444	62.04	1.454	61.74	1.432	61.59	1.428	**61.55**	1.425	63.07	1.377
90	70.6	1.577	70.56	1.554	70.32	1.593	70.16	1.585	**70.1**	1.583	71.79	1.509
100	78.71	1.713	78.66	1.734	78.57	1.74	78.4	1.696	**78.32**	1.721	80.16	1.561
110	87.4	1.609	87.37	1.597	87.35	1.633	87.18	1.619	**87.05**	1.612	88.99	1.464
120	95.94	1.63	95.91	1.606	96.12	1.602	95.93	1.613	**95.81**	1.672	97.77	1.488
130	104.66	1.739	**104.62**	1.725	104.99	1.742	104.84	1.697	104.7	1.742	106.68	1.557
140	113.14	2.085	**113.1**	2.034	113.62	2.097	113.48	2.029	113.36	2.105	115.39	1.835
150	121.91	1.481	**121.86**	1.482	122.63	1.523	122.52	1.475	122.34	1.529	124.26	1.342

A Tabela 4.2 mostra os resultados deste experimento, onde os números em negrito são os melhores valores para cada comprimento. Desta tabela podemos observar o seguinte: AMBO tem os melhores valores para permutações até de comprimento 50; para permutações de comprimento maior o igual a 60 até 120, AMBO-Híbrido tem os melhores valores; para permutações de comprimentos maior o igual a 130, AG-Híbrido tem os melhores valores, e com AG-SA tendo os segundos melhores valores. Note que na Tabela 4.2 não são mostrados os resultados do algoritmo de raio de aproximação 1.5 [27] (e a sua versão corrigida [75]) para o problema de OPSSR, já que foi relatado em [76] que AG-SA e AG-Híbrido conseguiram melhores resultados.

Testes Estatísticos

Para a comparação estatística do desempenho dos algoritmos, a seguinte metodologia proposta por Demšar [37] foi aplicada (ver também [42] e [38]):

- Primeiro, o teste de Friedman é usado para testar a hipótese nula que todos os algoritmos têm o mesmo desempenho.
- Se o teste anterior rejeita a hipótese nula, se procede a realizar o teste de Holm como teste post-hoc. Este teste considera um algoritmo de controle e o compara como os algoritmos restantes para testar a hipótese nula de que o algoritmo de controle e um dos outros algoritmos restantes têm o mesmo desempenho.

O pacote `CONTROLTEST` (implementado em `Java`) foi usado para realizar os testes de Friedman e Holm, o qual está disponível no web site do grupo SCI2S `http://sci2s.ugr.es/sicidm`. Neste pacote o algoritmos de controle é aquele com o menor *rank* computado no teste de Friedman. Para ambos testes foi usado um nível de significância de $\alpha = 0.05$.

Tabela 4.3: Resultados do teste de Holm para conjuntos de cem permutações de comprimento de 20 até 50.

Comprimento	**Algoritmo Controle**	**i**	**Algoritmo**	**Rank**	**P-value**	α/i
20	AMBO (Rank: 2.73)	5	**AG-Híbrido**	4.52	1.9625E-6	0.01
		4	**AG-SA**	4.26	4.8577E-5	0.0125
		3	AG-Paralelo	3.41	0.0733	0.0167
		2	AMBO-Híbrido	3.21	0.2091	0.025
		1	AM	2.86	0.7484	0.05
30	AMBO (Rank: 2.12)	5	**AG-Paralelo**	4.77	1.4166E-12	0.01
		4	**AG-SA**	4.68	7.8156E-12	0.0125
		3	**AG-Híbrido**	4.42	7.8958E-10	0.0167
		2	AM	2.61	0.1903	0.025
		1	AMBO-Híbrido	2.40	0.4543	0.05
40	AMBO (Rank: 1.76)	5	**AG-Paralelo**	5.72	3.5526E-26	0.01
		4	**AG-SA**	4.60	3.1936E-14	0.0125
		3	**AG-Híbrido**	4.52	1.6261E-13	0.0167
		2	AM	2.35	0.1148	0.025
		1	AMBO-Híbrido	2.05	0.4383	0.05
50	AMBO (Rank: 1.66)	5	**AG-Paralelo**	5.96	1.4433E-30	0.01
		4	**AG-SA**	4.62	2.5547E-15	0.0125
		3	**AG-Híbrido**	4.42	1.6261E-13	0.0167
		2	**AM**	2.55	0.0174	0.025
		1	AMBO-Híbrido	1.79	0.7283	0.05

As saídas de cada algoritmo, para o experimento atual, foram usadas como amostra de entrada para o teste de Friedman e Holm. Estas amostras contêm 100 elementos correspondentes aos resultados (número de reversões) de um algoritmo para um conjunto de

cem permutações. Cada elemento de uma amostra de cem elementos foi pré-processada calculando seu inverso multiplicativo, para usá-los nos testes estatísticos, os quais comparam o desempenho dos algoritmos. Assim, enquanto o número de reversões diminui o desempenho aumenta.

Tabela 4.4: Resultados do teste de Holm para conjuntos de cem permutações de comprimento de 60 até 150.

Comprimento	**Algoritmo Controle**	**i**	**Algoritmo**	**Rank**	**P-value**	α/i
60	AMBO-Híbrido (Rank: 1.42)	5	**AG-Paralelo**	5.96	7.0047E-34	0.01
		4	**AG-SA**	4.56	4.7793E-17	0.0125
		3	**AG-Híbrido**	4.48	2.8813E-16	0.0167
		2	**AM**	2.83	1.6431E-4	0.025
		1	AMBO	1.75	0.3778	0.05
70	AMBO-Híbrido (Rank: 1.39)	5	**AG-Paralelo**	6.00	7.0046E-35	0.01
		4	**AG-SA**	4.52	5.9974E-17	0.0125
		3	**AG-Híbrido**	4.48	1.4767E-16	0.0167
		2	**AM**	2.81	1.4758E-4	0.025
		1	AMBO	1.80	0.2732	0.05
80	AMBO-Híbrido (Rank: 1.49)	5	**AG-Paralelo**	6.00	1.8592E-33	0.01
		4	**AG-SA**	4.51	6.9560E-16	0.0125
		3	**AG-Híbrido**	4.45	2.5547E-15	0.0167
		2	**AM**	2.96	8.5392E-5	0.025
		1	AMBO	1.59	0.7893	0.05
90	AMBO-Híbrido (Rank: 1.50)	5	**AG-Paralelo**	6.00	2.5706E-33	0.01
		4	**AG-SA**	4.51	8.6553E-16	0.0125
		3	**AG-Híbrido**	4.19	6.5111E-13	0.0167
		2	**AM**	3.08	2.4136E-5	0.025
		1	AMBO	1.72	0.5565	0.05
100	AMBO-Híbrido (Rank: 1.38)	5	**AG-Paralelo**	6.00	5.0268E-35	0.01
		4	**AG-SA**	4.21	3.9239E-14	0.0125
		3	**AG-Híbrido**	3.99	3.0474	0.0167
		2	**AM**	3.44	3.6795E-8	0.025
		1	AMBO	1.98	0.1088	0.05
110	AMBO-Híbrido (Rank: 1.44)	5	**AG-Paralelo**	6.00	3.6409E-34	0.01
		4	**AG-Híbrido**	3.95	1.9696E-11	0.0125
		3	**AG-SA**	3.93	2.8368E-11	0.0167
		2	**AM**	3.52	2.7127E-8	0.025
		1	AMBO	2.16	0.0543	0.05
120	AMBO-Híbrido (Rank: 2.15)	5	**AG-Paralelo**	6.00	7.8536E-25	0.01
		4	**AM**	4.49	4.0029E-10	0.0125
		3	AG-SA	2.83	0.0692	0.0167
		2	AMBO	2.82	0.0733	0.025
		1	AG-Híbrido	2.71	0.1345	0.05
130	AG-Híbrido (Rank: 2.13)	5	**AG-Paralelo**	6.00	4.5017E-25	0.01
		4	**AM**	4.41	1.1043E-9	0.0125
		3	**AMBO**	3.57	1.1881E-4	0.0167
		2	AMBO-Híbrido	2.48	0.3496	0.025
		1	AG-SA	2.41	0.4543	0.05
140	AG-Híbrido (Rank: 1.67)	5	**AG-Paralelo**	6.00	5.6861E-31	0.01
		4	**AM**	4.44	1.3301E-13	0.0125
		3	**AMBO**	3.79	1.4622E-8	0.0167
		2	**AMBO-Híbrido**	3.05	2.2584E-4	0.025
		1	AG-SA	2.05	0.3098	0.05
150	AG-Híbrido (Rank: 1.45)	5	**AG-Paralelo**	6.00	5.0519E-34	0.01
		4	**AM**	4.45	1.0762E-15	0.0125
		3	**AMBO**	4.11	1.1676E-12	0.0167
		2	**AMBO-Híbrido**	3.24	1.7186E-6	0.025
		1	AG-SA	1.75	0.4227	0.05

Os resultados dos testes estatísticos são os seguintes: o teste de Friedman rejeitou a hipótese nula para todos os comprimentos, exceto a amostra para permutações de comprimento 10, assim esta amostra não sera incluída para o teste de Holm. A Tabela 4.3 e 4.4 mostram os resultados do teste de Holm, onde os algoritmos em negrito são aqueles que tem uma diferença estatisticamente significativa (p-value $\leq \alpha/i$) com seus respectivos algoritmos de controle.

Da Tabela 4.3 pode ser observado o seguinte: AMBO é o algoritmo de controle para todos os casos (comprimentos de 20 até 50), tendo os mínimos *ranks*. Note que em todos os casos AMBO não tem uma diferença estatisticamente significativa com respeito a AMBO-Híbrido.

Da Tabela 4.4 pode ser observado o seguinte: para permutações de comprimento 60 até 120, AMBO-Híbrido é o algoritmo de controle e não tem uma diferença estatisticamente significativa com respeito a AMBO; para permutações de comprimento 130 até 150, o AG-Híbrido é o algoritmo de controle e não tem uma diferença estatisticamente significativa com respeito a AG-SA.

4.4.2 Experimentos usando Permutações sem Sinal Simples (Benchmarks)

Para este experimento, os casos mais difíceis das permutações *benchmarks* propostas em [77] foram usados. Estas permutações são as seguintes: 1RPL50, 2RPL50, 1RPL100, 2RPL100, 1RPL150, 2RPL150; o sufixo númerico depois de "RPL"significa o comprimento da permutação. A comparação foi realizada com os algoritmos AG-SA, AG-Híbrido, AM, AMBO, AMBO-Híbrido, e AG-Paralelo. O tamanho da população, para todos os algoritmos, foi fixada na mesma forma que na subseção 4.4.1. A configuração deste experimento segue a continuação.

