Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graducação em Informática

Ordenação por Translocação de Genomas sem Sinal Utilizando Algoritmos Genéticos

Lucas Ângelo da Silveira

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Mauricio Ayala Rincón

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-graduação em Informática

Coordenadora: Profa. Dra Célia Ghedini Ralha

Banca examinadora composta por:

Prof. Dr. Mauricio Ayala Rincón (Orientador) — CIC/UnB
Prof. Dr. George Luiz Medeiros Teodoro — CIC/UnB
Prof. Dr. Guilherme Pimentel Telles — Unicamp

CIP — Catalogação Internacional na Publicação

Ângelo da Silveira, Lucas.

Ordenação por Translocação de Genomas sem Sinal Utilizando Algoritmos Genéticos / Lucas Ângelo da Silveira. Brasília : UnB, 2016.
126 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2016.

1. Distância por Translocações, 2. Distância entre Genomas Com e Sem Sinal, 3. Algoritmos de Aproximação, 4. Algoritmos Genéticos, 5. Paralelismo.

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graducação em Informática

Ordenação por Translocação de Genomas sem Sinal Utilizando Algoritmos Genéticos

Lucas Ângelo da Silveira

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

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

Prof. Dr. George Luiz Medeiros Teodoro
CIC/UnB

Prof. Dr. Guilherme Pimentel Telles
Unicamp

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

Brasília, 29 de fevereiro de 2016

Resumo

Translocações são usadas para mensurar a distância evolutiva entre espécies. Do ponto de vista biológico dois tipos de genomas tem recebido atenção: genomas com e sem sinal. Ao considerar genomas com sinal, computar a distância mínima de translocações é linear enquanto que o caso sem sinal é $\mathcal{NP}$-difícil.

Propõem-se algoritmos genéticos (AGs) para resolver o problema de distância de translocação entre genomas sem sinal. A abordagem, consiste em utilizar uma população composta por indivíduos representando genomas com sinal obtidos de um genoma sem sinal provido como entrada. A solução de cada indivíduo é também uma solução admissível para o genoma dado. A *função de aptidão* utilizada, que é a distância para genomas com sinal, é computada linearmente com um algoritmo proposto por Bergeron et al.

O AG baseado nessa abordagem foi aprimorado com duas técnicas de otimização: memética e aprendizagem baseada em oposição. Além disso, foram propostas paralelizações do AG memético buscando diminuir o tempo de processamento assim como melhorar a precisão .

A qualidade dos resultados foi validada utilizando uma implementação de um algoritmo de raio de aproximação $1.5+\varepsilon$ recentemente proposto por Cui et al.

Experimentos foram realizados tomando como entrada genomas sintéticos e gerados a partir de dados biológicos. Os AGs forneceram melhores resultados que o algoritmo de controle de qualidade. As paralelizações apresentaram melhoras tanto no tempo de execução quanto na precisão dos resultados.

Utilizou-se o teste de hipóteses de Wilcoxon a fim de verificar a significância estatística das melhorias fornecidas pelos AGs aprimorados em relação àquelas fornecidas pelo AG básico. Desta análise foi possível identificar que o AG memético provê resultados diferentes (melhores) que o AG básico, e que este último e o AG com aprendizagem baseada em oposição não apresentam nenhuma diferença significativa. O teste foi também aplicado para comparar as soluções das paralelizações confirmando que existem aprimoramentos dos resultados comparados com o AG memético.

Palavras-chave: Distância por Translocações, Distância entre Genomas Com e Sem Sinal, Algoritmos de Aproximação, Algoritmos Genéticos, Paralelismo.

Abstract

Translocations are used to measure the evolutionary distance between species. From a biological point of view two types of genomes have received attention: signed and unsigned genomes. When considering signed genomes, the problem can be solved in linear time, while, in the case of unsigned genomes the problem was shown to be $\mathcal{NP}$-hard.

Genetic algorithms (GAs) are proposed to solve the translocation distance problem between unsigned genomes. The approach consists in using a population composed of individuals representing signed genomes obtained from a given unsigned genome provided as input. The solution of each individual is also an admissible solution to the given genome. The fitness function used, which is the distance for signed genome, is computed linearly with an algorithm proposed by Bergeron et al.

The GA based on this approach has been enhanced with two optimization techniques: memetic and opposition based learning. Also, parallelizations of the GA embedded with memetic were proposed seeking to improve both running time as the accuracy of results.

The quality of the results was verified using an implementation of a $1.5+\varepsilon$-approximation algorithm recently proposed by Cui et al.

Experiments were performed taking as input synthetic genomes and genomes generated from biological data. The GAs provided better results than the quality control algorithm. The parallelizations showed improvements both regarding runtime as well as accuracy.

A statistical analysis based on the Wilcoxon test was performed to check if the improvements in the solutions provided by enhanced GAs compared to those provided by the basic GA have some significance. This analysis can identify that the GA embedded with the technical memetic provides different (better) results than GA and that the results provided by the GA embedded with opposition based learning presents no significant difference. The test was also performed to compare the solutions of the parallelizations confirming that there are improvements of the results regarding the GA embedded memetic.

Keywords: Translocation Distance, Distance between Signed and Unsigned Genomes, Approximation Algorithms, Genetic Algorithms, Parallelism.

Sumário

Capítulo 1

Introdução

1.1 Descrição do Problema

Um genoma de um organismo é constituído de toda a informação hereditária que está codificada em seu DNA (se tratando de alguns vírus tal informação se encontra no RNA). Isto inclui tanto diferentes genes como as sequências não-codificadoras que são de grande relevância para a regulação gênica, dentre outras funções. A partir de genomas de diferentes espécies pode-se mensurar a distância evolutiva que separa uma espécie de outra.

O estudo da distância evolutiva entre duas espécies usando dados genéticos requer a reconstrução da sequência de eventos evolutivos que transformam um genoma em outro. Atualmente os mecanismos de rearranjos mais utilizados incluem operações globais como reversão, transposição, duplicação e translocação, e a tarefa básica é: dados dois genomas, encontrar a sequência mais curta de operações de rearranjo que transforma um genoma em outro. Para este trabalho, considera-se translocação, que consiste em: dado dois cromossomos, particionar ambos os cromossomos em um ponto aleatório e combinar os segmentos formados para construir dois novos cromossomos [30].

1.2 Motivação

Comparações de sequências biológicas são problemas relevante em bioinformática para determinar as relações evolutivas entre organismos. Este problema pode ser resolvido utilizando algoritmos que levam em consideração mutações locais (deleções, inserções), como por exemplo o algoritmo clássico de programação dinâmica para alinhar duas sequências de DNA. Porém, quando se tenta entender como as sequências genéticas sofrem mutações a nível cromossômico é necessário considerar operações globais como reversões, translocações e transposições.

Ao falar do estudo de distância evolutiva entre espécies dois tipos de genomas tem recebido atenção do ponto de vista biológico: genomas com e sem sinal. Para genomas sem sinal, os genes são abstraídos sem qualquer orientação, enquanto que em genomas com sinal, cada gene tem um sinal positivo ou negativo refletindo sua orientação dentro do genoma.

Dados dois genomas, o número mínimo de translocações para transformar um genoma em outro é conhecido como distância de translocação, que é o mesmo que encontrar a sequência mínima de translocações necessária para transformar um genoma em outro. O problema é conhecido na literatura como *ordenação por translocação*. Considerando os dois tipos de genomas, tem-se dois problemas de complexidades diferentes. No caso de genomas sem sinal, o problema foi demostrado ser $\mathcal{NP}$-difícil por D. Zhu et al. em [50]. Para genomas com sinal, existem algoritmos que fornecem tanto a distância quanto a sequência em tempo polinomial ([48], [5]).

No estudo do problema de distância de translocação entre genomas uma estrutura de suma importância é o grafo de Pontos-de-Quebra, que será definido no Capítulo 2. Tal grafo é construído a partir das relações de adjacência entre dois genomas. A distância de translocação está intimamente relacionada com o número de ciclos deste grafo. Para genomas com sinal a decomposição em ciclos deste grafo é única. Em contrapartida, isto não ocorre para genomas sem sinal. Esta observação é a base para compreender porque o problema de distância entre genomas com sinal é de complexidade polinomial e utilizando genomas sem sinal é $\mathcal{NP}$-difícil. Após demonstrar a dificuldade do problema considerando genomas na versão sem sinal, vários algoritmos aproximados foram propostos. Em sua essência, algoritmos aproximados são algoritmos que não necessariamente produzem uma solução ótima, porém soluções que estão demonstradas estar dentro de um certo raio da solução ótima, fato este provado matematicamente.

Como contribuição, neste trabalho é apresentada a compilação da prova de que o problema de distância de translocação entre genomas sem sinal (DTS) é $\mathcal{NP}$-difícil expondo todos os passos necessários para fornecer uma prova detalhada bem como um conjunto apropriado de algoritmos genéticos (AGs) para resolver o problema. O primeiro algoritmo evolucionário denotado como AG (publicado em [42]), proposto para a resolução do problema DTS, tem como enfoque mapear um dado genoma sem sinal de tamanho n fornecido como entrada (com n representado a quantidade de genes) em um subconjunto das 2^n possíveis versões com sinal. Isto é feito assinalando aleatoriamente um sinal positivo ou negativo para cada gene do genoma fornecido na entrada. Todavia nem sempre é possível explorar todo esse espaço de busca, assim apenas um subconjunto de tamanho $n \log n$ contendo genomas com sinal constitui a população utilizada no AG. A solução de cada um destes indivíduos da população corresponde a uma solução admissível para o genoma

sem sinal provido na entrada. A função de aptidão referente aos indivíduos na população é dada como a distância exata provida em tempo linear pelo algoritmo proposto em [5] para o problema envolvendo genomas com sinal. Em um segundo estágio, melhorias foram adicionadas ao AG e dois novos algoritmos evolucionários foram propostos (publicados em [43]). O primeiro algoritmo incorpora estágios contendo a técnica memético e o segundo, estágios contendo a técnica conhecida na literatura como aprendizado baseado em oposição, e são denotados neste trabalho respectivamente como AM e $OBL\text{-}AG$. Como mecanismo de controle de qualidade para as soluções providas pelos AGs, o algoritmo aproximado de raio $1.5 + \varepsilon$ fornecido em [14] foi implementado. Experimentos envolvendo genomas construídos a partir de dados biológicos e sintéticos mostram que os AGs fornecem soluções que incrementam consideravelmente a precisão dos resultados (número de translocações) quando comparados com os resultados providos pelo algoritmo aproximado de raio $1.5 + \varepsilon$. Testes estatísticos mostram que AM possui melhor desempenho resguardando a precisão dos resultados obtidos pelo AG, e que $OBL\text{-}AG$ não fornece qualquer melhoria na qualidade das soluções quando comparadas com as computadas pelo AG.

Por fim, são fornecidos duas abordagens paralelas para melhorar o AM (submetido em [41]).

- A primeira (denotada como AM_F) busca acelerar a execução do AM mantendo a acurácia dos resultados, fazendo uso do modelo paralelo mestre-escravo, onde o processo mestre mantém os estágios do AM com uma única população de tamanho $n \log n$, enquanto os processos escravos são responsáveis pelo cálculo da função de aptidão dos indivíduos.
- A segunda abordagem paralela, busca incrementar a acurácia dos resultados utilizando múltiplas populações, tal que cada processo mantém uma instância do AM com sua respectiva população. Duas variantes foram propostas:
  - A primeira denotada como AM_{MPT}, compartilha os melhores indivíduos entre as diferentes populações durante o ciclo de reprodução. A estratégia consiste em incrementar a amostra do espaço de busca do problema DTS. Através da troca de melhores indivíduos, espera-se acelerar a convergência de cada população para soluções melhores adaptadas ao problema DTS.
  - A segunda variante, denotada como AM_{MPS}, na qual cada população evolui de forma homogênea sem compartilhar quaisquer indivíduos. A estratégia consiste em unicamente aumentar o tamanho da amostra do espaço de busca do problema DTS, de forma que cada população evolua independentemente uma da outra. Em ambos os casos, cada processo mantém uma população de ta-

manho $n \log n$ e o objetivo é essencialmente melhorar a acurácia dos resultados por explorar uma amostra maior do espaço de busca do problema *DTS*.