- Para cada permutação benchmark:
  - Primeiro, executar o algoritmo AMBO uma única vez por 200 gerações e salvar o número de avaliações da função de aptidão. Este número, digamos k, será usado como critério de parada para outros algoritmos quando for usado a permutação benchmark atual.
  - Logo, executar todos os algoritmos 50 vezes para a permutação atual. Em cada execução, parar o algoritmo quando for alcançado k avaliações da função de aptidão.
  - Finalmente, calcular as seguintes medidas para 50 saídas (número de reversões) de cada algoritmo: melhor, pior, media, mediana, e desvio padrão.

Tabela 4.5: Medidas diferentes para os resultados (número de reversões) do experimento com seis permutações benchmarks.

Bench.	**Algoritmo**	**Melhor**	**Pior**	**Mediana**	**Média**	**Dev. Pdr.**
1RPL50	AG-SA	37	38	37.0	37.18	0.388
	AG-Híbrido	37	39	37.0	37.28	0.497
	AM	37	38	37.0	37.06	0.24
	AMBO	37	38	37.0	37.06	0.24
	AMBO-Híbrido	**37**	**37**	**37.0**	**37.0**	**0.0**
	AG-Paralelo	37	39	38.0	37.8	0.452
2RPL50	AG-SA	36	39	37.0	37.2	0.756
	AG-Híbrido	36	39	37.0	37.14	0.756
	AM	36	38	37.0	36.58	0.575
	AMBO	**36**	**37**	**36.0**	**36.28**	**0.454**
	AMBO-Híbrido	36	38	36.0	36.48	0.544
	AG-Paralelo	36	38	38.0	37.58	0.538
1RPL100	AG-SA	78	83	80.0	80.48	1.074
	AG-Híbrido	79	83	81.0	80.68	0.999
	AM	79	82	81.0	80.46	0.908
	AMBO	79	83	80.0	80.36	0.776
	AMBO-Híbrido	**79**	**82**	**80.0**	**80.2**	**0.728**
	AG-Paralelo	80	83	82.0	81.72	0.607
2RPL100	AG-SA	77	81	79.0	78.9	0.886
	AG-Híbrido	76	80	79.0	78.62	0.923
	AM	77	80	79.0	78.5	0.735
	AMBO	77	80	79.0	78.58	0.859
	AMBO-Híbrido	**77**	**80**	**78.0**	**78.3**	**0.707**
	AG-Paralelo	79	81	80.0	80.0	0.606
1RPL150	AG-SA	120	125	122.5	122.44	1.128
	AG-Híbrido	**120**	**126**	**122.0**	**122.12**	**1.409**
	AM	121	125	123.0	123.12	1.272
	AMBO	120	125	123.0	122.82	1.173
	AMBO-Híbrido	120	125	123.0	122.88	0.982
	AG-Paralelo	121	126	125.0	124.5	0.995
2RPL150	AG-SA	121	127	124.0	124.12	1.256
	AG-Híbrido	**121**	**129**	**124.0**	**123.94**	**1.621**
	AM	123	128	125.0	125.14	1.125
	AMBO	122	127	125.0	124.9	1.199
	AMBO-Híbrido	122	127	125.0	125.04	1.087
	AG-Paralelo	125	128	127.0	126.56	0.837

A Tabela 4.5 mostra os resultados dos experimentos, onde as linhas em negrito representam os algoritmos com a menor média. Desta tabela pode ser visto o seguinte: para os benchmarks 1RPL50 e 2RPL50, AMBO-Híbrido e AMBO calculam os valores mínimos para todas as medidas respectivamente; para os benchmarks 1RPL100 e 2RPL100, AMBO-Híbrido tem os valores mínimos para as medidas pior, mediana, e média; para os benchmarks 1RPL150 e 2RPL150, AG-Híbrido tem os valores mínimos para as medidas melhor, mediana, e média.

Testes Estatísticos

A mesma metodologia previamente explicada na subseção 4.4.1 foi aplicada para a comparação estatística. Neste caso as amostras de entrada contém 50 elementos que são os resultados de 50 execuções de um algoritmo para uma permutação benchmark.

Os resultados dos testes estatísticos são os seguintes: o teste de Friedman rejeito a hipótese nula de que todos os algoritmos têm o mesmo desempenho para todos os benchmarks. A Tabela 4.6 mostra os resultados do teste de Holm, onde os algoritmos em negrito são aqueles que têm uma diferença estatisticamente significativa (p-value $\leq \alpha/i$) com seu respectivo algoritmo de controle.

Da Tabela 4.6 podemos observar o seguinte: para os benchmarks 1RPL50 e 2RPL50, AMBO-Híbrido e AMBO são os algoritmos de controle, e não tem uma diferença estatisticamente significativa com respeito a AMBO e AMBO-Híbrido respectivamente; para os benchmarks 1RPL100 e 2RPL100, AMBO-Híbrido é o algoritmo de controle, e não tem uma diferença estatisticamente significativa com respeito a AMBO; para os benchmarks 1RPL150 e 2RPL150, Hybrid-GA é o algoritmo de controle, e não tem uma diferença estatisticamente significativa com respeito a AG-SA.

Tabela 4.6: Resultados do teste de Holm para permutações benchmarks.

Comprimento	**Algoritmo Controle**	**i**	**Algoritmo**	**Rank**	**P-value**	α/i
1rpl50	AMBO-Híbrido (Rank: 2.83)	5	**AG-Paralelo**	5.16	4.7488E-10	0.01
		4	AG-Híbrido	3.62	0.0347	0.0125
		3	GA	3.37	0.1490	0.0167
		2	AM	3.01	0.6305	0.025
		1	AMBO	3.01	0.6305	0.05
2rpl50	AMBO (Rank: 2.24)	5	**AG-Paralelo**	5.0	1.6261E-13	0.01
		4	**AG-SA**	4.15	3.3134E-7	0.0125
		3	**AG-Híbrido**	4.0	2.5537E-6	0.0167
		2	AM	2.93	0.0652	0.025
		1	AMBO-Híbrido	2.68	0.2396	0.05
1rpl100	AMBO-Híbrido (Rank: 2.77)	5	**AG-Paralelo**	5.29	1.6395E-11	0.01
		4	AG-Híbrido	3.47	0.0614	0.0125
		3	AG-SA	3.29	0.1646	0.0167
		2	AM	3.23	0.2189	0.025
		1	AMBO	2.95	0.6305	0.05
2rpl100	AMBO-Híbrido (Rank: 2.57)	5	**AG-Paralelo**	5.52	3.1654E-15	0.01
		4	**AG-SA**	3.57	0.0075	0.0125
		3	AG-Híbrido	3.23	0.0777	0.0167
		2	AMBO	3.17	0.1088	0.025
		1	AM	2.94	0.3227	0.05
1rpl150	AG-Híbrido (Rank: 2.51)	5	**AG-Paralelo**	5.40	1.1287E-14	0.01
		4	**AM**	3.66	0.0021	0.0125
		3	**AMBO-Híbrido**	3.45	0.012	0.0167
		2	AMBO	3.15	0.0872	0.025
		1	AG-SA	2.83	0.3924	0.05
2rpl150	AG-Híbrido (Rank: 2.44)	5	**AG-Paralelo**	5.38	3.9194E-15	0.01
		4	**AM**	3.65	0.0012	0.0125
		3	**AMBO-Híbrido**	3.6	0.0019	0.0167
		2	**AMBO**	3.47	0.0059	0.025
		1	AG-SA	2.46	0.9574	0.05

4.4.3 Experimentos usando Permutações baseadas em Dados Biológicos

Permutações baseadas no genoma mitocondrial de muitos organismos foram usadas em [50], as quais foram construídas de acordo com um procedimento similar ao proposto em [77]. Estas permutações são criadas na seguinte forma:

- Primeiro, os dois genomas A e B são escolhidos.
- Aqueles genes que não são compartilhados por ambos genomas A e B são eliminados.
- Logo, uma sequência crescente de naturais é atribuída aos genes do genoma B de tal forma que está sequência forma a permutação identidade.
- Finalmente, a permutação π_{A_B} é construída como uma sequência de naturais dos genes do genoma A, de acordo com a mesma atribuição de naturais dado aos genes no genoma B.

Tabela 4.7: Número de genes dos genomas mitocondriais de alguns organismos.

Nome Cientifico	**Nome Comum**	**Num.**	**Abrev.**
Homo sapiens	Humano	37	Hom.
Drosophila melanogaster	Mosca da Fruta	37	Dro.
Crocodylus mindorensis	Crocodilo Filipino	35	Cro.
Sibon nebulatus	*Clouded Snake*	37	Sib.
Caretta caretta	Tartaruga Marinha Comum	36	Car.

As permutações usadas neste experimento são aquelas construídas usando o procedimento anterior e são baseadas nos genomas mitocondriais dos organismos mostrados na Tabela 4.7. A permutação relacionada a *Homo sapiens* e *Caretta caretta* não foi incluída porque depois da eliminação dos genes não compartilhados, a permutação gerada foi a identidade, a qual não precisa ser ordenada. A comparação foi realizada com os seguintes algoritmos: AG-SA, AG-Híbrido, AM, AMBO, AMBO-Híbrido,e AG-Paralelo. O tamanho da população foi fixado em $n \log n$ para todos os algoritmos, onde n é o comprimento da permutação de entrada. Para o caso do AG-Paralelo o tamanho da população foi divido por 23, que é o número de processos escravos. A configuração deste experimento segue a continuação.

- Para cada permutação baseada em dados biológicos:
  - Primeiro, executar o algoritmo AMBO uma única vez por 200 gerações e salvar o número de avaliações da função de aptidão. Este número de avaliações,

digamos k, será usada como critério de parada para os outros algoritmos quando for usada a permutação benchmark atual.

– Segundo, executar todos os algoritmos 50 vezes para a permutação atual. Em cada execução, parar os algoritmos quando alcançarem k avaliações da função de aptidão.

– Finalmente, calcular as seguintes medidas para as 50 saídas (número de reversões) de cada algoritmo: melhor, pior, média, mediana, e desvio padrão.

Tabela 4.8: Medidas diferentes para os resultados (número de reversões) do experimento com permutações biológicas.

Perm.	**Algoritmo**	**Melhor**	**Pior**	**Mediana**	**Média**	**Desv. Pad.**
Hom.-Dro.	AG-SA	**16**	**17**	**16.0**	**16.04**	**0.198**
	AG-Híbrido	16	16	16.0	16.0	0.0
	AM	16	16	16.0	16.0	0.0
	AMBO	16	16	16.0	16.0	0.0
	AMBO-Híbrido	16	16	16.0	16.0	0.0
	AG-Paralelo	16	16	16.0	16.0	0.0
Hom.-Cro.	Todos os algoritmos	3	3	3.0	3.0	0.0
Hom.-Sib.	Todos os algoritmos	2	2	2.0	2.0	0.0
Dro.-Cro.	AG-SA	**15**	**16**	**15.0**	**15.02**	**0.141**
	AG-Híbrido	15	15	15.0	15.0	0.0
	AM	15	15	15.0	15.0	0.0
	AMBO	15	15	15.0	15.0	0.0
	AMBO-Híbrido	15	15	15.0	15.0	0.0
	AG-Paralelo	15	15	15.0	15.0	0.0
Dro.-Sib.	AG-SA	**17**	**18**	**17.0**	**17.12**	**0.328**
	AG-Híbrido	**17**	**18**	**17.0**	**17.04**	**0.198**
	AM	**17**	**18**	**17.0**	**17.02**	**0.141**
	AMBO	17	17	17.0	17.0	0.0
	AMBO-Híbrido	17	17	17.0	17.0	0.0
	AG-Paralelo	17	17	17.0	17.0	0.0
Dro.-Car.	AG-SA	16	16	16.0	16.0	0.0
	AG-Híbrido	**16**	**17**	**16.0**	**16.38**	**0.49**
	AM	16	16	16.0	16.0	0.0
	AMBO	16	16	16.0	16.0	0.0
	AMBO-Híbrido	**16**	**17**	**16.0**	**16.36**	**0.485**
	AG-Paralelo	16	16	16.0	16.0	0.0
Cro.-Sib.	Todos os algoritmos	5	5	5.0	5.0	0.0
Cro.-Car.	Todos os algoritmos	3	3	3.0	3.0	0.0
Sib.-Car.	Todos os algoritmos	2	2	2.0	2.0	0.0

A Tabela 4.8 mostra os resultados do experimento atual, onde as linhas em negrito representam os algoritmos que não têm os valores mínimos para alguma medida. Desta tabela os seguintes resultados podem ser observados: para todas as permutações biológicas, AMBO é o unico em alcançar os resultados mínimos; para as permutações Hom.-Cro., Hom.-Sib., Cro.-Sib., Cro.-Car. e Sib.-Car., todos os algoritmos têm os mesmos resultados.

Note que que os dados mostrados na Tabela 4.8 podem ser usados como informação valiosa para construir uma arvore filogenética dos organismos mostrados na Tabela 4.7, como foi feito por Sankoff et al. em [70] usando uma variante da distância de reversão.

Testes Estatísticos

A comparação estatística foi realizada usando a metodologia explicada na subseção 4.4.1. Neste caso, a amostra de entrada contêm 50 elementos que são os resultados de 50 execuções de um algoritmo para uma determinada permutação biológica.

Tabela 4.9: Resultados do teste de Holm para a permutação Dro.-Car.

Comprimento	**Algoritmo Controle**	**i**	**Algoritmo**	**Rank**	**P-value**	α/i
Dro.-Car.	AG-SA (Rank: 3.13)	5	**AG-Híbrido**	4.27	0.0023	0.01
		4	**AMBO-Híbrido**	4.21	0.0039	0.0125
		3	AM	3.13	1.0	0.0167
		2	AMBO	3.13	1.0	0.025
		1	AG-Paralelo	3.13	1.0	0.05

Os resultados dos testes estatísticos são os seguintes: o teste de Friedman rejeitou a hipótese nula de que todos os algoritmos têm o mesmo desempenho só para a permutação Dro.-Cro. A Tabela 4.9 mostra os resultados do teste de Holm onde os algoritmos em negrito são aqueles que têm uma diferença estatisticamente significante (p-value $\leq \alpha/i$) com respeito ao algoritmo de controle. A partir desta tabela podemos observar o seguinte: o algoritmo de controle é AG-SA com *rank* 3.13, mas note que AG-Paralelo, AMBO, e AM tem o mesmo *rank*, e qualquer desses algoritmos poderia ser tomado como algoritmo de controle.

4.4.4 Discussão

Outro tipo de permutação interessante que não foi incluída na seção dos experimentos são as permutações de Gollan, as quais precisam exatamente $n - 1$ reversões para ser ordenadas. As permutações de Gollan e suas inversas são as permutações como o pior caso com respeito ao problema da distância de reversão ([9]). Adicionalmente experimentos

Tabela 4.10: Média, desvio padrão, e speedup do tempo de execução (em milissegundos) do AG-SA e AG-Paralelo.

Cmpr.	**AG-SA**		**AG-Paralelo**		**Speedup**
	Média	Desv. Pad.	Média	Desv. Pad.	
10	71.02	8.688	1258.69	10.101	0.06
20	279.31	21.104	1323.28	24.963	0.21
30	641.78	26.525	1374.06	24.622	0.47
40	1193.57	33.215	1424.4	45.558	0.84
50	1912.76	43.92	1531.56	73.856	**1.25**
60	2820.85	67.228	1647.03	96.263	**1.71**
70	3939.69	67.111	1772.48	126.515	**2.22**
80	5218.54	81.364	1880.58	158.324	**2.77**
90	6706.0	92.378	1979.74	140.465	**3.39**
100	8501.2	148.582	2174.99	180.419	**3.91**
110	10374.58	141.598	2224.43	97.91	**4.66**
120	12600.36	143.978	2470.85	118.27	**5.1**
130	14896.29	187.142	2703.62	130.188	**5.51**
140	17620.71	212.51	2981.15	171.148	**5.91**
150	20523.42	218.488	3231.33	128.509	**6.35**

foram realizados usando estas permutações, assim foram executados todos os algoritmos até alcançarem $n-1$ reversões. Resultados deste experimento mostraram que todos os algoritmos precisaram só uma ou duas gerações para alcançar $n-1$ reversões, portanto, este tipo de permutações são instâncias fáceis de serem resolvidas e não são adequadas para realizar comparações.

Nos trabalhos [79], [74] e [77] afirmamos que o AG-Paralelo (baseado no modelo de execuções independentes, de acordo com [80]) tinha os melhores resultados para o problema OPSSR, mas o fato que este algoritmo usa mais recursos (23 vezes o tamanho da população de AG-SA ($n \log n$)) claramente indica uma comparação injusta. Quando a mesma quantidade de recursos (tamanho da população e número de avaliações da função de aptidão) são atribuídos a todos os algoritmos, como anteriormente visto no experimento usando conjuntos de cem permutações sem sinal, o algoritmo AG-Paralelo não produz os melhores resultados. Logo, um experimento adicional foi realizado (usando os mesmos recursos e executados ambos algoritmos na plataforma `Ubuntu Linux` com dois processadores Intel Xeon) para comparar o tempo de execução de AG-SA e AG-Paralelo para conjuntos de cem permutações sem sinal. A Tabela 4.10 mostra os resultados deste experimento, onde o AG-Paralelo mostra um melhor speedup para permutações de comprimento maior ou igual a 50. Assim, como esperado, a contribuição real do AG-Paralelo é melhorar o tempo de AG-SA.

Logo, a convergência dos algoritmos até os resultados mínimos foi calculada para visualizar o fato que o AG-Híbrido tem os melhores resultados para permutações maiores ou iguais a 130. As Figuras 4.5, 4.6, e 4.7 mostram os gráficos de convergência que foram calculados para as seguintes permutações benchmarks 1RPL50, 1RPL100, e 1RPL150.

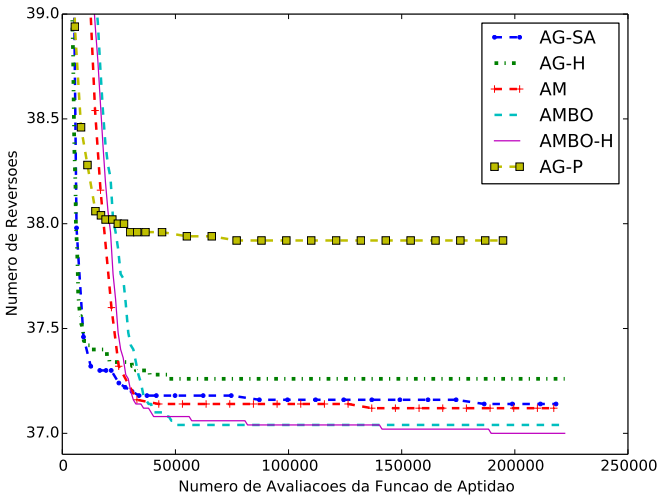

Figura 4.5: Resultados da Média de 50 execuções para o benchmark 1RPL50

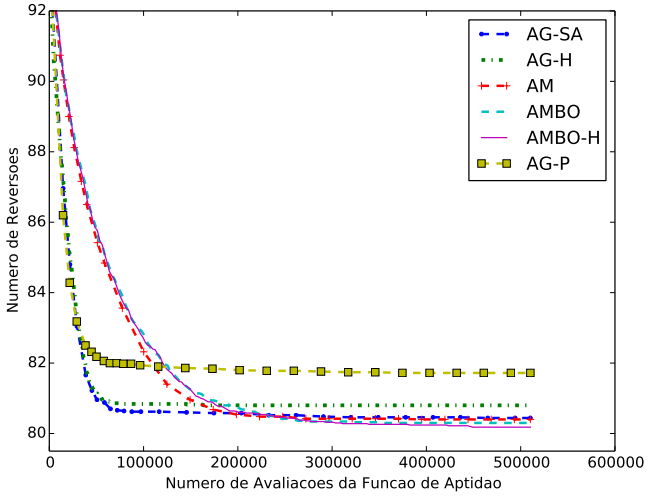

Figura 4.6: Resultados da Média de 50 execuções para o benchmark 1RPL100

A Figura 4.5 mostra que para o benchmark 1RPL50, AMBO e AMBO-Híbrido têm uma convergência aos valores mínimos (número de reversões) quando é incrementado o número de avaliações da função de aptidão. A Figura 4.6 mostra que para o benchmark 1RPL100, ainda AMBO e AMBO-Híbrido têm uma convergência aos valores mínimos quando é incrementado o número de avaliações da função de aptidão, mas desta vez mais

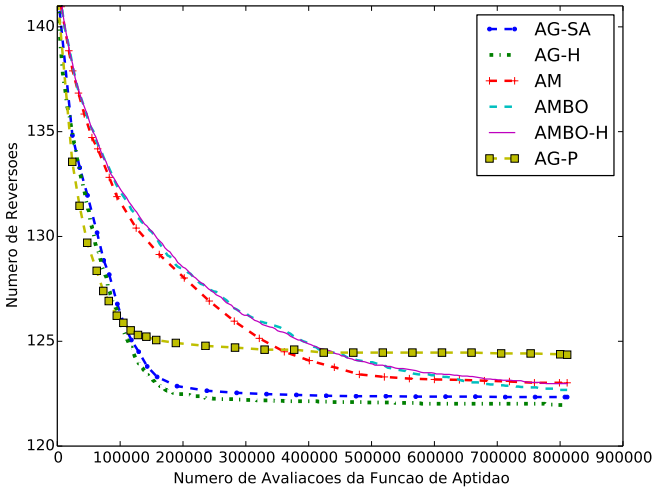

Figura 4.7: Resultados da Média de 50 execuções para o benchmark 1RPL150

perto daqueles valores de AG-SA e AG-Híbrido. A Figura 4.7 mostra que para o benchmark 1RPL150, AG-SA e AG-Híbrido têm uma convergência aos mínimos valores, desta vez melhor que AMBO e AMBO-Híbrido. Baseado nestes resultados podemos dizer que a convergência de AMBO (e AMBO-Híbrido) piora para permutações grandes (comprimentos ≥ 130), o que mostra que as estrategias de busca local e busca por oposição não são tão efetivas quando o espaço de busca para o problema OPSSR é consideravelmente grande.