Resultados a partir de experimentos mostram que o AM_F reduz significativamente o tempo de execução do AM. Para as variantes com múltiplas populações, AM_{MPS} incrementa a acurácia dos resultados obtidos quando comparado com o AM; todavia, a variante que envolve troca de indivíduos (AM_{MPT}) não provê qualquer incremento na acurácia dos resultados. Através de testes estatísticos é possível identificar que AM_{MPS} fornece resultados tanto diferentes quanto melhores quando comparado com os resultados providos pelos AM e AM_{MPT}, respectivamente.

1.3 Objetivo

O objetivo deste trabalho é propor algoritmos genéticos para resolver o problema de distância de translocação entre genomas sem sinal. Para isso foi necessário desenvolver as seguintes tarefas:

1. Realizar um estudo para compreender porque a distância de translocação entre genomas sem sinal é $\mathcal{NP}$-difícil.
2. Realizar um estudo teórico e implementação do algoritmo de raio de aproximação $1.5+\varepsilon$.
3. Propor e implementar um algoritmo genético básico para o problema de distância de translocação entre genomas sem sinal.
4. Adicionar ao algoritmo genético técnicas de otimização conhecidas na literatura como: memético e aprendizagem baseada em oposição.
5. Propor e implementar versões em paralelo do melhor algoritmo genético, com foco no tempo de execução e na precisão dos resultados.
6. Realizar experimentos utilizando como entrada genomas sintéticos (gerados a partir de um algoritmo) e biológicos para ambos algoritmos genéticos e de raio $1.5+\varepsilon$.
7. Comparar os resultados do algoritmo de raio $1.5+\varepsilon$, com os resultados providos pelos algoritmos genéticos propostos.

1.4 Organização do Trabalho

O trabalho segue a seguinte organização:

Capítulo 2. [Referencial Teórico] Apresenta as definições e terminologias usadas com relação aos genomas e o problema de distância de translocação, bem como os conceitos relacionados com algoritmos genéticos (AG).

Capítulo 3. [Revisão da Literatura] Revisa a literatura para o problema de distância de translocação para genomas com e sem sinal, no último, a compilação da prova que o problema é $\mathcal{NP}$-difícil.

Capítulo 4. [Algoritmo Aproximado de Raio $1.5 + \varepsilon$] Apresenta o algoritmo de raio de aproximação $1.5+\varepsilon$ para o problema de distância de translocação entre genomas sem sinal.

Capítulo 5. [Proposto *AG*] Apresenta o AG proposto para solucionar o problema de distância de translocação entre genomas sem sinal, o qual foi publicado em [42].

Capítulo 6. [Melhorias Incorporadas Ao *AG*] Apresenta duas técnicas de otimização (memético e aprendizagem baseada em oposição) incorporadas ao AG. Tal trabalho foi publicado em [43].

Capítulo 7. [Versão Paralela do *AG*] Apresenta os conceitos sobre paralelismo, e dois métodos propostos para o AG com a técnica memético. Tal trabalho foi submetido em [41].

Capítulo 8. [Experimentos e Resultados] Mostra os experimentos realizados tomando como entrada genomas sintéticos e biológicos, além de apresentar uma discussão sobre os resultados dos experimentos.

Capítulo 9. [Conclusão e Trabalhos Futuros] Apresenta as conclusões e indica sugestões de continuação da pesquisa.

Capítulo 2

Fundamentação Teórica

O capítulo possui a seguinte organização. A Seção 2.1 apresenta definições e terminologias para genomas com e sem sinal, subpermutações e uma estrutura de dados denotada como Ponto-de-Quebra utilizada para se computar a distância evolutiva entre espécies. Na Seção 2.2 são apresentados os conceitos sobre computação evolucionária com ênfase nos algoritmos genéticos descrevendo com detalhes todo o processo envolvido para a modelagem de um algoritmo genético.

2.1 Definições e Terminologia sobre Genomas

As seguintes abstrações serão adequadas para o nosso propósito, tendo como fonte [5] e [23].

2.1.1 Genes, Cromossomos e Genomas

Um genoma é constituído por uma sequência de genes. Em informática, para representar os genes de um genoma, cada gene é associado com um número inteiro. Inteiros com e sem sinal representam genes orientados e não orientados, respectivamente. Um cromossomo é uma sequência de genes e um genoma é formado por um conjunto ordenado de cromossomos. Para simplificar o modelo, considera-se que cada gene aparece um única vez em um dado genoma. Assim, um genoma G com n genes orientados e N cromossomos pode ser visto como um conjunto $\{(x_{11}, \ldots, x_{1r_1}), \ldots, (x_{k1}, \ldots, x_{kr_k}), \ldots (x_{N1}, \ldots, x_{Nr_N})\}$, onde $\sum_{i=1}^{N} r_i = n$, $x_{ij} \in \{\pm 1, \ldots, \pm n\}$ e $|x_{ij}| \neq |x_{lk}|$ sempre que $i \neq l$ ou $j \neq k$. Para a versão sem sinal, $x_{ij} \in \{1, \ldots, n\}$.

Cromossomos não tem orientação. Assim, os cromossomos $X = (x_1, x_2, \ldots, x_k)$ e X' $= (-x_k, -x_{k-1}, \ldots, -x_1)$ são os mesmos no caso com sinal; enquanto que X e $X'' = (x_k, x_{k-1}, \ldots, x_1)$ são iguais no caso sem sinal. Assim, por exemplo $G = \{(+1, -3),$

$(-4, +2, -5)\}$ é um genoma com 5 genes e 2 cromossomos, além disso, G e $G' = \{(+1, -3), (+5, -2, +4)\}$ são os mesmos genomas. Para distinguir genoma com sinal de sem sinal, o primeiro é indicado com uma seta: $\vec{G}$.

Um Genoma contendo um único cromossomo pode ser visto como uma permutação π no grupo simétrico S_n. Uma permutação é uma função bijetiva de naturais para naturais num conjunto finito $\{1, \ldots, n\}$. Uma permutação π pode ser representada como $(\pi_1, \ldots, \pi_n)$, onde π_i abrevia $\pi(i)$, para $1 \leq i \leq n$.

2.1.2 Subpermutações

Sejam A e B dois genomas com os mesmos genes e $S = (x_1, x_2, \ldots, x_n)$ um cromossomo em A. Uma subpermutação no cromossomo S em A para B (de forma abreviada, *SP* em A para B) é um segmento $[x_i, x_{i+1}, \ldots, x_{i+l}]$ ocorrendo em S com pelo menos 3 genes, tal que exatamente os naturais entre $|x_i|$ e $|x_{i+l}|$ ocorrem no conjunto $\{|x_{i+1}|, \ldots, |x_{i+l-1}|\}$ e existe outro segmento $[y_j, y_{j+1}, \ldots, y_{j+l}]$ em algum cromossomo T do genoma B, satisfazendo:

- $|x_i| = |y_j|$ e $|x_{i+l}| = |y_{j+l}|$;
- $\{|x_{i+1}|, \ldots, |x_{i+l-1}|\} = \{|y_{j+1}|, \ldots, |y_{j+l-1}|\}$;
- $[x_i, x_{i+1}, \ldots, x_{i+l}] \neq [y_j, y_{j+1}, \ldots, y_{j+l}]$.

Uma *MinSP* é uma *SP* em A para B que não contém qualquer outra *SP*. Por exemplo, considere os genomas:
$A = \{(1, 3, 2, 4, 5, 8, 6), (7, 9)\}$ e
$B = \{(1, 2, 3, 4, 5, 6), (7, 8, 9)\}$;
$[1, 3, 2, 4, 5]$ é uma *SP* e $[1, 3, 2, 4]$ é uma *MinSP*. Para os genomas com sinal $\vec{A}$ e $\vec{B}$ abaixo, $[+1, -3, +2, +4, +5]$ é uma *SP* e $[+1, -3, +2, +4]$ é uma *MinSP*.
$\vec{A} = \{(+1, -3, +2, +4, +5, +8, +6), (+7, +9)\}$ e
$\vec{B} = \{(+1, +2, +3, +4, +5, +6), (+7, +8, +9)\}$.

2.1.3 Grafo de Pontos-de-Quebra

O grafo de Pontos-de-Quebra é uma importante estrutura de dados usada em combinatória de permutações e também útil para mensurar a distância evolutiva entre as espécies a partir de seus genomas utilizando a métrica translocação, entre outras.

Dado um cromossomo $X = (x_1, x_2, \ldots, x_n)$ de um (com ou sem sinal) genoma, os genes x_i e x_{i+1}, para $1 \leq i \leq n - 1$, são adjacentes; caso contrário, eles são não adjacentes. Além disso, genes em diferentes cromossomos são não adjacentes.

Considere dois genomas com sinal $\vec{A}$ e $\vec{B}$ com os mesmos genes e a mesma quantidade de cromossomos. O grafo de Pontos-de-Quebra $G_s(\vec{A}, \vec{B})$ é construído como segue. Os

vértices do grafo são construídos a partir dos genes em $\vec{A}$, tal que, para cada cromossomo $X = (x_1, x_2, \ldots, x_n)$ de $\vec{A}$, substitui-se cada x_i por um par ordenado de vértices $(L(x_i), R(x_i))$ com $L(x_i)$ representando a esquerda de x_i, e $R(x_i)$ representando a direita de x_i. Se x_i é positivo, então $(L(x_i), R(x_i)) = (x_i^t, x_i^h)$, ou se x_i é negativo tem-se $(L(x_i), R(x_i)) = (x_i^h, x_i^t)$. Se os genes x_i e x_{i+1} são adjacentes em $\vec{A}$, adiciona-se uma aresta preta $(R(x_i), L(x_{i+1}))$ em $G_s(\vec{A}, \vec{B})$. Se os genes x_i e x_{i+1} são adjacentes em $\vec{B}$, adiciona-se uma aresta cinza $(R(x_i), L(x_{i+1}))$ em $G_s(\vec{A}, \vec{B})$. A Figura 2.1 ilustra o grafo $G_s(\vec{A}, \vec{B})$ para $\vec{A}$=$\{(+1, +3, +7), (+5, -2, +6, +4)\}$ e $\vec{B}$=$\{(+1, +2, +3, +4), (+5, +6, +7)\}$

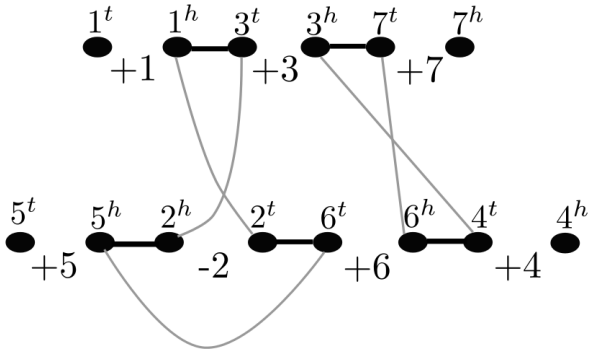

Figura 2.1: Grafo $G_s(\vec{A}, \vec{B})$ para genomas com sinal.

Note que há unicamente uma aresta preta e uma cinza incidentes para cada vértice em $G_s(\vec{A}, \vec{B})$, exceto para os vértices formados a partir dos genes finais dos cromossomos. Portanto, um grafo de Pontos-de-Quebra para genomas com sinal tem apenas uma forma de ser decomposto em ciclos de cor alternada, onde arestas pretas e cinzas aparecem alternadas. (veja em [12]). Um ciclo é chamado longo, se contém pelo menos duas arestas pretas (ou cinzas), caso contrário é curto.

Os grafos de Pontos-de-Quebra também são definidos para genomas sem sinal. Considere dois genomas sem sinal A e B com os mesmos genes e número de cromossomos. O grafo $G_u(A, B)$ para A e B é construído como segue: cada vértice em $G_u(A, B)$ representa um único gene em A. Para todo cromossomo $X = (x_1, x_2, \ldots, x_n)$ em A, existe uma aresta preta entre x_i e x_{i+1}, $1 \leq i < n$; e, para todo cromossomo $Y = (y_1, y_2, \ldots, y_m)$ de B existe uma aresta cinza entre vértices rotulados com y_j e y_{j+1}, sempre que y_j e y_{j+1} são adjacentes em B.

A Figura 2.2 ilustra o grafo $G_u(A, B)$ para $A = \{(1, 3, 7), (5, 2, 6, 4)\}$ e $B = \{(1, 2, 3, 4), (5, 6, 7)\}$.

O grafo $G_u(A, B)$ pode ser particionado em um conjunto de ciclos de cor alternada. Note que cada vértice em $G_u(A, B)$ tem o mesmo número de arestas pretas e cinzas incidentes; vértices associados com genes nos extremos dos cromossomos em A tem uni-

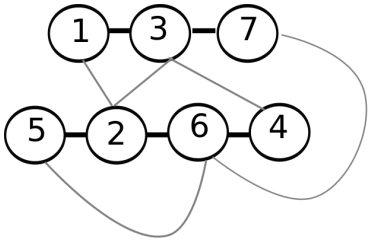

Figura 2.2: $G_u(A, B)$ para genomas sem sinal.

camente uma aresta preta e uma cinza, e vértices associados a genes internos nos cromossomos em A tem exatamente duas arestas pretas e duas cinzas. Assim, há mais de uma forma de particionar $G_u(A, B)$ em ciclos de cor alternada.

Grafos de Pontos-de-Quebra também serão definidos para permutações no Capítulo 3, onde se pode ver que tais grafos são semelhantes aos grafos obtidos para genomas A e B, onde A e B têm apenas um cromossomo.

2.1.4 Translocação

Uma translocação é dita ser "atuante" em dois cromossomos X e Y quando ambos são cortados e representados como $X = (X_1, X_2)$ e $Y = (Y_1, Y_2)$ e os segmentos produzidos em ambos os cromossomos são intercambiados, transformando X e Y em dois novos cromossomos X' e Y'. É importante mencionar que uma operação de translocação trabalha com a hipótese que os quatros segmentos são não vazios.

No cenário de translocação, dois tipos de operações sobre segmentos de cromossomos são apresentadas: *Prefixo-Prefixo* e *Prefixo-Sufixo*. Dados dois cromossomos $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_m)$ em um genoma com sinal, aplicando uma operação de translocação *Prefixo-Prefixo* $\rho(X, Y, x_i, y_k)$, obtém-se dois novos cromossomos $X' = (x_1, \ldots, x_i, y_{k+1}, \ldots, y_m)$ e $Y' = (y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$. Por outro lado, aplicando a operação *Prefixo-Sufixo* $\theta(X, Y, x_i, y_k)$ obtém-se os novos cromossomos $X' = (x_1, \ldots, x_i, -y_k, \ldots, -y_1)$ e $Y' = (-y_m, \ldots, -y_{k+1}, x_{i+1}, \ldots, x_n)$ (veja Figura 2.3).

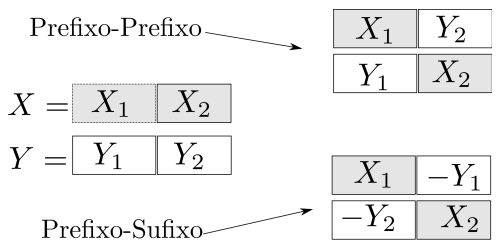

Figura 2.3: Tipos de operações de translocação *Prefixo-Prefixo* e *Prefixo-Sufixo*.

Exemplo: Considere o genoma $\vec{A} = \{X, Y, Z\}$ com $X = (+1, +2, -7, +5), Y = (+4, +3)$ e $Z = (+6, -8, +9)$. A translocação $\rho(X, Y, +2, +4)$ transforma $\vec{A}$ no genoma

$$\vec{A'} = \{(+1, +2, +3), (+4, -7, +5), Z\}.$$

Aplicando a translocação $\theta(X, Z, +2, +6)$ em $\vec{A}$, obtém-se o genoma

$$\vec{A''} = \{(+1, +2, -6), Y, (-9, +8, -7, +5)\}.$$

Para o caso sem sinal, translocação é definida como segue. Considere dois cromossomos $X = (x_1, x_2, \ldots, x_n)$ e $Y = (y_1, y_2, \ldots, y_m)$. Uma translocação por *Prefixo-Prefixo* $\rho(X, Y, x_i, y_k)$ transforma X e Y em dois novos cromossomos $X' = (x_1, \ldots, x_i, y_{k+1}, \ldots, y_m)$ e $Y' = (y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$; por outro lado uma translocação por *Prefixo-Sufixo* $\theta(X, Y, x_i, y_k)$ transforma X e Y em $X' = (x_1, \ldots, x_i, y_k, \ldots, y_1)$ e $Y' = (y_m, \ldots, y_{k+1}, x_{i+1}, \ldots, x_n)$.

Exemplo: Considere o genoma sem sinal $A = \{X, Y, Z\}$, com cromossomos $X = (1, 2, 7, 5), Y = (4, 3)$ e $Z = (6, 8, 9)$. $\rho = (X, Y, 2, 4)$ transforma A em

$$A' = \{(1, 2, 3), (4, 7, 5), Z\}.$$

$\theta(X, Z, 2, 6)$ transforma A em

$$A'' = \{(1, 2, 6), Y, (9, 8, 7, 5)\}.$$

A seguir, serão definidos os problemas de distância de translocação para genomas com e sem sinal.

Problema de distância de translocação com sinal (DTC): considere dois genomas com sinal $\vec{A}$ e $\vec{B}$ com os mesmos genes e o mesmo número de cromossomos, onde os genes de B são inteiros positivos em ordem incremental. O DTC consiste em encontrar o mínimo número de translocações necessárias para transformar $\vec{A}$ em $\vec{B}$.

Problema de distância de translocação sem sinal (DTS): o DTS é definido como o DTC, porém restrito para genomas sem sinal.

Considere um cromossomo $X = (x_1, x_2, ..., x_k)$, os genes $+x_1$ e $-x_k$ são chamados *caudas* de X para genomas com sinal; considerando genomas sem sinal, as *caudas* de X são x_1 e x_k. Dois genomas são chamados *co-caudais* se os seus conjuntos de *caudas* são os mesmos. Os genomas $\vec{B}$ e $\vec{C}$ abaixo são *co-caudais* uma vez que eles compartilham os

mesmos conjuntos de *caudas*, que é $\{+1, -6, +7, -10\}$.

$$\vec{B} = \{(+1, +2, -4, +3, +5, +6), (+7, -9, +8, +10)\},$$

$$\vec{C} = \{(+1, +2, +3, +4, +5, +6), (+7, +8, +9, +10)\}.$$

Note que, os genomas $\vec{A}$, $\vec{A'}$ e $\vec{A''}$ assim como A, A' e A'' de exemplos prévios são *co-caudais*.

Translocações por *Prefixo-Prefixo* ou *Prefixo-Sufixo* não modificam o conjunto de caudas de um genoma. Assim, a fim de transformar um genoma A em B (considerando genomas com e sem sinal) por translocações, A e B satisfazem a propriedade abaixo.

Propriedade 1 Os genomas A e B tem os mesmos genes, o mesmo número de cromossomos e A e B são *co-caudais*. Note que, quando A e B são *co-caudais*, os genes de A e B podem ser renomeados de maneira que B seja reescrito como uma identidade (sequência de genes ordenados em ordem incremental). Assim, sem perca de generalidade, sempre que A e B são *co-caudais* pode-se assumir que B é uma identidade.

No restante deste trabalho n sempre representará o tamanho (quantidade de genes) de um dado genoma em questão.

2.2 Computação Evolucionária

A proposta de que evolução natural poderia ser utilizada como um mecanismo para a análise de sistemas mais complexos do que aqueles matematicamente analisáveis veio com a invenção do computador. Nas décadas de 70 e 80 a principal ideia foi desenvolvida em diferentes implementações algorítmicas sob nomes tais como programação evolucionária [19], estratégia evolucionária (proposta por Rechenberg et al. e detalhada em [32]) e algoritmos genéticos [26]. Estes ramos se fundiram na década de 90, e nos últimos 20 anos, computação evolucionária tem provado ser muito bem sucedida em uma ampla gama de problemas computacionais envolvendo otimização e modelagem.

Computação evolucionária obteve um sucesso estrondoso no campo computacional de forma que muitas das suas técnicas hoje em dia são utilizadas por uma ampla variedade de algoritmos de otimização, tais como os algoritmos baseados em inteligência de enxame (*Swarm intelligence*), onde dentre estes algoritmos, temos: otimização por enxame de partículas, otimização por colônia de formigas, colônia artificial de abelhas etc.

2.2.1 Algoritmos Genéticos

Os conceitos e terminologias apresentados nesta seção foram em sua maioria tomados do livro *"Introduction to Genetic Algorithms"* [44].

Definindo Algoritmos Genéticos

Charles Darwin é considerado o pai da teoria da evolução natural na origem das espécies, onde tais espécies se mantém em um processo evolutivo baseado no princípio de seleção natural "sobrevivência dos mais aptos". Partindo da ideia de evolução encontrada na natureza, então deve ser interessante simular a evolução natural e desenvolver um método que resolva problemas de busca e otimização.

Em 1975, Holland fazendo uso da ideia de evolução na natureza, aplicou tais definições afim de simular a evolução natural para desenvolver métodos que resolviam problemas de otimização em seu livro "*Adaptation in natural and artificial systems*". Ele descreveu como aplicar os princípios da evolução natural para problemas de otimização, além de fornecer os primeiros algoritmos genéticos. A teoria de Holland foi amplamente estudada e desenvolvida de uma tal forma que hoje, algoritmos genéticos são uma ferramenta poderosa de resolução de problemas voltados a busca e otimização.
Os algoritmos genéticos estão baseados nos princípios da genética e evolução, como segue: a informação genética de uma determinada população contém potencialmente a solução, ou converge para uma solução melhor, para um determinado problema. Esta solução, possivelmente está contida em um determinado indivíduo. Assim, através de técnicas de combinações genéticas entre vários indivíduos, pode-se obter indivíduos cada vez mais próximos do ideal, que após um certo tempo converge a uma boa solução.

Terminologia Básica

É apresentada a terminologia básica relacionada aos algoritmos genéticos. A fim de modelar uma solução baseada em evolução natural, algoritmos genéticos emulam o processo evolucionário que é realizado na natureza. Os seguintes conceitos são necessários a fim de entender os algoritmos genéticos.