4.5 Algoritmos Evolutivos e Outras Distâncias

Como contribuição em outros trabalhos [34, 35, 33], que não formam parte desta tese, foram reutilizadas muitas das abordagens mostradas neste capítulo, desta vez para resolver o problema do cálculo da distância de translocação. Desta forma mostramos a portabilidade da nossa metodologia para poder resolver outro problemas similares, ou seja, aproveitar as soluções exatas a um problema que pertence a classe $\mathcal{P}$ para encontrar soluções a um problema da classe $\mathcal{NP}$-Difícil. Nesta seção citaremos estes trabalhos e os resultados mais relevantes.

Uma operação de translocação envolve o intercâmbio de blocos de genes entre os cromossomos de um genoma. Por exemplo, considere o genoma A com dois cromossomos $A = (x_1,\ x_2,\ x_3,\ x_4,\ x_5)\ (y_1,\ y_2,\ y_3,\ y_4,\ y_5)$ uma translocação tomando como referência os elementos x_3 e y_2 transforma o genoma A em $(\mathbf{x_1},\ \mathbf{x_2},\ \mathbf{x_3},\ y_3,\ y_4,\ y_5)\ (\mathbf{y_1},\ \mathbf{y_2},\ x_4,\ x_5)$, este tipo de translocação é conhecida como prefixo-prefixo. Por outro lado, existe mais um tipo de translocação conhecido prefixo-sufixo onde a aplicação de uma translocação sobre

o genoma A tomando como referência x_3 e y_2 da como resultado $A = (\mathbf{x_1},\ \mathbf{x_2},\ \mathbf{x_3},\ y_2,\ y_1)$ $(\mathbf{y_5},\ \mathbf{y_4},\ \mathbf{y_3},\ x_4,\ x_5)$.

O problema da distância de translocação consiste em encontrar o número mínimo de translocações para transformar um genoma em outro. Este problema tem duas versões dependendo do tipo de genomas usados: com sinal ou sem sinal. A versão do problema com genomas com sinal pertence a classe $\mathcal{P}$, em efeito, existe um algoritmo de tempo linear proposto por Bergeron et al. [11]. Por outro lado, a versão do problema com genomas sem sinal pertence a classe $\mathcal{NP}$-Difícil [92].

Para a versão do problema sem sinal, propusemos um AG (Da Silveira et al. [34]) onde o espaço de soluções consiste em 2^n versões com sinal do genoma sem sinal de entrada para o qual precisa ser calculado a distância de translocação. Logo, esta abordagem é similar a proposta para o problema OPSSR, ou seja, é baseada em duas observações: (1) a solução para um genoma com sinal (versão com sinal de um genoma sem sinal A) é uma solução válida para o genoma sem sinal A, (2) a solução de um dos genomas do espaço de soluções (versões com sinal do genoma sem sinal A) representa uma solução ótima para o genoma sem sinal A. Ademais, outra característica importante deste AG é que usa o algoritmo linear de Bergeron et al. [11] para avaliar a aptidão dos indivíduos da população. Resultados dos experimentos mostraram que o AG teve melhores resultados que a implementação do algoritmo teórico de raio de aproximação $1.5+\varepsilon$ [32].

Depois, propomos melhorias [35] sobre o AG proposto em [34]. Heurísticas de busca local e busca por oposição (tipo-I) foram embebidas nas seguintes etapas: (1) geração da população inicial, (2) reinicio da população, e (3) como uma nova etapa depois do ciclo de reprodução. A inclusão destas heurísticas deu lugar a dois novos algoritmos: AG com busca local, e AG com busca por oposição. Resultados dos experimentos mostraram que o AG com busca local teve os melhores resultados com respeito aos outros algoritmos. Também foram realizados experimentos com dados biológicos para os quais os três AG's tiveram o mesmo desempenho.

Posteriormente, propomos duas abordagens paralelas [33] baseadas no AG com busca local proposto em [35]. A primeira abordagem usa o modelo de Mestre-Escravo de População-Única no qual o cálculo da aptidão da população é feito em paralelo. A segunda abordagem usa o modelo de Múltiplas-Populações, a qual tem duas variantes uma com intercâmbio de indivíduos e outra sem intercâmbio. Resultados dos experimentos mostraram que a primeira abordagem paralela melhora o tempo de execução do AG com busca local, logo a segunda abordagem (variante sem intercâmbio de indivíduos) melhorou a acurácia (número de translocações) com respeito ao AG com busca local.

Sumário do Capítulo:

Neste capítulo foram apresentadas melhorias sobre um algoritmo genético simples para resolver o problema OPSSR. Dentre as melhorias temos a inclusão de heurísticas como busca local, busca por oposição, e eliminação de pontos de quebra, também foram propostas abordagens paralelas. Vários experimentos foram realizados usando permutações geradas aleatoriamente, e baseadas em dados biológicos. Os resultados dos experimentos mostraram que o algoritmo AMBO e AMBO-Híbrido são aos mais adequados para casos práticos, ou seja, permutações de comprimento até 120. Estes resultados foram confirmados pelos seguintes testes estatísticos: Friedman e Holm.

No final deste capítulo foi mostrada a portabilidade da nossa metodologia para resolver o problema do cálculo da distância de translocação.

Capítulo 5

Heurísticas para Construção de Árvores Filogenéticas

5.1 Otimizador Guloso para o Problema da Pequena Filogenia

Baseada na abordagem de programação dinâmica proposta por Kováč et al. [51] (chamada de **Kovac-Opt**) neste trabalho propomos uma abordagem gulosa para solucionar o problema da pequena filogenia. A principal motivação para propor uma abordagem gulosa (chamada de **Greedy-Opt**) foi diminuir o tempo de execução usado pelo software `PIVO`, o qual é extremamente lento.

O Algoritmo 15 mostra o pseudocódigo da abordagem gulosa. A ideia principal é gerar um conjunto de candidatos para um determinado nó i, os quais são gerados aplicando uma operação DCJ. Um dos candidatos que tem o escore mínimo é escolhido para substituir o nó i. O escore de um candidato é calculado somando as distâncias do candidato com respeito a: o antecessor de i, o descendente esquerdo de i, e o descendente direito de i.

O principal inconveniente de gerar candidatos é que o número de candidatos pode ser grande, e portanto este fato pode incrementar o tempo de processamento. Para superar este inconveniente o conjunto de candidatos foi reduzido evitando a aplicação de operações DCJ sobre adjacências (Definição 11) que também aparecem simultaneamente no antecessor de i, o descendente esquerdo de i, e o descendente direito de i.

Uma melhoria importante tomada do software `PIVO` [51] é reusar como candidatos os nós internos rotulados que foram encontrados pelo Algoritmo 15 em uma execução prévia. Isto implica que devemos executar o Algoritmo 15 por algumas iterações para explorar esta melhoria.

Algoritmo 15: Algoritmo Guloso para o Problema da Pequena Filogenia

Entrada: Um dataset de genomas, a estrutura de uma árvore T
Saída: O escore de da arvore T

```
1  Realizar uma rotulação inicial dos nós internos;
2  novoEscore ← Calcular o custo da árvore T;
3  escore ← ∞;
4  enquanto novoEscore < escore faça
5      escore ← novoEscore;
6      para cada nó interno i faça
7          d1, d2, d3 ← Calc. distâncias de i e seu antecessor, i e seu desc. esqu., e i e
             seu desc. dir. respectivamente;
8          d ← d1 + d2 + d3;
9          Gerar candidatos para o nó i;
10         para cada candidato c faça
11             nd1, nd2, nd3 ← Calc. distâncias de c e o antecessor de i, c e o desc.
                 esqu. de i, e c e o desc. dir. de i respectivamente;
12             nd ← nd1 + nd2 + nd3;
13             se nd < d então
14                 d ← nd;
15                 Salvar o candidato c;
16         Substituir nó interno i pelo candidato salvo c (caso exista);
17     novoEscore ← Calcular custo da árvore T;
18 Recuperar estado anterior dos nós rotulados;
19 Retornar escore;
```

5.2 Busca em Vizinhança Variável para o Problema da Grande Filogenia

O espaço de busca, formado por estruturas de árvores, relacionado com o problema da grande filogenia quando é usada a ordem dos genes foi explorado ou de forma exaustiva [17, 18] ou tomando amostras [58, 57]. Heurísticas para explorar este espaço de busca no contexto da ordem dos genes não foram aplicadas, apesar disso para o caso quando são usadas sequências de caracteres (como sequências de DNA) existe literatura extensa sobre a reconstrução de árvores filogenéticas usando heurísticas e algoritmos evolutivos [54, 4, 39, 40, 66].

Neste trabalho estamos propondo uma abordagem baseada na busca em vizinhança variável (*Variable Neighborhood Search* - VNS). O VNS é uma meta-heurística proposta por Mladenović e Hansen [55], este framework alterna entre estruturas de vizinhanças buscando por uma solução ótima (ou quase ótima)[46].

Dada uma solução T' (uma estrutura de árvore) dizemos que é um vizinho de T (outra

estrutura de árvore), se a primeira pode ser alcançada a partir da segunda em um único passo (aplicando um operador de movimento). A estrutura da vizinhança $\mathcal{N}(T)$ de uma solução T é o conjunto de todas seus vizinhos [36].

Para o caso de reconstruir árvores filogenéticas usando como data de entrada sequências de caracteres, Andreatta e Ribeiro [4] usaram a variante de VNS conhecida como VNS General com três estruturas de vizinhança. No presente trabalho, uma variante mais simples foi usada, conhecida como VNS Reduzido (ver esta variante em [46]) com quatro estruturas de vizinhança. O Algoritmo 15 é usado para avaliar o escore (custo) de uma determinada estrutura de árvore.

O Algoritmo 16 mostra o pseudocódigo do VNS Reduzido para o problema da grande filogenia, este algoritmo foi implementado no sistema `HELPHY` (***He****uristics for the* ***L****arge and Small* ***Phy****logeny Problem*). Em este algoritmo, quatro estruturas de vizinhanças são usadas e representadas como $\mathcal{N}_k,\ k = 1, \ldots, k_{max} = 4$. Estas estruturas serão apresentadas na seguintes subseções.

Algoritmo 16: VNS Reduzido para o problema da grande filogenia

Entrada: Um dataset de genomas
Saída: Uma estrutura de uma árvore com um escore

```
1  Gerar uma solução inicial T (estrutura de árvore);
2  escore ← Calcular o custo de T (Algoritmo 15);
3  iteração ← 1;
4  enquanto iteração ≤ maxIterações faça
5      k ← 1;
6      enquanto k ≤ k_max faça
7          Agitação:
8          Gerar aleatoriamente uma solução T' (estrutura de árvore) para a estrutura
             de vizinhança N_k(T);
9          novoEscore ← Calcular o custo de T' (Algoritmo 15);
10         Mover-se ou não:
11         se novoEscore < escore então
12             escore ← novoEscore;
13             T ← T';
14             k ← 1;
15         senão
16             k ← k + 1;
17 Retornar árvore T e escore;
```

Como um passo adicional, um procedimento de refinamento é executado sobre a árvore encontrada pelo Algoritmo 16, este procedimento consiste em encontrar os melhores vizinhos das seguintes estruturas de vizinhança: $\mathcal{N}_5$ e $\mathcal{N}_6$ (ver próximas subseções).

Em vez de definir uma estrutura de vizinhança enumerando todos seus vizinhos, podemos defini-la implicitamente fazendo referência as transições potenciais que podem ser alcançadas depois da aplicação de um operador de movimento [36]. Nas seguintes subseções serão apresentados os procedimentos para gerar uma solução inicial e as estruturas de vizinhança já mencionadas nesta seção: $\mathcal{N}_1, \ldots, \mathcal{N}_6$.

5.2.1 Gerando Soluções Iniciais

Dado um conjunto de S genomas (folhas), podemos construir uma árvore filogenética inicial T seguindo os próximos passos:

1. Remover dois elementos aleatoriamente de S os quais formarão uma aresta da árvore inicial T.
2. Remover um elemento aleatório g de S.
3. Insertar g em alguma aresta da árvore parcial T.
4. Ir para o passo 2) ou parar se o conjunto S está vazio.

Existem muitas políticas para inserir um genoma em uma árvore, as quais dão origem a muitas variantes deste algoritmo básico [4]. A variante usada neste trabalho é aquela que insere um genoma na aresta que leva ao primeiro incremento mínimo no custo da árvore. Andreatta e Ribeiro [4] mostraram que esta variante é aquela com o melhor *trade-off* entre acurácia e tempo de execução.

5.2.2 Estruturas de Vizinhança

As estruturas de vizinhança que foram usadas neste trabalho são as seguintes:

- $\mathcal{N}_1$: Intercâmbio do vizinho mais próximo (*Nearest Neighborhood Interchange* - NNI) [88, 4]. Um vizinho é gerado seguindo os seguintes passos: (1) selecionar aleatoriamente (de uma árvore) uma aresta interna, a qual terá quatro subárvores ligadas a ela; (2) intercambiar as posições de dois subárvores não adjacentes, ou seja, subárvores que não compartilham o mesmo nó interno.
- $\mathcal{N}_2$:Passo Simples (*Single Step* - STEP) [4]. Um vizinho é gerado tirando um nó folha de uma árvore e reinserindo-o em uma nova aresta.
- $\mathcal{N}_3$: Poda e Reinserção de Subárvore (*Subtree Pruning and Regrafting* - SPR) [4, 65, 64, 67]. Um vizinho é gerado seguindo os seguintes passos: (1) tirar um nó interno (de uma árvore) dando lugar a três subárvores; (2) juntar dois subárvores por uma aresta; (3) juntar a árvore restante por uma aresta diferente da posição original.

- $\mathcal{N}_4$: Bisseção e Reconexão de Árvore (*Tree Bisection and Reconnection* - TBR) [3, 67]. Um vizinho é gerado seguindo os seguintes passos: (1) tirar uma aresta (de uma árvore) dando lugar a dois subárvores; (2) reconectar as duas subárvores por qualquer par de arestas da primeira e segunda subárvore.
- $\mathcal{N}_5$: Embaralhamento de Subárvore (*Subtree Scramble* - SCRAMBLE) [31]. Um vizinho é gerado selecionando aleatoriamente uma subárvore (de um arvore) e depois rearranjando aleatoriamente a sua estrutura.
- $\mathcal{N}_6$: Intercâmbio de Folhas (*Leaf Swap* - SWAP) [31]. Um vizinho é gerado selecionando aleatoriamente duas folhas (de uma árvore) e depois intercambiando suas posições.

5.3 Experimentos e Resultados

Os experimentos foram realizados usando como entrada dois datasets da literatura: o dataset *Campanulaceae* cpDNA [30, 29] e o dataset *Hemiascomycetes* mtDNA [87]. Estes experimentos foram executados no sistema operacional `OSX` com processador i7 e 16GB de RAM. Também, o software `HELPHY` foi compilado com `gcc` usando a opção `-O2`. A opção de otimização `-O2` foi escolhida em vez de `-O3` porque é mais estável.

Nos experimentos a distância DCJ com restrições tem definições diferentes para cada dataset. No caso do dataset *Campanulaceae*, a distância DCJ com restrições só considera genomas ancestrais (nós internos) com um cromossomo circular. No caso do dataset *Hemiascomycetes*, a distância DCJ com restrições considera ou genomas ancestrais (nós internos) com um cromossomo circular, ou com um ou mais cromossomos lineares.

5.3.1 Experimentos para o Problema da Pequena Filogenia

Experimento 1. Para cada dataset e cada distância (DCJ e DCJ com restrições) o seguinte experimento foi realizado para encontrar os melhores possíveis escores:

- Executar `HELPHY` 50 vezes para o problema da pequena filogenia, usando a reimplementação de Kovac-Opt, e salvar o melhor escore encontrado. A implementação de Kovac-Opt é iterada 10 vezes.
- Executar `HELPHY` 50 vezes para o problema da pequena filogenia, usando Greedy-Opt (Algoritmo 15), e salvar o melhor escore encontrado. O algoritmo Greedy-Opt é iterado 10 vezes.

A Tabela 5.1 mostra o resultado (escore de árvores), nas últimas duas linhas, usando a configuração do Experimento 1 para o dataset *Campanulaceae*. A estrutura da árvore

foi tomada de [19] com um escore de 65 reversões. Note que o software HELPHY, usando a reimplementação de Kovac-Opt, alcançou os mesmos resultados que o software PIVO (para a distância DCJ e DCJ com restrições). Assim, para o dataset *Campanulaceae* a replicação dos resultados foi bem sucedida mesmo quando PIVO representa as filogenias como árvores binarias com raiz. Também, o software HELPHY usando Greedy-Opt alcançou os mesmos resultados que PIVO para a distância DCJ, mas não para a distância DCJ com restrições.

Tabela 5.1: Escores Encontrados por HELPHY para o Problema da Pequena Filogenia usando o Dataset *Campanulaceae*

	Distância Reversão	**DCJ**	**DCJ com restrições**
GRAPPA [57]	67	-	-
MGR [19]	65	-	-
GRAPPA [56]	64	-	-
BADGER [53]	64	-	-
ABC [1]	-	59	64
PIVO [51]	-	59	62
PIVO2 [48]	-	56	59
HELPHY (Kovac-Opt [1])	-	59	62
HELPHY (Greedy-Opt)	-	59	63

A Tabela 5.2 mostra o tempo de comparação para os dois otimizadores (Kovac-Opt e Greedy-Opt) usados por HELPHY depois de 50 execuções, usando o dataset *Campanulaceae* e a distância DCJ. A partir desta tabela, podemos observar que HELPHY com Greedy-opt é 6.72 vezes mais rápido que HELPHY usando Kovac-Opt.

Tabela 5.2: Comparação de Tempo dos Otimizadores usados por HELPHY para o Problema da Pequena Filogenia usando o Dataset *Campanulaceae* (para a distância DCJ)

	Tempo Total	**Tempo Promédio**
HELPHY (Kovac-Opt)	4.64 min	5.57 seg
HELPHY (Greedy-Opt)	0.69 min	0.83 seg

A Tabela 5.3 mostra os resultados (escore de árvores), nas últimas duas linhas, usando a configuração do Experimento 1 para o dataset *Hemiascomycetes*. A estrutura da árvore usada como entrada foi calculada em [51] usando o programa MrBayes [68]. A partir desta tabela podemos observar que HELPHY, usando uma reimplementação de Kovac-Opt, tem menor escore que PIVO2 só com respeito à distância DCJ.

A Tabela 5.4 mostra a comparação de tempo para os dois otimizadores usados por HELPHY depois de 50 execuções, usando o dataset *Hemiascomycetes* e a distância DCJ.

[1]Esta é a reimplementação de Kovac-Opt proposta em [51]

Tabela 5.3: Escores Encontrados por HELPHY para o Problema da Pequena Filogenia usando o Dataset *Hemiascomycetes*

	DCJ	**DCJ com restrições**
PIVO [51]	-	78
PIVO2 [48]	75	77
HELPHY (Kovac-Opt)	74	79
HELPHY (Greedy-Opt)	77	82

A partir desta tabela pode ser observado que HELPHY com Greedy-Opt é 3.95 vezes mais rápido que HELPHY com Kovac-Opt.