Indivíduos Um indivíduo é um portador de código genético representando uma única solução para um dado problema. Além disso, cada indivíduo é representado como um cromossomo, que por sua vez está subdividido em genes. Através deste cromossomo é possível tanto adquirir uma solução como também explorar seu código genético para gerar novos indivíduos que melhor se adaptem ao problema em questão.

Genes Genes são os componentes básicos dos cromossomos. Tais cromossomos são formados por uma sequência de genes de comprimento arbitrário. Os genes trazem consigo código genético que pode descrever uma possível solução para um problema sem ser realmente a solução.

Função de Aptidão A função de aptidão é utilizada para mensurar quão bom é um individuo, levando em consideração algum método de avaliação (de acordo com o problema envolvido) para posterior uso pelos operadores de reprodução. A ideia desta função, consiste em tomar como entrada um indivíduo e através de um conjunto de métricas a serem alcançadas em tal indivíduo, avaliar e fornecer como saída um valor real. Deve-se mencionar que identificar um modo para avaliar o quão distante um indivíduo está da solução ideal nem sempre é uma tarefa trivial.

População A população é um conjunto de indivíduos representando o espaço de busca de possíveis soluções que estará sendo manipulada pelo algoritmo genético. Devem-se salientar dois aspectos importantes relacionado a população: **1)** A geração da população inicial e **2)** o tamanho da população. A geração da população inicial por padrão (muitas vezes) é criada de forma aleatória. Tal método é preferível para problemas sobre os quais nenhum conhecimento é diagnosticado ou para avaliar o desempenho de um algoritmo. Todavia há casos específicos em que é utilizado um conhecimento a priori sobre o dado problema. Utilizando tal conhecimento, um conjunto de requerimentos é obtido e soluções as quais satisfazem estes requerimentos são coletadas para formar uma população inicial. O tamanho da população depende em muito da complexidade do problema, uma vez que muitos dos problemas tratados com algoritmos genéticos são de caráter $\mathcal{NP}$-difícil ou $\mathcal{NP}$-completo. Logo, explorar todo o espaço de busca de soluções se torna inviável em tempo polinomial.

Reprodução

No ciclo de reprodução, indivíduos são selecionados da população para se reproduzirem, gerando descendentes que irão compor a próxima geração. Os pais são selecionados da população usando um esquema que favorece na maioria das vezes os indivíduos mais aptos. O ciclo de reprodução é dividido em três etapas: seleção de indivíduos, cruzamento dos indivíduos selecionados para criar os descendentes, mutação aplicada sobre os descendentes e substituições de indivíduos velhos na população pelos descendentes.

- **Seleção**
  Consiste no processo de escolha dos indivíduos (pais) da população corrente para a geração de descendentes. Tais indivíduos são selecionados a partir dos melhores indivíduos na população encontrados pelos seus valores de aptidão; visto que, segundo a teoria da evolução de Darwin o melhor sobrevive para criar uma nova descendência.

- **Cruzamento**

  Tal operador toma dois indivíduos, particiona suas cadeias de cromossomos em alguma posição escolhida aleatoriamente, para produzir dois segmentos cabeça e dois segmentos cauda. Os segmentos da cauda são então trocados para produzir dois novos indivíduos completos (Figura 2.4). Isto é conhecido como cruzamento de único ponto. Tal método é o mais utilizado em algoritmos genéticos, uma vez que é simples e efetivo.

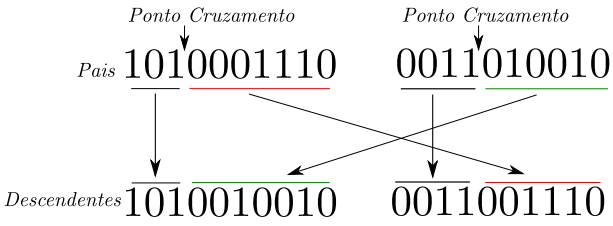

Figura 2.4: Exemplo de recombinação de ponto único.

- **Mutação**

  Mutação consiste em alterações que acontecem na sequência dos nucleotídeos do material genético de um determinado organismo. As mutações conhecidas na literatura como benéficas, podem trazer mudanças evolutivas adaptativas, tendo como consequência a evolução da espécie em questão [20]. Assim, Depois da fase de cruzamento, os descendentes são submetidos a mutação. A mutação desempenha o papel de recuperar o material genético perdido, e também para perturbar aleatoriamente a informação genética. Se o operador de cruzamento explora a solução atual no intuito de encontrar melhores soluções, além de ajudar na exploração do espaço de busca, a mutação é vista como um operador para manter a diversidade genética na população.

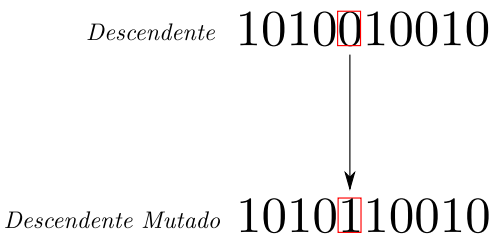

Figura 2.5: Exemplo de mutação.

- **Substituição**

  A substituição é o último passo do ciclo de reprodução. Após a escolha dos pais

de uma população e a partir deles, gerar novos indivíduos, tem-se a necessidade de atualizar a população. Porém nem todos podem voltar. Logo, deve-se escolher que membros da população atual devem ser substituídos pelas novas soluções (normalmente os piores indivíduos na população são substituídos).

Parâmetros Genéticos

O desempenho de um algoritmo genético está intimamente relacionado com a forma de definição do espaço de busca (população) bem como de cada parâmetro a ser utilizado nos operadores de ciclo de reprodução. Assim, há uma necessidade de analisar cuidadosamente estes parâmetros para poder estabelecê-los conforme a necessidade de cada problema a ser resolvido [10]. Na literatura comumente estes parâmetros são definidos através de exaustivos testes, onde se executa o algoritmo genético fazendo uso de possíveis valores obtidos de intervalos discretos para cada parâmetro, dessa forma é possível definir os melhores valores para tais parâmetros, os quais provém as melhores soluções. Os principais parâmetros utilizados e que devem ser ajustados previamente à computação dos resultados, são listados na sequência. Além disso são sugeridos intervalos de bons valores para estes parâmetros tomando como base trabalhos encontrados na literatura que utilizam algoritmos genéticos:

- **Tamanho da População**
  O tamanho da população tem impacto no desempenho dos algoritmos genéticos. Ao escolher uma população de tamanho reduzido, o desempenho do algoritmo pode ser privilegiado (caso o número de gerações seja estático), contudo a eficácia do algoritmo pode estar sendo comprometida, uma vez que a população está refletindo uma pequena amostra do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura bem mais ampla do domínio do problema, além de evitar um problema muito comum em algoritmos evolutivos, denotado como o "falso global", que nada mais é que a convergência prematura para uma solução local ao invés de uma global. No entanto, ao se escolher uma amostra muito grande do domínio do problema, tem-se a necessidade de uma fatia muito maior de recursos computacionais, ou que o algoritmo execute por um período de tempo maior ou mesmo desconhecido. Logo, um balanceamento entre desempenho versus eficácia é visto com bons olhos na busca por um algoritmo eficaz no sentido de prover soluções com alto grau de acurácia dentro de um tempo de execução razoável.

- **Porcentagem de Cruzamento**
  Representa uma porcentagem estimada para aplicar ou não o operador de cruzamento entre dois pais. Ao utilizar valores altos, haverá um grande número de

descendentes introduzidos rapidamente na população. Todavia, podem-se perder indivíduos contendo bons genes, antes que seja possível utilizá-los. Contudo, ao optar por um porcentagem muito baixa, o algoritmo pode se tornar lento ou mesmo chegar a estagnar. Frequentemente um valor bom para esta taxa varia entre 50% até 70%. Isso varia muito de problema para problema.

- **Porcentagem de Mutação**
  Representa uma porcentagem estimada para aplicar ou não o operador de mutação em cada gene em um indivíduo descendente. Uma porcentagem baixa de mutação previne que a busca permaneça estagnada em regiões do espaço de busca. Normalmente, esse valor varia entre 1% a 5%, escolher uma porcentagem mais alta, pode acabar mudando drasticamente os genes herdados dos seus respectivos pais, o que normalmente não é o desejado.

- **Porcentagem de Substituição**
  Representa uma porcentagem estimada para selecionar parte da população que será o espaço para inserção dos descendentes que possuam melhor valor de aptidão. Com um valor médio (de 40% a 70% da população), a tendência é que a população tenha uma convergência mais rápida sem o risco de uma convergência prematura, utilizando um valor baixo, o algoritmo genético pode tornar-se muito lento.

É importante enfatizar que um algoritmo genético pode envolver mais parâmetros que estes apresentados acima, dependendo das técnicas que podem estarem sendo utilizadas para acelerar o processo de convergência para uma solução viável em tempo polinomial.

Algoritmo Genético Simples (SGA)

Em [44] faz-se uma classificação dos algoritmos genéticos como AG Paralelo, AG *Messy*, AG Distribuído. Para o problema de distância de translocação, foi tomado como modelo o algoritmo genético simples (*SGA*) que é descrito como segue:

O procedimento padrão de execução do *SGA* contém 6 passos, que são:

1. Definir uma população inicial gerada aleatoriamente.
2. Selecionar indivíduos (pais) para o ciclo de reprodução (usando o valor de aptidão).
3. Aplicar cruzamento sobre um par de pais.
4. Aplicar a mutação sobre a descendência.

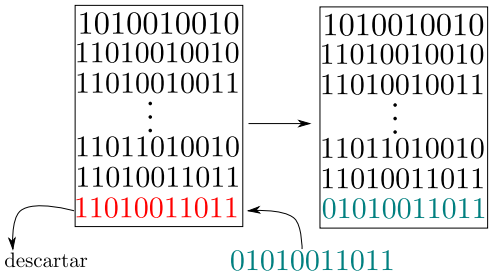

Figura 2.6: Exemplo de inserção de um descendente melhor adaptado na população.

5. Inserir os novos indivíduos (descendentes) na população (veja Figura 2.6, o qual considera a inserção de um descendente na população).

6. Se um indivíduo (representando uma solução) alcançou a meta preestabelecida, ou o número de gerações se esgotou, retornar o melhor indivíduo encontrado. Caso contrário voltar ao item (2).

A partir da simulação do processo evolutivo, tem-se indivíduos cada vez mais bem adaptados. Isto é, com melhor valor de aptidão, de maneira que no final obtém-se soluções com alto grau de adequação ao problema proposto.

Capítulo 3

Contextualização-O problema de Distância de Translocação

O capítulo possui a seguinte organização. A Seção 3.1 apresenta um resumo do estado da arte para o problema de *DTC* com os trabalhos mais relevantes na literatura. Na sequência, a Seção 3.2 apresenta a prova de que o problema de *DTS* é $\mathcal{NP}$-difícil. Posteriormente, na Seção 4.4 são apresentados os trabalhos encontrados na literatura para o problema de *DTS*.

3.1 Distância de Translocação com Sinal

O problema de distância de translocação tem sido estudado extensivamente. Kececioglu e Ravi foram os primeiros a estudar o problema na versão com sinal e em [30] forneceram o primeiro algoritmo aproximado com um raio de aproximação 1.5.

No ano de 1996 Hannenhalli [23] forneceu o primeiro algoritmo polinomial ($O(n^3)$) exato para o cálculo da distância de translocação, neste mesmo artigo ele propôs um segundo algoritmo, que computa as translocações necessárias para o rearranjo dos genomas. A ideia por trás destes algoritmos consiste em: a partir da estrutura de grafo de Pontos-de-Quebra encontrar uma máxima decomposição em ciclos de cor alternada e realizar tanto o cálculo da distância quanto explorar tais ciclos para encontrar a sequência de translocações.