Tabela 5.4: Comparação de Tempo dos Otimizadores usados por HELPHY para o Problema da Pequena Filogenia usando o Dataset *Hemiascomycetes* (para a Distância DCJ)

	Tempo Total	**Tempo Promédio**
HELPHY (Kovac-Opt)	2.49 min	2.99 seg
HELPHY (Greedy-Opt)	0.63 min	0.77 seg

5.3.2 Experimentos para o Problema da Grande Filogenia

Os resultados dos experimentos prévios nos deram informação valiosa para escolher o otimizador que vai ser usado como padrão por HELPHY para solucionar ou o problema da grande filogenia ou o problema da pequena filogenia. De fato, o otimizador Greedy-Opt é mais rápido que Kovac-Opt e portanto é a opção que será usada como padrão ao lidar com o problema da grande filogenia, devido a que tem que ser explorado um espaço de busca exponencial formado por estruturas de árvores. No entanto, quando lidamos com o problema da pequena filogenia o principal objetivo é reduzir o escore para uma determinada estrutura de árvore, e neste caso a opção padrão é o otimizador Kovac-Opt. Ambos otimizadores são iterados por padrão dez vezes para o problema da pequena filogenia, e executados só uma vez para o problema da grande filogenia.

Experimento 2. Para explorar o espaço de busca e encontrar uma estrutura com o melhor escore possível, o seguinte experimento foi realizado para cada dataset:

- Executar HELPHY 50 vezes para o problema da grande filogenia e salvar as estrutura das árvores respectivos escores (reversões ou operações DCJ).
- Escolher aquelas estruturas de árvores com os melhores escores.

Experimento 3. Depois de encontrar um conjunto de estruturas de árvores com bons escores, o seguinte experimento foi realizado para cada distância (DCJ ou DCJ com restrições) e para cada aŕvore com o objetivo de reduzir seus escores:

- Executar HELPHY 50 vezes para o problema da pequena filogenia e salvar o melhor escore encontrado (DCJ ou DCJ com restrições).

A Tabela 5.5 mostra o resultado dos Experimentos 2 e 3 para o dataset *Campanulaceae*. Na segunda coluna aparece o escore das melhores árvores encontradas (com 64 e 65 reversões) para o problema da filogenia grande (Experimento 2). A terceira e quarta colunas mostram os escores (DCJ e DCJ com restrições) encontrados por HELPHY para o Experimento 3. Note que duas árvores (Arv32 e Arv33) foram encontradas tendo os melhores escores: 64 reversões, 59 operações DCJ, e 62 operações DCJ com restrições. Estes resultados são iguais àqueles encontrados pelo software PIVO usando a estrutura da árvore encontrada pelo software MGR [19].

Tabela 5.5: Árvores (e Escores) Encontrados por HELPHY para o Problema da Grande (e Pequena) Filogenia usando o Dataset *Campanulaceae*

	Grande Filogenia (Reversões)	**Pequena Filogenia (DCJ)**	**Pequena Filogenia (DCJ com restrições)**
Arv32	**64**	**59**	**62**
Arv33	**64**	**59**	**62**
Arv40	**64**	**59**	63
Arv1	65	**59**	**62**
Arv11	65	61	63
Arv12	65	60	65
Arv18	65	61	**62**
Arv26	65	61	63
Arv27	65	61	63
Arv28	65	61	63
Arv38	65	61	63
Arv6	65	61	63

A Tabela 5.6 mostra o tempo de execução do software HELPHY para os experimentos realizados para gerar a Tabela 5.5. O software HELPHY demorou só 1.27 minutos para encontrar três árvores com um escore de 64 reversões e nove árvores com um escore de 65 reversões. Quando lidamos com a distância de reversão o software HELPHY usa o algoritmo que iterativamente soluciona instâncias do problema da mediana para encontrar o escore de uma determinada árvore. A rotina para solucionar o problema da mediana para reversões foi tomada do software GRAPPA. Para o problema da pequena filogenia o software HELPHY demorou 66.04 minutos e 70.52 minutos para a distância DCJ e DCJ com restrições respectivamente. O tempo total gasto para analisar o dataset *Campanulaceae* foi de 137.83 minutos (2.29 horas).

A Tabela 5.7 mostra os resultados do Experimento 2 e 3 para o dataset *Hemiascomycetes*. Na segunda coluna podemos observar o escore das melhores árvores encontradas

Tabela 5.6: Tempo de Execução de HELPHY para os Problemas da Grande e Pequena Filogenia usando o Dataset *Campanulaceae*

	Tempo (minutos)
Grande Filogenia (Reversões)	1.27
Pequena Filogenia (DCJ)	66.04
Pequena Filogenia (DCJ com restrições)	70.52
Total	137.83

(76, 77, e 78 operações DCJ) para o problema da grande filogenia (Experimento 2). A terceira e quarta colunas mostram o escore (DCJ e DCJ com restrições) encontrados por HELPHY para o problema da pequena filogenia (Experimento 3). Foi encontrada uma árvore (Arv4) com os seguintes melhores escores: 73 operações DCJ e 76 operações DCJ com restrições (ver Figura 5.1). Esta árvore tem melhores escores que aquelas árvores encontradas por PIVO2 (e PIVO) para a estrutura da árvore encontrada usando MrBayes.

Tabela 5.7: Árvores (e Escores) Encontrados por HELPHY para o Problema da Grande (e Pequena) Filogenia usando o Dataset *Hemiascomycetes*

	Grande Filogenia (DCJ)	**Pequena Filogenia (DCJ)**	**Pequena Filogenia (DCJ com restrições)**
Arv48	**76**	76	82
Arv5	77	76	84
Arv20	77	75	77
Arv30	77	**73**	80
Arv4	78	**73**	**76**
Arv11	78	76	79
Arv22	78	74	80
Arv42	78	75	78
Arv44	78	76	79

A Tabela 5.8 mostra o tempo de execução de HELPHY para os experimentos realizados para gerar a Tabela 5.7. HELPHY demorou 38.37 minutos para encontrar uma árvore com escore de 76 operações DCJ, três árvores com escore de 77 operações DCJ, e cinco árvores com escore de 78 operações DCJ. Para o caso do problema da pequena filogenia, HELPHY demorou 19.34 minutos e 20.73 minutos para as distâncias DCJ e DCJ com restrições respectivamente. O tempo total gasto para analisar o dataset *Hemiascomycetes* foi de 78.44 minutos (1.31 horas).

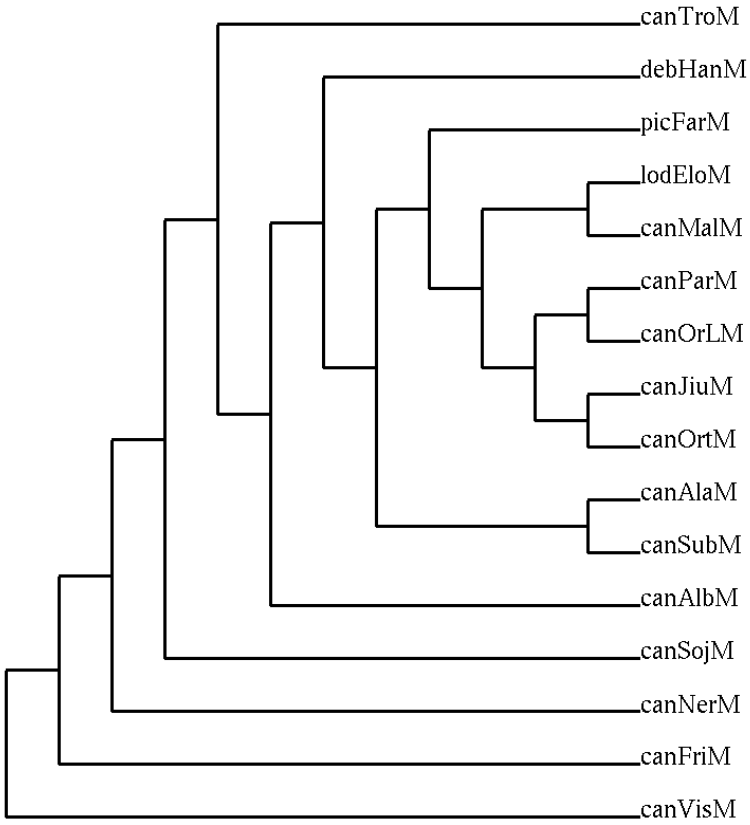

Figura 5.1: Árvore filogenética do dataset *Hemiascomycetes* (Arv4 da Tabela 5.7) encontrada pelo software `HELPHY`.

Tabela 5.8: Tempo de Execução de `HELPHY` para os Problemas da Grande e Pequena Filogenia usando o Dataset *Hemiascomycetes*

	Tempo (minutos)
Grande Filogenia (DCJ)	38.37
Pequena Filogenia (DCJ)	19.34
Pequena Filogenia (DCJ com restrições)	20.73
Total	78.44

Sumário do Capítulo:

Neste capítulo foram apresentados dois algoritmos para os problemas da pequena e grande filogenia. A primeira abordagem é gulosa e é usada para resolver o problema da pequena filogenia. Experimentos mostraram que esta abordagem tem tempos de execução menores que a abordagem de programação dinâmica proposta por Kováč et al. [51], mas perde precisão nos resultados. A segunda abordagem é baseada na meta-heurística de busca em vizinhança, para resolver o problema da grande filogenia. Experimentos com o dataset *Campanulaceae* mostraram que podem ser encontradas árvores filogenéticas com bons escores (distância de reversão) como os encontrados na literatura, em só 1.27 minutos. Logo, para o caso do dataset *Hemiascomycetes*, experimentos deram como resultado uma nova árvore diferente daquela usada na literatura e com um melhor escore.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

Em este trabalho foram propostos novos algoritmos evolutivos para o problema de ordenação de permutações (sem sinal) por reversões: especificamente foram propostos algoritmos paralelos (AG-Paralelo), AG com busca local (AM), AM com busca por oposição (AMBO), e AM híbrido com busca por oposição (AMBO-Híbrido). Também foi proposto um software chamado `HELPHY` para construir árvores filogenéticas usando como entrada dados baseados na ordem dos genes (permutações com sinal). Nesse software foi implementada uma abordagem gulosa para o problema da pequena filogenia, e uma abordagem baseada em busca em vizinhança variável para o problema da grande filogenia.

Com respeito aos algoritmos evolutivos para o problema de ordenação de permutações por reversões temos o seguinte: o AG-Paralelo usa um modelo de execuções independentes (*independet runs*, ver [80]); a busca local do AM é embutida nas etapas de geração e reinício da população; a heurística de busca por oposição do AMBO, também é embutida nas etapas de geração e reinício da população. A última heurística usada em AMBO-Híbrido, consiste numa fase de pré-processamento da permutação de entrada onde são aplicadas reversões que eliminam dois pontos de quebra até que não possa ser mais aplicada esta heurística.

Experimentos foram realizados com permutações geradas de diferentes formas, e para que as comparações entre algoritmos sejam justas os mesmos recursos computacionais foram atribuídos a todos os algoritmos: o mesmo tamanho de população e o mesmo número de avaliações da função de aptidão.