Após tal feito vários outros algoritmos foram propostos, tais como um de complexidade quadrática proposto por Wang et al. em [48] (utiliza a máxima decomposição em ciclos de cor alternada do grafo de Pontos-de-Quebra) para obter tanto a distância quanto a sequência de translocações, e os propostos por Bergeron et al. em [5] que fazem uso de um sofisticado método, onde um genoma é visto como uma única permutação (permutação é uma forma simples de representar um genoma contendo um único cromossomo) de

forma que uma estrutura contendo marcadores em forma de pontos brancos representam os genes finais de cada cromossomo, e pontos pretos representam os genes internos nos cromossomos. A partir desta estrutura é computada a decomposição em ciclos de cor alternada. Por fim, eles introduziram a noção de intervalos elementares (apresentada no Capítulo 2 como *MinSP*), e destes intervalos elementares construir uma floresta de árvores (as árvores representam os intervalos elementares) o qual é parte chave para o cálculo da distância e a sequência de translocações. Além disso, é importante enfatizar que Bergeron et al. em [5] encontraram um problema no algoritmo proposto por Hannenhalli [23] que provia as translocações necessárias para transformar um genoma em outro, e o corrigiram (a complexidade se manteve ($O(n^3)$) e introduziram um novo algoritmo para o cálculo da distância em tempo linear.

Ozery-Flato e Shamir em [37] melhoraram a complexidade do algoritmo que fornecia as translocações proposto por Bergeron et al. para $O(n^{\frac{3}{2}}\sqrt{\log n})$, tal algoritmo faz uso da estrutura de dados chamada de floresta de sobreposição proposta em [3] inicialmente para ordenação por reversão e adaptada para translocações, e utilizam reversões para imitar translocações (*Prefixo-Prefixo* e *Sufixo-Sufixo*) fato este que tinha sido levantado e provado por Hannenhalli e Pevzner em [24].

3.2 Complexidade do Problema de Distância de Translocação sem Sinal

Para o problema com genomas sem sinal, os que são tratados nesse trabalho, o problema foi demonstrado ser $\mathcal{NP}$-difícil por D. Zhu e L. Wang em [50]. Antes de apresentar uma breve descrição dos trabalhos encontrados na literatura para distância de translocação envolvendo genomas sem sinal, será apresentada uma compilação contendo os passos necessários da prova que não foram expostos por D. Zhu e L. Wang em [50].

3.2.1 Roteiro da Prova

Para um bom entendimento dos estágios da prova, um roteiro é apresentado contendo todos os passos necessários que foram implicitamente ou explicitamente usados em [50]. Primeiramente, alguns problemas devem ser definidos.

k-cliques: checa-se o conjunto de arestas de um grafo H pode ser particionado em cliques de tamanho k. Para $k \geq 3$, k-cliques é um problema $\mathcal{NP}$-completo.

MAX-ECD: considere um grafo euleriano H. O problema consiste em encontrar uma máxima decomposição em ciclos de H, ou seja, um particionamento do conjunto de arestas de H no máximo número de ciclos.

MAX-ACD: dado um grafo de Pontos-de-Quebra $G(\pi)$ de uma permutação π, o problema consiste em encontrar uma máxima decomposição em ciclos alternados de $G(\pi)$.

A prova é organizada nos seguintes passos (Veja Figura 3.1, onde $G_u(X, Y)$ e $G_u(A, B)$ representam grafos de Pontos-de-Quebra para genomas com um único cromossomo e mais de um cromossomo, respectivamente):

- Inicialmente Seção 3.2.2 demonstra que o problema de particionar grafos em ciclos para $k = 3$ o qual é $\mathcal{NP}$-completo é uma instância do problema *MAX-ECD* [27]; Assim, *MAX-ECD* é um problema $\mathcal{NP}$-completo.
- Depois Seção 3.2.3 apresenta uma redução polinomial de *MAX-ECD* em *MAX-ACD* [12]; consequentemente, *MAX-ACD* é um problema $\mathcal{NP}$-difícil;
- Finalmente Seção 3.2.4 reduz polinomialmente *MAX-ACD* no problema *DTS* [50]; assim, o problema *DTS* é $\mathcal{NP}$-difícil.

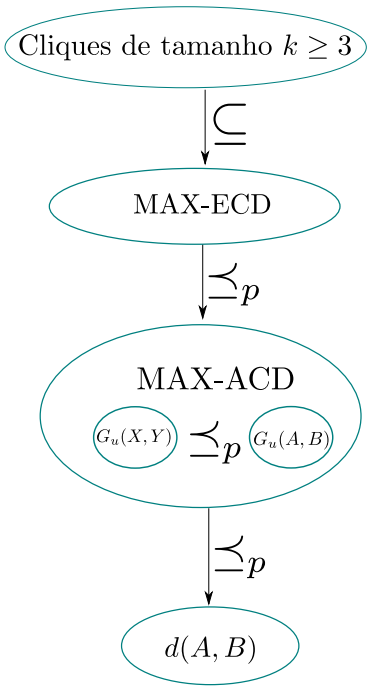

Figura 3.1: Reduções de que*DTS* é $\mathcal{NP}$-difícil.

3.2.2 k-cliques $\subseteq$ MAX-ECD

Na década de 80 Holyer provou em [27] que verificar se o conjunto de arestas de um dado grafo H pode ser particionado em cliques de tamanho k é $\mathcal{NP}$-completo para $k \geq 3$. Em particular, para $k = 3$ se quer verificar se o conjunto de arestas do grafo H pode ser

particionado em triângulos. Neste caso, sem perca de generalidade H pode ser assumido Euleriano. Assim, o problema de decidir se o conjunto de arestas de um grafo Euleriano é particionado em triângulos está em $\mathcal{NP}$-completo. Além disso, uma decomposição de um grafo Euleriano em 3-cliques provê uma máxima decomposição Euleriana, dessa forma pode-se concluir que 3-cliques $\subseteq$ *MAX-ECD*.

3.2.3 MAX-ECD $\preceq_p$ MAX-ACD

Antes de apresentar os detalhes da redução polinomial a partir de *MAX-ECD* para *MAX-ACD* proposta em [12], algumas definições e propriedades devem ser apresentadas.

Grafo de Pontos-de-Quebra $G(\pi)$

Considere uma permutação $\pi = (\pi_1, \ldots, \pi_n)$ em S_n representando um genoma constituído por unicamente um cromossomo.

O grafo de Pontos-de-Quebra $G(\pi) = \langle V, E = R \cup B \rangle$ de π é construído como segue: inicialmente, adiciona-se a π os elementos $\pi_0 = 0$ e $\pi_{n+1} = n + 1$, e considere $\pi = (0, 1, \ldots, n + 1)$. Cada vértice $v \in V$ representa um elemento de π. O grafo $G(\pi)$ é um grafo bi-colorido, onde o conjunto de arestas é particionado em dois subconjuntos: arestas vermelhas (R) e azuis (B). Há uma aresta vermelha (π_i, π_{i+1}) sempre que $|\pi_i - \pi_{i+1}| \neq 1$, para $0 \leq i \leq n$, ou seja, existe uma aresta vermelha entre vértices consecutivos π_i e π_{i+1} que não contém rótulos consecutivos. Neste caso, o par π_i, π_{i+1} é chamado um ponto de quebra de $G(\pi)$. Há uma aresta azul entre vértices rotulados com i e $i + 1$, $1 \leq i \leq n$, sempre que i e $i + 1$ não são posições consecutivas em π. Figura 3.2 ilustra esta noção.

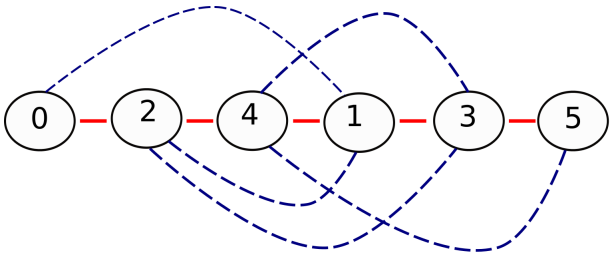

Figura 3.2: Grafo de Pontos-de-Quebra $G(\pi)$ associado a um genoma $\pi = (0, 2, 4, 1, 3, 5)$

Teorema 3.1 (Propriedades de $G(\pi)$ - Teorema 4 em [12])**.** *Um grafo bi-colorido $G = \langle V, R \cup B \rangle$ é o grafo de Pontos-de-Quebra de algum genoma π sse*

- *Cada componente conectada dos subgrafos G(R) e G(B), induzidos por arestas vermelhas e azuis respectivamente, representa um caminho simples;*
- *Cada vértice $i \in V$ tem o mesmo grau (0, 1 ou 2) em G(R) e G(B);*

- *Para cada par de vértices $i, j \in V$ conectados por uma aresta em G(R), i e j não são conectados por uma aresta em G(B), e vice-versa, ou seja, se $(i, j) \in R$ então $(i, j) \notin B$ e vice-versa.*

A Suficiência segue das definições de grafos de Pontos-de-Quebra. A Necessidade requer a construção de uma permutação π através do caminho Hamiltoniano [12].

Um ciclo alternado de $G(\pi)$, é uma sequência de arestas $(r_1, b_1, r_2, b_2, \ldots,$ $r_m, b_m)$, onde $r_i \in R$, $b_i \in B$ para $i = 1, \ldots, m$; r_i e b_j tem um vértice em comum para $i = j = 1, \ldots, m$ e para $i = j + 1$, $j = 1, \ldots, m$ (onde $r_{m+1} = r_1$); e $r_i \neq r_j$, $b_i \neq b_j$ para $1 \leq i < j \leq m$.

Uma decomposição em ciclos alternados de $G(\pi)$ é uma coleção de ciclos contendo alternadas arestas disjuntas, tal que cada aresta de $G(\pi)$ está contida exatamente em um ciclo da coleção. Assim, *MAX-ACD* é um problema de otimização que consiste na busca de uma máxima decomposição em ciclos alternados de $G(\pi)$. Por exemplo veja a *MAX-ACD* na Figura 3.3.

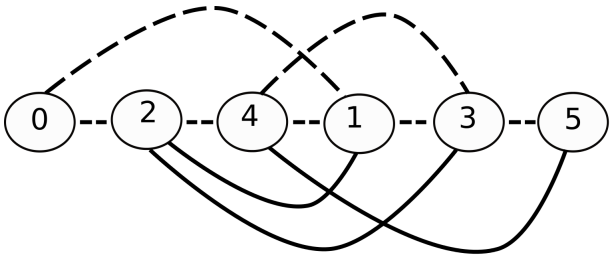

Figura 3.3: *MAX-ACD* de tamanho 2 para $G(\pi)$ para $\pi = (0, 2, 4, 1, 3, 5)$ (Fig. 3.2)

MAX-ACD é $\mathcal{NP}$-difícil

A prova é baseada em uma redução polinomial a partir de *MAX-ECD* para *MAX-ACD*. Dado um grafo Euleriano $H = \langle W, E \rangle$ pode-se construir um grafo bi-colorido G em tempo polinomial, tal que existe uma correspondência um-para-um entre ciclos de H e ciclos alternados de G. O grafo G é construído substituindo cada vértice v de H de grau d, por um subgrafo bi-colorido $G(v)$, contendo d nodos *bottom*. As ligações entre os vértices originais de H se mantém representadas por arestas vermelhas entre os subgrafos bicoloridos $G(v) \in G$ (veja Figura 3.4). Para completar a prova, o grafo G deve satisfazer o Teorema 3.1. A estrutura interna de cada subgrafo $G(v) \in G$ é definida como segue. Seja d e m representando a quantidade de nodos e a quantidade de níveis em $G(v)$, por simplificação escreve-se $G(d, m)$. Seja $s = \frac{d}{2}$ e $\top r$ (representando o teto de r) para $r = \frac{d}{4}$. $G(d, m)$ é subdividido em m níveis, que são todos iguais com exceção do último nível.