Dos experimentos usando conjuntos de cem permutações (geradas aleatoriamente) temos que AMBO e AMBO-Híbrido têm os melhores resultados para permutações de comprimento até 120, estas instâncias podem ser consideradas como casos práticos, visto que o maior número de genes mitocondriais que foi encontrado é tamanho 97 que se

corresponde com o DNA mitocondrial do protozoário *Reclinomonas americana* (ver [28]). Para permutações de comprimento ≥ 130, os melhores resultados foram obtidos por AG-SA e AG-Híbrido.

Dos experimentos com permutações benchmarks temos que os resultados confirmaram aqueles obtidos com conjuntos de cem permutações: AMBO e AMBO-Híbrido são a melhor escolha para casos práticos. Para os benchmarks de comprimento 50 (1RPL50 e 2RPL50), AMBO-Híbrido e AMBO mostraram os melhores resultados para a média, respectivamente. Para os benchmarks de comprimento 100 (1RPL100 e 2RPL100) AMBO mostrou os melhores resultados para a média, respectivamente. Para os benchmarks de comprimento 150 (1RPL150 e 2RPL150), AG-Híbrido e AG-SA mostraram os melhores resultados para a média.

Dos experimentos com permutações baseadas em dados biológicos temos que para todos os casos AMBO teve os melhores resultados, em 5 de 9 casos todos os algoritmos tiveram os mesmos resultados, e em 3 de 9 casos AG-SA não teve os melhores resultados em alguma das medidas.

Tomando como base todos os resultados dos experimentos podemos concluir que AMBO e AMBO-Híbrido são algoritmos adequados para calcular a distância de reversão de permutações sem sinal de comprimento até 120, os quais podem ser considerados casos de interesse prático. Assim, já que sabemos o tamanho da entrada a priori, o método que melhor se ajuste aos nossos interesses pode ser escolhido. Também, os resultados dos experimentos nos quais usamos os mesmos recursos para todos os algoritmos, mostraram que o AG-Paralelo só fornece melhores resultados quando o tamanho da população é incrementado (ver [79, 74]), o que deveria acontecer com todos os algoritmos. De fato, o AG-Paralelo usa um modelo de execuções independentes ou modelo de ilha sem migração cuja contribuição real é restrita a melhorar o speed-up da sua versão sequencial (AG-SA), como visto na discussão do Capitulo 4. Um próximo passo seria explorar outros modelos de paralelização como aqueles com políticas de migração sobre diferentes topologias (ver [80]), não somente para o AG-Paralelo mas também para versões paralelas de AMBO e AMBO-Híbrido.

Com respeito à construção de aŕvores filogenéticas, dois casos importantes da literatura foram abordados: os datasets *Campanulaceae* e *Hemiascomycetes*. Os resultados dos experimentos mostraram que a abordagem gulosa melhora o tempo de execução com respeito à abordagem por programação dinâmica (proposta por Kováč et al. [51]), mas perde acurácia. Baseada nestes resultados a abordagem gulosa foi usada como padrão (caso da distância DCJ) para avaliar o custo das árvores para o problema da grande filogenia, e a abordagem de programação dinâmica foi usada como padrão (caso da distância DCJ) para o problema da pequena filogenia. Para o caso da distância de reversão foi

usado o algoritmo iterativo apresentado por Blanchette et al. [16] para avaliar o custo da estrutura de uma árvore, o qual usa como sub-rotina o algoritmo proposto por Caprara para o problema da mediana (tomado do software `GRAPPA`).

O software `HELPHY` mostrou que é adequado para analisar datasets usando as distâncias de revesão e DCJ. No caso do dataset *Campanulaceae* o tempo de execução foi melhorado a só 1.27 minutos para descobrir três árvores de escore 64 reversões e nove árvores de escore 65 reversões, já que o melhor tempo de execução encontrado na literatura [56] foi de uma hora para encontrar várias árvores de escore 64 reversões. Para o caso do dataset *Hemiascomycetes*, tomou um tempo de execução razoável de 38.37 minutos para encontrar uma árvore de escore 76 operações DCJ, três árvores de escore 77 operações DCJ, e cinco árvores de escore 78 operações DCJ. No entanto, para reduzir estes escores foi necessário executar `HELPHY` para o problema da pequena filogenia, incrementando o tempo de execução total a 1.31 horas, mas melhorando o escore das árvores, com uma delas tendo um escore de 73 operações DCJ (e 76 operações DCJ restritas); esta estrutura encontrada para o dataset *Hemiascomycetes* é diferente (e com melhores escores) daquela encontrada pelo programa `MrBayes` em [48] com escores de 75 operações DCJ (e 77 operações DCJ restritas) encontrados pelo software `PIVO2` [48].

6.2 Trabalhos Futuros

Como trabalho futuro incluiremos melhorias na etapa de busca local dos algoritmos evolutivos para o problema de ordenação de permutações por reversões, como adaptar dinamicamente a sua computação de acordo com o progresso das gerações; de modo que este processo seja computacionalmente menos intenso. Também, pesquisaremos sobre o operador de oposição do tipo-II, que é uma técnica que explora os opostos da função de aptidão. Ademais, como mencionado anteriormente é importante investigar outras topologias e politicas de migração para os algoritmos paralelos.

Com respeito à construção de árvores filogenéticas, é interessante implementar outros algoritmos evolutivos como algoritmos genéticos, algoritmos meméticos, ou recozimento simulado (*simulated annealing*). Para estas abordagens evolutivas, o algoritmo guloso para o problema da pequena filogenia (Algoritmo 15) poderia ser usado como função de aptidão para avaliar o custo da estrutura de uma árvore. Também, é importante implementar abordagens paralelas para o algoritmo de busca em vizinhança variável como os que foram comentados em [46], isto vai reduzir o tempo total para analisar datasets para o problema da grande filogenia. Finalmente, é interessante testar o software `HELPHY` com outros datasets, baseados na ordem dos genes, usando a distância de reversão e DCJ.

Referências

[1] Zaky Adam and David Sankoff. The abcs of mgr with dcj. *Evolutionary Bioinformatics*, 4, 2008. 39, 77

[2] F.S. AlQunaieer, H.R. Tizhoosh, and S. Rahnamayan. Opposition based computing a survey. In *Proc. IEEE Int. Conf. Neural Networks*, pages 1098–7576, Barcelona, Spain, 2010. 47

[3] Benjamin L Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of combinatorics*, 5(1):1–15, 2001. 76

[4] Alexandre A Andreatta and Celso C Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8(4):429–447, 2002. 73, 74, 75

[5] A. Auyeung and A. Abraham. Estimating genome reversal distance by genetic algorithm. In *Congress on Evolutionary Computation. CEC'03.*, volume 2, pages 1157–1161. IEEE, 2003. 13, 14, 41, 55

[6] Andy Auyeung and Ajith Abraham. Estimating genome reversal distance by genetic algorithm. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1157–1161. IEEE, 2003. 3

[7] David A. Bader, Bernard M. E. Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. In *WADS*, volume 2125, pages 365–376, 2001. 14, 41

[8] David A Bader, Bernard ME Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001. 3, 38

[9] V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 148 –157, 1993. 1, 2, 6, 13, 15, 42, 44, 66

[10] Anne Bergeron. *A Very Elementary Presentation of the Hannenhalli-Pevzner Theory*, volume 2089 of *Lecture Notes in Computer Science*, chapter chapter 9, pages 106–117. Springer Berlin Heidelberg, 2001. 10, 14

[11] Anne Bergeron, Julia Mixtacki, and Jens Stoye. On sorting by translocations. *Journal of Computational Biology*, 13(2):567–578, 2006. 70

[12] Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In *Algorithms in Bioinformatics*, pages 163–173. Springer, 2006. 2, 16, 19, 20, 39

[13] Piotr Berman and Sridhar Hannenhalli. Fast sorting by reversal. In *CPM*, volume 1075, pages 168–185, 1996. 14

[14] Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. *1.375-approximation algorithm for sorting by reversals*. Springer, 2002. 3

[15] Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 200–210, 2002. 13

[16] M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In *Genome informatics. Workshop on Genome Informatics*, volume 8, pages 25–34, 1997. 85

[17] Mathieu Blanchette, Guillaume Bourque, and David Sankoff. Breakpoint phylogenies. *Genome Informatics*, 8:25–34, 1997. 38, 73

[18] Mathieu Blanchette, Takashi Kunisawa, and David Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203, 1999. 38, 73

[19] Guillaume Bourque and Pavel A Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome research*, 12(1):26–36, 2002. 2, 38, 39, 40, 77, 79

[20] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998. 51

[21] Alberto Caprara. Sorting by reversals is difficult. In *Proceedings of the first annual international conference on Computational molecular biology*, pages 75–83. ACM, 1997. 3, 13, 21

[22] Alberto Caprara. Formulations and hardness of multiple sorting by reversals. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 84–93. ACM, 1999. 20, 27, 30, 31, 32, 33, 34, 35, 36, 37

[23] Alberto Caprara. On the practical solution of the reversal median problem. In *International Workshop on Algorithms in Bioinformatics*, pages 238–251. Springer, 2001. 38

[24] Fabio Caraffini, Ferrante Neri, and Lorenzo Picinali. An analysis on separability for memetic computing automatic design. *Information Sciences*, 265:1–22, 2014. 45

[25] Xianshun Chen, Yew-Soon Ong, Meng-Hiot Lim, and Kay Chen Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011. 45

[26] David A Christie. A 3/2-approximation algorithm for sorting by reversals. In *SODA*, pages 244–252, 1998. 3

[27] David A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, SODA '98, pages 244–252. Society for Industrial and Applied Mathematics, 1998. 6, 13, 15, 44, 56

[28] Geoffrey M Cooper. Mitochondria. In *The Cell: A Molecular Approach. 2nd edition*. Sunderland (MA): Sinauer Associates, 2000. Available from: http://www.ncbi.nlm.nih.gov/books/NBK9896/. 84

[29] Mary E Cosner, Robert K Jansen, Bernard ME Moret, Linda A Raubeson, Li-San Wang, Tandy Warnow, and Stacia Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae. In *Comparative Genomics*, pages 99–121. Springer, 2000. 76

[30] Mary E Cosner, Robert K Jansen, Bernard ME Moret, Linda A Raubeson, Li-San Wang, Tandy Warnow, Stacia K Wyman, et al. A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data. In *ISMB*, pages 104–115, 2000. 76

[31] Carlos Cotta and Pablo Moscato. Inferring phylogenetic trees using evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 720–729. Springer, 2002. 76

[32] Yun Cui, Lusheng Wang, Daming Zhu, and Xiaowen Liu. A (1.5+ ε)-approximation algorithm for unsigned translocation distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 5(1):56–66, 2008. 70