Cada nível l, $l = 1, \cdots, m$ contém $2s + 1$ nodos; $s + 1$ deles são nodos de nível superior, denotados como $q_1^l, \cdots, q_{s+1}^l$, e os outros s são os nodos de nível inferior, denotados

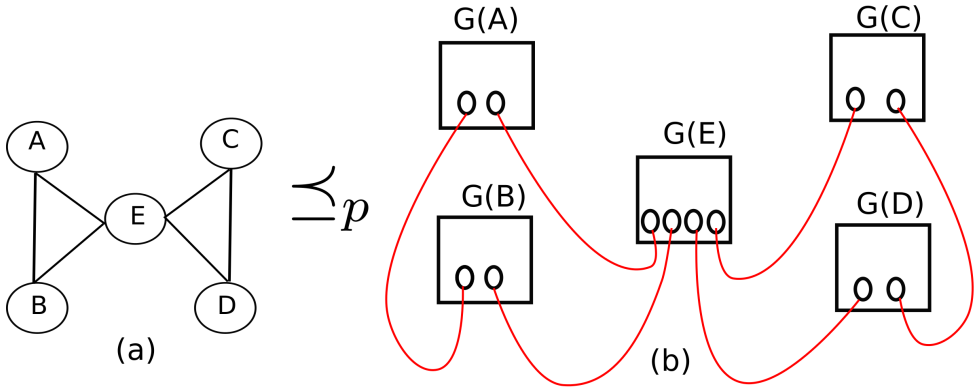

Figura 3.4: (a) O grafo Euleriano H. (b) O grafo bi-colorido $G(\pi)$ construído a partir de H.

por $p_1^l, \cdots, p_s^l$. Nodos $q_1^l, \cdots, q_{s+1}^l$ são conectados aos $p_1^l, \cdots, p_s^l$ por d arestas vermelhas $(q_i^l, p_i), (q_{i+1}^l, p_i)$, para $i = 1, \cdots, s$. Além disso, para $l = 1, \cdots, m-1$ os nodos $q_1^l, \cdots, q_{s+1}^l$ são conectados aos nodos inferiores do nível $l+1$ por d arestas azuis $(p_i^{l+1}, q_i^l), (p_i^{l+1}, q_{i+1}^l)$, para $i = 1, \cdots, s$. Finalmente, os nodos superiores do último nível m são conectados de acordo com suas adjacências uns com os outros por s arestas azuis (q_i^m, q_{i+1}^m), para $i = 1, \cdots, s$. $G(d, m)$ também contém d nodos *bottom*, representados por $b_1, \cdots, b_d$, conectados aos nodos inferiores do primeiro nível por d arestas azuis $(b_{2i-1}, p_i^1), (b_{2i}, p_i^1)$, para $i = 1, \cdots, s$ (veja Figura 3.5).

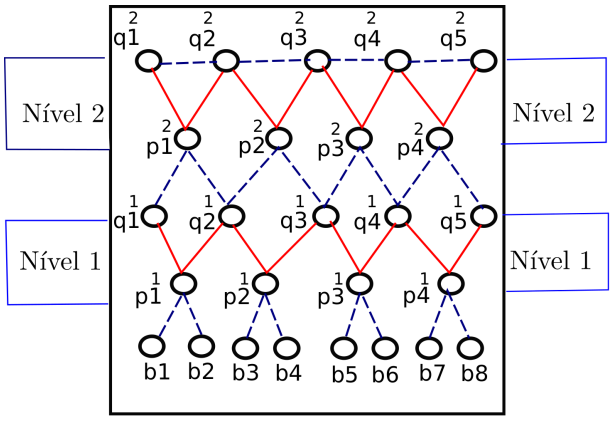

Figura 3.5: Subgrafo $G(d, m)$ for $d = 8$ e $m = 2$.

Observe que todos os nodos de $G(d, m)$ exceto os nodos *bottom* tem o mesmo número de arestas incidentes vermelhas e azuis. Dados dois nodos de $G(d, m)$ um caminho alternado entre tais nodos é um caminho onde as cores das arestas são alternadas. Tomando dois diferentes nodos *bottom* b_i e b_j, é possível construir um caminho alternado entre b_i e b_j, sempre que há um número suficiente de níveis em $G(d, m)$. Pelo Teorema 5 em [12], o conjunto de arestas de $G(d, m)$ pode ser decomposto em s caminhos alternados, conectando qualquer seleção de diferentes pares de nodos *bottom*, sse $m \geq r(s-1)+1$.

Assim, pode-se garantir que é sempre possível de um nodo *bottom* b_i alcançar qualquer outro nodo *bottom* b_j por um caminho alternado e então conectar b_j a outro subgrafo pertencente a G por uma aresta vermelha (veja Figura 3.4). Logo, se existe um ciclo em H, então tal ciclo pode ser representado por um ciclo alternado em G e vice-versa; consequentemente existe uma correspondência entre uma decomposição em ciclos de H e G. A Figura 3.6 ilustra esta correspondência.

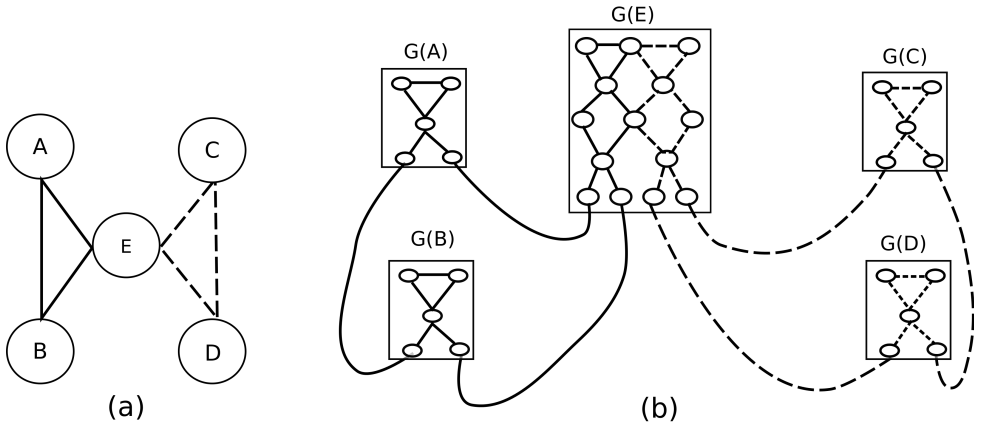

Figura 3.6: (a) Grafo Euleriano H contendo dois ciclos. (b) O grafo bi-colorido G representando os mesmos dois ciclos de H.

Para concluir, pode-se observar que G satisfaz as condições do Teorema 3.1. Consequentemente, G é um grafo de Pontos-de-Quebra. A redução é feita em tempo polinomial tomando $m = r(s-1)+1$. Logo, tem-se uma redução em tempo polinomial de *MAX-ECD* para *MAX-ACD*, e *MAX-ACD* é $\mathcal{NP}$-difícil.

3.2.4 MAX-ACD $\preceq_p$ DTS

Nesta seção, uma redução polinomial de *MAX-ACD* para *DTS* é dada seguindo a apresentação em [50]. Isto permite concluir que o último problema é $\mathcal{NP}$-difícil.

Sejam X e Y dois cromossomos sem sinal. Sem perca de generalidade, considere $X = (g_1, g_2, \ldots g_n)$ e $(Y = 1, 2, \ldots, n)$, onde $\{g_1, g_2, \ldots, g_n\} = \{1, 2, \ldots, n\}$ e $g_1 = 1$, $g_n = n$. A partir de X e Y, podem-se construir dois genomas $A = \{X_1, X_2\}$ e $B = \{Y_1, Y_2\}$ como será descrito na sequência. Além disso, considere um inteiro d usado para controlar o número de ciclos curtos na decomposição de $G_u(A, B)$; este número é detalhado no Lema 3.1. O cromossomo X_1 do genoma A é construído por inserção de $n-1$ novos genes entre dois genes adjacentes em X como segue:

$$X_1 = (1,\ t_{1,1},\ g_2,\ t_{1,2},\ \ldots,\ g_{n-1},\ t_{1,n-1},\ n)$$

onde, $t_{1,k} = 3n - 2 + k, 1 \leq k \leq n - 1$.
X_2 contém dois tipos de novos genes, $t_{2,l} = n + l$, $1 \leq l \leq 2(n-1)$ e $s_i = 4n - 3 + i$, $1 \leq i \leq (n-2)d$.

$$\begin{aligned} X_2 \;=\; & (t_{2,1},\ t_{2,2},\ s_1,\ s_2, \ldots, s_d, \\ & t_{2,3},\ t_{2,4},\ s_{d+1},\ s_{d+2}, \ldots, s_{2d}, \\ & \vdots \\ & t_{2,2(n-2)-1},\ t_{2,2(n-2)},\ s_{(n-3)d+1}, \ldots, s_{(n-2)d}, \\ & t_{2,2(n-1)-1},\ t_{2,2(n-1)}) \end{aligned}$$

Para construir o genoma $B = \{Y_1, Y_2\}$, considere os mesmos inteiros $t_{1,k}, t_{2,l}$ e s_i, $1 \leq k \leq n-1$, $1 \leq l \leq 2(n-1)$, $1 \leq i \leq (n-2)d$, como calculado em A. O cromossomo $Y_1 = Y = (1, 2, \ldots, n)$ e Y_2 é construído de X_2 inserindo $t_{1,k}$ entre $t_{2,2k-1}$ e $t_{2,2k}$ em X_2.

$$\begin{aligned} Y_2 \;=\; & (t_{2,1},\ t_{1,1},\ t_{2,2},\ s_1,\ s_2, \ldots, s_d, \\ & t_{2,3},\ t_{1,2},\ t_{2,4},\ s_{d+1}, \ldots, s_{2d}, \\ & \vdots \\ & t_{2,2(n-2)-1}, t_{1,n-2}, t_{2,2(n-2)}, s_{(n-3)d+1}, \ldots, s_{(n-2)d}, \\ & t_{2,2(n-1)-1},\ t_{1,n-1},\ t_{2,2(n-1)}) \end{aligned}$$

No fim da construção, cada um dos genomas A e B tem um número total de $4n - 3 + (n-2)d$ genes.
Exemplo. Seja $X = (1, 3, 4, 2, 5)$ e $Y = (1, 2, 3, 4, 5)$; Figura 3.7 (a) ilustra o grafo $G_u(X, Y)$. Considere $d = 4$. Assim, os genomas A e B são:

$$\begin{aligned} A = \;& \{X_1, X_2\},\ \text{onde} \\ X_1 = \;& (1, 14, 3, 15, 4, 16, 2, 17, 5)\ \text{e} \\ X_2 = \;& (6, 7, 18, 19, 20, 21, 8, 9, 22, 23, 24, \\ & 25, 10, 11, 26, 27, 28, 29, 12, 13) \end{aligned}$$

e

$$\begin{aligned} B = \;& \{Y_1, Y_2\},\ \text{onde} \\ Y_1 = \;& (1, 2, 3, 4, 5)\ \text{e} \\ Y_2 = \;& (6, 14, 7, 18, 19, 20, 21, 8, 15, 9, 22, 23, 24, \\ & 25, 10, 16, 11, 26, 27, 28, 29, 12, 17, 13) \end{aligned}$$