[33] Lucas A da Silveira, José L Soncco-Alvarez, and Mauricio Ayala-Rincón. Parallel memetic genetic algorithms for sorting unsigned genomes by translocations. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 185–192. IEEE, 2016. 69, 70

[34] Lucas A Da Silveira, José L Soncco-Alvarez, Thaynara A de Lima, and Mauricio Ayala-Rincón. Computing translocation distance by a genetic algorithm. In *Computing Conference (CLEI), 2015 Latin American*, pages 1–12. IEEE, 2015. 69, 70

[35] Lucas A da Silveira, José L Soncco-Álvarez, Thaynara A de Lima, and Mauricio Ayala-Rincón. Memetic and opposition-based learning genetic algorithms for sorting unsigned genomes by translocations. In *Advances in Nature and Biologically Inspired Computing*, pages 73–85. Springer, 2016. 69, 70

[36] Marco A Montes de Oca, Carlos Cotta, and Ferrante Neri. Local search. In *Handbook of Memetic Algorithms*, pages 29–41. Springer, 2012. 46, 74, 75

[37] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006. 57

[38] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011. 57

[39] Gary B Fogel. Evolutionary computation for the inference of natural evolutionary histories. *IEEE Connections*, 3(1):11–14, 2005. 73

[40] José E Gallardo, Carlos Cotta, and Antonio J Fernández. Reconstructing phylogenies with memetic algorithms and branch-and-bound., 2007. 73

[41] Nan Gao, Ning Yang, and Jijun Tang. Ancestral genome inference using a genetic algorithm approach. *PloS one*, 8(5):e62156, 2013. 39

[42] Salvador Garcıa and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008. 57

[43] Ahmadreza Ghaffarizadeh, Kamilia Ahmadi, and Nicholas S. Flann. Sorting unsigned permutations by reversals using multi-objective evolutionary algorithms with variable size individuals. In *IEEE Congress of Evolutionary Computation (CEC)*, pages 292–295. IEEE, Jun 2011. 13, 14

[44] Sridhar Hannenhalli and Pavel Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 178–189. ACM, 1995. 3, 33

[45] Sridhar Hannenhalli and Pavel Pevzner. Transforming cabbage into turnip, polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing. STOC'95.*, pages 178–189. ACM Press, 1995. 6, 10, 14

[46] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, pages 1–32, 2016. 73, 74, 85

[47] Jin-Kao Hao. Memetic algorithms in discrete optimization. In *Handbook of memetic algorithms*, pages 73–94. Springer, 2012. 45

[48] Albert Herencsár and Broňa Brejová. An Improved Algorithm for Ancestral Gene Order Reconstruction. In V. Kurkova, L. Bajer, and V. Svatek, editors, *Information Technologies - Applications and Theory (ITAT)*, number 1003 in CEUR-WS, pages 46–53, Jasna, Slovakia, 2014. 40, 77, 78, 85

[49] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892, Jan. 2000. 14

[50] John Kececioglu and David Sankoff. *Exact and approximation algorithms for the inversion distance between two chromosomes*, volume 684 of *Lecture Notes in Computer Science*, chapter chapter 8, pages 87–105. Springer-Verlag, 1993. 6, 13, 64

[51] Jakub Kováč, Broňa Brejová, and Tomáš Vinař. A practical algorithm for ancestral rearrangement reconstruction. In *International Workshop on Algorithms in Bioinformatics*, pages 163–174. Springer, 2011. 3, 4, 39, 72, 77, 78, 82, 84

[52] Natalio Krasnogor, Alberto Aragón, and Joaquín Pacheco. *Memetic Algorithms*, volume 36 of *Operations Research/Computer Science Interfaces Series*, chapter chapter 11, pages 225–248. Springer US, 2006. 45

[53] Bret Larget, Joseph B Kadane, and Donald L Simon. A bayesian approach to the estimation of ancestral genome arrangements. *Molecular phylogenetics and evolution*, 36(2):214–223, 2005. 39, 77

[54] Hideo Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. In *Pacific symposium on biocomputing*, volume 96, pages 512–523. Citeseer, 1996. 73

[55] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. 73

[56] Bernard ME Moret, Adam C Siepel, Jijun Tang, and Tao Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *International Workshop on Algorithms in Bioinformatics*, pages 521–536. Springer, 2002. 38, 77, 85

[57] Bernard ME Moret, Li-San Wang, Tandy Warnow, and Stacia K Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(suppl 1):S165–S173, 2001. 73, 77

[58] BME Moret, SK Wyman, DA Bader, T Warnow, and M Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp. on Biocomputing (PSB01)*, number LCBB-CONF-2001-007, pages 583–594. World Scientific Pub., 2001. 38, 73

[59] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Technical report, Caltech Concurrent Computation Program 158-79, 1989. 45

[60] Pablo Moscato and Carlos Cotta. *A gentle introduction to memetic algorithms*, volume 57 of *International Series in Operations Research & Management Science*, chapter chapter 5, pages 105–144. Kluwer Academic Publishers, 2003. 45

[61] Ferrante Neri and Carlos Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012. 45

[62] Itsik Pe'er and Ron Shamir. The median problems for breakpoints are np-complete. In *Elec. Colloq. on Comput. Complexity*, volume 71. Citeseer, 1998. 20

[63] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy Salama. Opposition-based differential evolution. *Evolutionary Computation, IEEE Transactions on*, 12(1):64–79, 2008. 49

[64] Celso C Ribeiro and Dalessandro S Vianna. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *International Transactions in Operational Research*, 16(5):641–657, 2009. 75

[65] Celso C Ribeiro and Dalessandro Soares Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In *WOB*, pages 97–102, 2003. 75

[66] Jean-Michel Richer, Adrien Goëffon, and Jin-Kao Hao. A memetic algorithm for phylogenetic reconstruction with maximum parsimony. In *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, pages 164–175. Springer, 2009. 73

[67] Jean-Michel Richer, Eduardo Rodriguez-Tello, and Karla E Vazquez-Ortiz. Maximum parsimony phylogenetic inference using simulated annealing. In *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, pages 189–203. Springer, 2013. 75, 76

[68] Fredrik Ronquist and John P. Huelsenbeck. Mrbayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003. 40, 77

[69] H. Salehinejad, S. Rahnamayan, and H.R. Tizhoosh. Type-ii opposition-based differential evolution. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1768–1775, July 2014. 48

[70] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. F. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences of the United States of America*, 89(14):6575–6579, 1992. 66

[71] David Sankoff, Robert J Cedergren, and Guy Lapalme. Frequency of insertion-deletion, transversion, and transition in the evolution of 5s ribosomal rna. *Journal of Molecular Evolution*, 7(2):133–149, 1976. 38

[72] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, pages 379–427, 1948. 46

[73] S.N. Sivanandam and S.N. Deepa. *Introduction to genetic algorithms*. 2007. 41

[74] José Luis Soncco-Álvarez, Gabriel Marchesan Almeida, Juergen Becker, and Mauricio Ayala-Rincón. Parallelization of genetic algorithms for sorting permutations by reversals over biological data. *International Journal of Hybrid Intelligent Systems*, 12(1):53–64, 2015. 4, 51, 52, 67, 84

[75] José Luis Soncco-Álvarez and Mauricio Ayala-Rincón. A genetic approach with a simple fitness function for sorting unsigned permutations by reversals. In *Computing Congress (CCC), 2012 7th Colombian*, pages 1–6. IEEE, 2012. 13, 14, 55, 56

[76] José Luis Soncco-Álvarez and Mauricio Ayala-Rincón. Sorting permutations by reversals through a hybrid genetic algorithm based on breakpoint elimination and exact solutions for signed permutations. *Electronic Notes in Theoretical Computer Science*, 292:119–133, 2013. 3, 4, 13, 15, 43, 44, 50, 53, 54, 55, 56

[77] Jose Luis Soncco-Alvarez and Mauricio Ayala-Rincon. Memetic algorithm for sorting unsigned permutations by reversals. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2770–2777. IEEE, 2014. 4, 54, 55, 60, 64, 67

[78] José Luis Soncco-Álvarez and Mauricio Ayala-Rincón. Variable neighborhood search for the large phylogeny problem using gene order data. In *CEC 2017*, 2017. Aceito para apresentação e publicação. 4

[79] José Luis Soncco-Alvarez, G Marchesan Almeida, Jürgen Becker, and Mauricio Ayala-Rincon. Parallelization and virtualization of genetic algorithms for sorting permutations by reversals. In *Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on*, pages 29–35. IEEE, 2013. 4, 13, 51, 52, 55, 67, 84

[80] Dirk Sudholt. *Springer Handbook of Computational Intelligence*, chapter Parallel Evolutionary Algorithms, pages 929–959. Springer Berlin Heidelberg, 2015. 51, 67, 83, 84

[81] Krister Swenson, Vaibhav Rajan, Yu Lin, and Bernard Moret. Sorting signed permutations by inversions in o (nlogn) time. In Serafim Batzoglou, editor, *Research in Computational Molecular Biology*, volume 5541, pages 386–399. 2009. 14

[82] Eric Tannier, Anne Bergeron, and Marie-France Sagot. Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, Apr. 2007. 14

[83] Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal genome median and halving problems. In *Algorithms in Bioinformatics*, pages 1–13. Springer, 2008. 20, 21

[84] Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC bioinformatics*, 10(1):120, 2009. 20, 21

[85] H.R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 1, pages 695–701, Nov 2005. 49

[86] H.R. Tizhoosh and Mario Ventresca. *Oppositional Concepts in Computational Intelligence*. Springer Berlin Heidelberg, 2008. 47

[87] Matus Valach, Zoltan Farkas, Dominika Fricova, Jakub Kovac, Brona Brejova, Tomas Vinar, Ilona Pfeiffer, Judit Kucsera, Lubomir Tomaska, B Franz Lang, et al. Evolution of linear chromosomes and multipartite genomes in yeast mitochondria. *Nucleic acids research*, page gkq1345, 2011. 76

[88] Michael S Waterman and Temple F Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73(4):789–800, 1978. 75

[89] Andrew Wei Xu and David Sankoff. Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem. In *Algorithms in Bioinformatics*, pages 25–37. Springer, 2008. 39

[90] Qingzheng Xu, Lei Wang, Na Wang, Xinhong Hei, and Li Zhao. A review of opposition-based learning from 2005 to 2012. *Engineering Applications of Artificial Intelligence*, 29:1–12, Mar. 2014. 49

[91] Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005. 2, 39

[92] Daming Zhu and Lusheng Wang. On the complexity of unsigned translocation distance. *Theoretical computer science*, 352(1-3):322–328, 2006. 70