O grafo $G_u(A, B)$ é mostrado na Figura 3.7 (b). Os Lemas 3.1 e 3.2 esclarecem a relação entre uma máxima decomposição em ciclos alternados de $G_u(X, Y)$ e a distância de

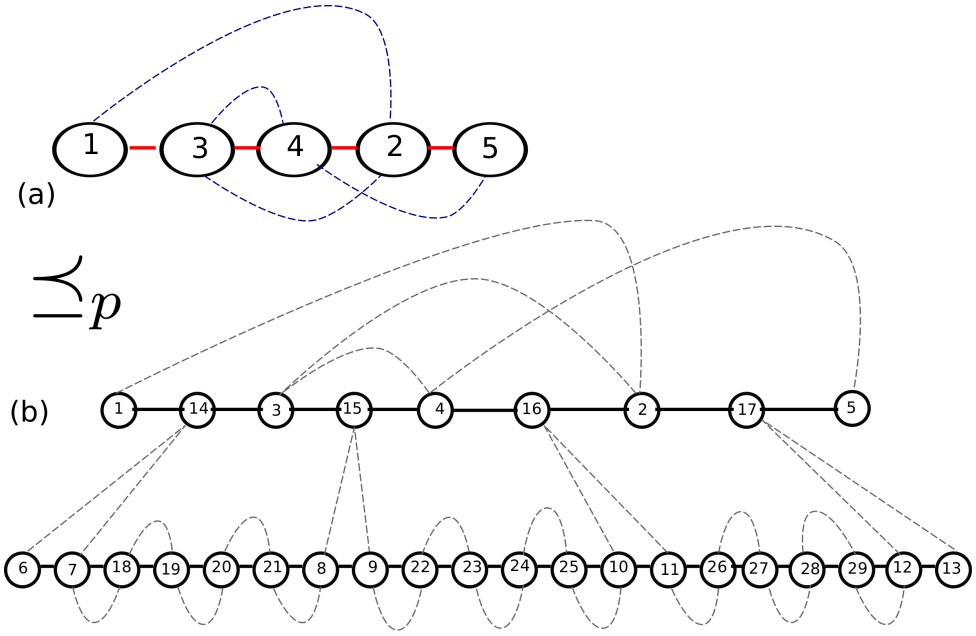

Figura 3.7: O grafo de Pontos-de-Quebra (a) $G_u(X, Y)$ e (b) $G_u(A, B)$

translocação entre dois genomas A e B.

Lema 3.1 (Lema 4 em [50]). *Assuma $d \geq n-1$. Há uma decomposição de $G_u(X, Y)$ em J ciclos alternados sse existe uma decomposição de $G_u(A, B)$ em pelo menos $(n-2)(d+1)+J$ ciclos alternados.*

Demonstração. Poucos detalhes são agregados a prova original dada em [50].
Suficiência: Assumindo que há uma decomposição M de $G_u(X, Y)$ em J ciclos alternados. A ideia consiste em associar de maneira unívoca para cada ciclo $C \in M$ um ciclo C' em $G_u(A, B)$, e com as arestas remanescentes de $G_u(A, B)$ construir $(n-2)(d+1)$ ciclos. Desta forma, tem-se uma decomposição de $G_u(A, B)$ em $(n-2)(d+1)+J$ ciclos alternados.

Para cada ciclo $C \in M$, C pode ser representado como uma lista $C = \{u_1, u_2, \ldots, u_{2k-1}, u_{2k}\}$, onde (u_{2i-1}, u_{2i}) é uma aresta preta e (u_{2i}, u_{2i+1}) é uma aresta cinza, $1 \leq i \leq k$ e $u_{2k+1} = u_1$. Note que se $u_{2i-1} = g_j$ então $u_{2i} = g_{j+1}$ or $u_{2i} = g_{j-1}$.

Um novo ciclo C' em $G_u(A, B)$ pode ser obtido substituindo cada aresta preta (u_{2i-i}, u_{2i}) de C por um caminho alternado $P_{2i-i,2_i}$, onde,

$$\begin{aligned} P_{2i-i,2i} &= g_j, t_{1,j}, t_{2,2j-1}, t_{2,2j}, t_{1,j}, g_{j+1} \\ &\quad \text{se } u_{2i-1} = g_j, u_{2i} = g_{j+1}; \\ P_{2i-i,2i} &= g_j, t_{1,j-1}, t_{2,2j-3}, t_{2,2j-2}, t_{1,j-1}, g_{j-1} \\ &\quad \text{se } u_{2i-1} = g_j, u_{2i} = g_{j-1}. \end{aligned}$$

Assim, para cada ciclo $C \in M$ um ciclo longo C' de $G_u(A, B)$ pode ser associado univocamente. Tal ciclo é longo porque cada caminho alternado $P_{2i-i,2i}$ tem 3 arestas pretas e 2 arestas cinzas (veja Figuras 3.7 e 3.8). Note que as únicas arestas de

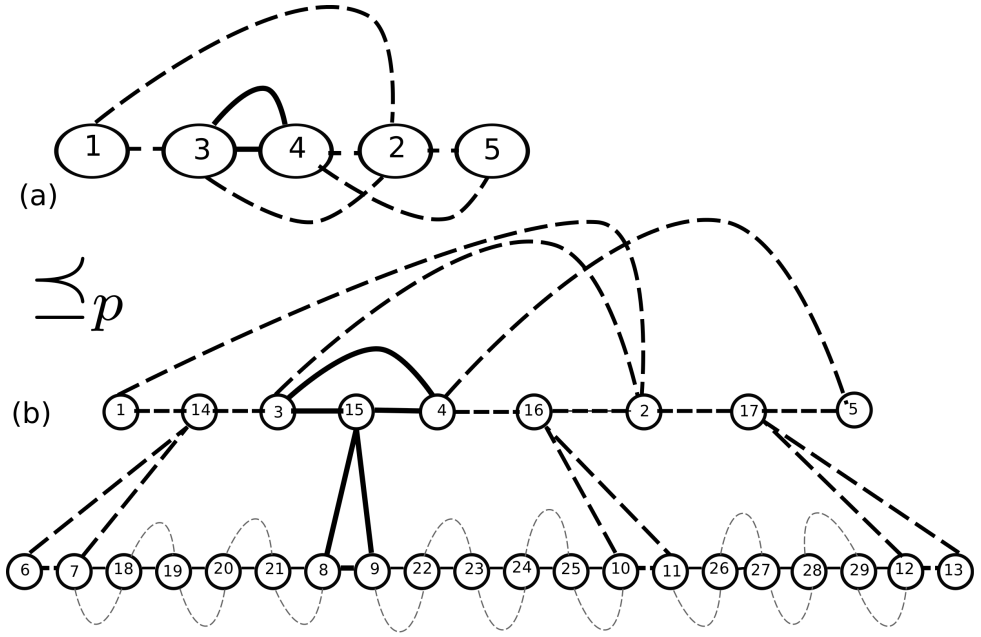

Figura 3.8: Ciclos alternados de $G_u(A, B)$ a partir de ciclos alternados de $G_u(X, Y)$

$G_u(A, B)$ não usadas para construir os J ciclos longos são as $d + 1$ arestas pretas e cinzas $(t_{2,2i}, s_{(i-1)d+1}), \ldots, (s_{id-1}, s_{id}), (s_{id}, t_{2,2i+1})$, cujos vértices pertencem a X_2 e Y_2, para $i \in \{1, \ldots, n-2\}$. Tais arestas são usadas para construir $(n-2)(d+1)$ ciclos curtos. Consequentemente, a decomposição M de $G_u(X, Y)$ em J ciclos alternados leva a uma decomposição de $G_u(A, B)$ em $(n-2)(d+1) + J$ ciclos alternados.

Necessidade: Seja M' um conjunto de $(n-2)(d+1)+J$ ciclos alternados formando uma decomposição de $G_u(A, B)$. Note que, unicamente as arestas $(t_{2,2i}, s_{(i-1)d+1}), \ldots, (s_{id-1}, s_{id})$, $(s_{id}, t_{2,2i+1})$, para $i \in \{1, \ldots, n-2\}$, podem formar ciclos curtos; consequentemente, existem pelo menos $(n-2)(d+1)$ ciclos curtos em M'. A ideia consiste em mostrar que sempre é possível construir uma decomposição de $G_u(A, B)$ com $(n-2)(d+1)$ ciclos curtos; e, para os outros J ciclos longos de $G_u(A, B)$ pode-se associar uma decomposição de J ciclos alternados de $G_u(X, Y)$.

Um ciclo alternado usando um vértice em X_1 deve ter no mínimo duas arestas pretas contendo unicamente vértices de X_1 e pelo menos duas arestas cinzas com um vértice em X_1 e outro em X_2. Sem perca de generalidade, suponha que há um ciclo $C = u_1, u_2, \ldots, u_{2k}$ de $G_u(A, B)$ tal que (u_{2i-1}, u_{2i}) é um aresta preta e (u_{2i}, u_{2i+1}) uma aresta cinza, $1 \leq i \leq k$, $u_{2k+1} = u_1$ e $u_1 = g_j$. Assim, $u_2 = t_{1,j-1}$ ou $u_2 = t_{1,j}$ e ambos os vértices de (u_1, u_2) estão

em X_1. A aresta cinza (u_2, u_3) satisfaz $u_3 = t_{2,2j-1}$ ou $u_3 = t_{2,2j}$. Assim, u_3 está em X_2. Note que (u_{2k}, u_1) é uma aresta cinza, assim $u_{2k} = g_l$, uma vez que $\{g_1, \ldots, g_n\} = \{1, \ldots, n\}$. Assumindo que $u_{2m-1} = t_{1,s}$ para algum $3 \leq m \leq k$. De fato, se $u_{2m-1} \neq t_{1,s}$ para qualquer $3 \leq m \leq k$, então (u_{2m-1}, u_{2m}) são arestas com vértices em X_2 e $u_{2k} \neq g_l$. Assim, a aresta preta $(u_{2m-1}, u_{2m}) = (t_{1,s}, g_t)$ contém vértices em X_1 e a aresta cinza $(u_{2m-2}, u_{2m-1}) = (u_{2m-2}, t_{1,s})$, onde $u_{2m-2} = t_{2,2s-1}$ ou $u_{2m-2} = t_{2,2s}$, contém um vértice em X_1 e outro em X_2.

Consequentemente, cada ciclo contendo um vértice em X_1 é um ciclo longo e, uma vez que X_1 tem $2n-1$ vértices, existem no máximo $n-1$ ciclos alternados em M' usando vértices em X_1. Para qualquer i, os $d+2$ vértices consecutivos $\{t_{2,2i}, s_{(i-1)d+1}, \ldots, s_{id-1}, s_{id}, s_{id}, t_{2,2i+1}\}$ em X_2 não podem estar em um mesmo ciclo em M'; caso contrário, o número de ciclos curtos será reduzido por $d + 1 \geq n$, por hipótese, e o máximo número de ciclos em M' deve ser

$$\begin{aligned}(n-3)(d+1) + (n-1) &= (n-2-1)(d+1) + n - 1 \\ &\leq (n-2)(d+1) - n + n - 1 \\ &< (n-2)(d+1) + J\end{aligned}$$

Assim, se um ciclo em M' contém uma das arestas cinzas $(t_{1,j}, t_{2,2j-1})$ e $(t_{1,j}, t_{2,2j})$, é necessário que este ciclo contenha ambas arestas cinzas; do contrário, existiria um ciclo em M' usando $d+2$ vértices consecutivos em X_2. Além disso, se um ciclo longo em M' faz uso de dois vértices consecutivos na sequência $(t_{2,2i}, s_{(i-1)d+1}), \ldots, (s_{id-1}, s_{id}), (s_{id}, t_{2,2i+1})$, tal ciclo deve conter ambas arestas cinzas e pretas entre estes dois vértices; então, este ciclo (longo) pode ser decomposto no intuito de incrementar o número de ciclos curtos. Assim, pode-se assumir que todos os $(n-2)(d+1)$ ciclos curtos estão em M'.

Finalmente, uma vez que todos os ciclos longos em $G_u(A, B)$ não podem unicamente conter arestas cinzas em X_1, cada ciclo longo em M' deve tomar o caminho $P_{1,2}, \ldots, P_{2k-1,2_k}$, onde $P_{2i-1,2i} = u_{2i-1}, t_{1,j}, t_{2,2j-1}, t_{2,2j}, t_{1,j}, u_{2i}$, e $\{u_{2i-1}, u_{2i}\} = \{g_j, g_{j+1}\}$. Substituindo o caminho $P_{2i-1,2i}$ pela aresta preta (u_{2i-1}, u_{2i}), um ciclo alternado em $G_u(X, Y)$ é obtido. Logo, $G_u(X, Y)$ pode ser decomposto em J ciclos alternados. □

Antes do Lema 3.2, é conveniente enunciar dois teoremas introduzidos em [50]. O primeiro relaciona distância de translocação e ciclos na decomposição do grafo de Pontos-de-Quebra para genomas com sinal. O segundo, distância de translocação entre genomas sem sinal.

Teorema 3.2 (Teorema 1 em [50])**.** *A distância de translocação entre dois genomas com sinal $\vec{A}$ e $\vec{B}$ satisfaz $d(\vec{A}, \vec{B}) \geq n - m - c_{AB}$. Se não existem subpermutações para $\vec{A}$ e*

$\vec{B}$, $d(\vec{A},\vec{B}) = n - m - c_{AB}$, *onde* n *é o número de genes,* m *o número de cromossomos e* c_{AB} *o número de ciclos na decomposição do grafo de Pontos-de-Quebra* $G_s(\vec{A},\vec{B})$.

Teorema 3.3 (Teorema 2 em [50])**.** *Sejam* A *e* B *genomas sem sinal. Considere* $\vec{B}$ *o genoma com sinal obtido de* B *por atribuir a cada gene um sinal positivo. Então,* $d(A,B) = min_{\vec{A}\in Spin(A)} d(\vec{A},\vec{B})$*, onde* $Spin(A)$ *é o conjunto de todos os genomas com sinal obtido de* A *atribuindo sinais para os seus genes.*

Lema 3.2 (Lema 5 em [50])**.** *Existe uma decomposição de* $G_u(A,B)$ *em* $(n-2)(d+1)+J$ *ciclos alternados sse* $d(A,B) \leq 3n-3-J$.

Demonstração. **Suficiência:** Se $G_u(A,B)$ pode ser decomposto em $(n-2)(d+1)+J$ ciclos alternados, há uma decomposição M' com $(n-2)(d+1)+J$ ciclos alternados de $G_u(A,B)$, onde cada ciclo longo possui uma aresta cinza contendo um vértice de X_1 e um vértice de X_2, como apresentado no Lema 3.1. A partir desta decomposição é possível construir genomas com sinal $\vec{A}$ e $\vec{B}$, tal que o número de ciclos alternados de $G_s(\vec{A},\vec{B})$ é o mesmo que em M' (veja Seção 3.4 em [28]); e porque cada ciclo longo em M' tem uma aresta cinza com um vértice em X_1 e um vértice em X_2. Além disso, não há subpermutações para $\vec{A}$ e $\vec{B}$, uma vez que unicamente ciclos longos geram subpermutações para $\vec{A}$ e $\vec{B}$ e todas as arestas de um ciclo alternado envolvido em uma subpermutação dever estar no mesmo cromossomo (veja [23]).

Seja n o número de genes em ambos os genomas A e B, N o número de cromossomos e c_{AB} o número de ciclos alternados na decomposição M' de $G_u(A,B)$. Do Teorema 3.3 tem-se $d(A,B) \leq d(\vec{A},\vec{B})$ e pelo Teorema 3.2,

$$\begin{aligned} d(\vec{A},\vec{B}) &= n - N - c_{AB} \\ &= 4n - 3 + (n-2)d - 2 - ((n-2)(d+1)+J) \\ &= 4n - 3 + dn - 2d - 2 - (dn + n - 2d - 2 + J) \\ &= 4n - 3 + dn - 2d - 2 - dn - n + 2d + 2 - J \\ &= 4n - n - 3 + 2 - 2 - 2d + 2d - J \\ &= 3n - 3 - J. \end{aligned}$$

Assim $d(A,B) \leq 3n-3-J$.

Necessidade: Seja $d(A,B) \leq 3n-3-J$. Pelo Teorema 3.3, há genomas $\vec{A}$ e $\vec{B}$ obtidos de A e B tal que $d(\vec{A},\vec{B}) \leq 3n-3-J$.

Utilizando o Teorema 3.2, $d(\vec{A},\vec{B}) \geq 4n-3+(n-2)d-2-c_{\vec{A}\vec{B}}$. Assim,

$$
\begin{aligned}
c_{\vec{A}\vec{B}} &\geq 4n - 3 + (n-2)d - 2 - d(\vec{A}, \vec{B}) \\
&\geq 4n - 3 + nd - 2d - 2 - 3n + 3 + J \\
&= n + nd - 2d - 2 + J \\
&= n(d+1) - 2(d+1) + J \\
&= (n-2)(d+1) + J.
\end{aligned}
$$

Portanto, o máximo número de ciclos alternados em uma decomposição de $G_u(A, B)$ é de no mínimo $(n-2)(d+1) + J$. □

Pelos Lemas 3.1 e 3.2, existe uma decomposição em J ciclos alternados de $G_u(X, Y)$, se e unicamente se, a distância de translocação ente A e B é dada por $3n - 3 - J$.

Para finalizar a prova, é necessário mostrar que a redução é realizada em tempo polinomial.
Considere uma instância de *MAX-ACD* contendo n genes e $d = n-1$. Assim, a correspondente instância de distância de translocação sem sinal tem $4n-3+(n-2).(n-1) = n^2+n-1$ genes. Consequentemente, há uma redução polinomial de *MAX-ACD* para o problema de distância de translocação sem sinal e o último problema é $\mathcal{NP}$-difícil.

3.3 Abordagens aproximadas para o Problema de Distância de Translocação sem Sinal

Antes de provar que o problema para a versão com genomas sem sinal está em $\mathcal{NP}$-difícil, Kececioglu e Ravi [30] desenharam um algoritmo de raio de aproximação 2. Em trabalhos mais recentes (a partir de 2005) voltou-se a intensificar a busca por melhores soluções aproximadas. A ideia destes novos algoritmos consiste em encontrar uma versão com sinal dos genomas sem sinal dados como entrada e assim utilizar um dos algoritmos apresentados na Seção 3.1 para computar a distância de translocação que será o resultado para os genomas sem sinal.

Yun Cui et al. propuseram em [13] um algoritmo com raio aproximado 1.75 com complexidade de tempo $O(n^2)$. O método utilizado no algoritmo consiste em buscar todos os 1-ciclos (ciclos contendo exatamente uma aresta cinza e uma preta) no grafo de Pontos-de-Quebra. Após identificar os 1-ciclos, é utilizado o método de máximo emparelhamento para encontrar um ciclo de decomposição que contenha um número suficiente de 2-ciclos (ciclos contendo exatamente duas arestas cinzas e pretas). Na sequência, cada gene sem sinal destes 1-ciclos e 2-ciclos tem sinais fixados de forma a obterem os mesmos 1-ciclos

e 2-ciclos na versão com sinal dos genomas sem sinal dados como entrada. É importante enfatizar que após atribuir sinais para todos os genes pertencentes aos 1-ciclos e 2-ciclos, há a chance de se terem remanescentes genes sem sinais. Assim, atribuísse arbitrariamente sinais para estes genes.

Os mesmos autores de [13], forneceram um novo algoritmo aproximado com raio $1.5+\varepsilon$ ($\varepsilon > 0$) em [14] com complexidade de tempo $O(n^2 + (\frac{4}{\varepsilon})^{1.5}.\sqrt{\log(\frac{4}{\varepsilon})2^{\frac{4}{\varepsilon}}})$. Tal algoritmo é um aprimoramento do algoritmo de raio 1.75. Neste novo algoritmo, após computar todos os 1-ciclos e 2-ciclos, mantém-se o foco nos 2-ciclos afim de encontrar um tipo especial de 2-ciclo denominado ciclo removível, tais ciclos removíveis são destruídos e não farão parte da solução final. No fim, atribuem-se sinais para os genes sem sinais para criar todos os 1-ciclos e 2-ciclos (para os restantes) e para os remanescentes genes, conforme explicado no parágrafo anterior.

Atualmente o algoritmo que provê o melhor raio de aproximação ($1.408 + \varepsilon$, onde $\varepsilon = \frac{1}{\log n}$) é o proposto por Haitao Jiang et al. em [28] de complexidade $O(n^2+n\log^2 n \log\log n)$. O algoritmo segue a mesma abordagem dos anteriores algoritmos aproximados. Primeiramente, encontra todos os 1-ciclos. Em seguida, identifica todas as *SP* com tamanho $n \log n$, então em tempo polinomial obtém todas as formas possíveis de decomposição de tais subpermutações (que seria todas as versões com sinal de uma subpermutação) e escolhe para cada subpermutação a versão que maximiza o número de ciclos e minimize o número de *MinSP* (*SP* e *MinSP* são definidas na Seção 2.1.2). Para os genes que não fizeram parte das *SPs*, utilizam-se duas rotinas envolvendo soluções de problemas $\mathcal{NP}$-completos [29] como segue. A primeira utiliza um algoritmo aproximado para computar o máximo conjunto independente em grafos com grau no máximo 4 para computar os 2-ciclos. A segunda utiliza um algoritmo aproximado para computar o máximo conjunto de empacotamento em conjuntos de tamanho no máximo 3 na busca por 2-ciclos e 3-ciclos (que não compartilhem genes). Após isto, escolhe-se o ciclo de decomposição com o maior número de ciclos providos através das subrotinas citadas acima. Caso restem genes sem sinais, atribuem-se sinais arbitrariamente a fim de obter versões com sinal dos genomas sem sinal dados como entrada.

Note que, estes algoritmos apresentados acima são compostos de métodos computacionais de difícil implementação, principalmente o algoritmo de raio $1.408 + \varepsilon$. Além disso, na literatura não é encontrada nenhuma implementação de nenhum deles (verificado com os próprios autores). Sendo assim, como parte deste trabalho foi implementado um destes algoritmos, o de raio $1.5 + \varepsilon$ introduzido em [14].

Capítulo 4

Algoritmo Aproximado de Raio $1.5 + \varepsilon$ para DTS

Neste capítulo será mostrado o método utilizado por Yun Cui et al. em [14] para aproximar a distância de translocação entre genomas A e B sem sinal, tendo como estratégia computar os ciclos de decomposição de $G_u(A, B)$, e a partir desta decomposição atribuir sinais para os genes de A e B. Uma vez que se tem $\vec{A}$ e $\vec{B}$, pode-se calcular a distância entre eles. Tal algoritmo é usado como mecanismo de controle para as soluções fornecidas pelo algoritmo genético proposto, o qual será descrito no Capítulo 5. Na literatura, as melhores soluções para o problema DTS são providas pelo algoritmo de raio $1.408 + \varepsilon$, contudo utilizar o algoritmo de raio $1.5 + \varepsilon$ foi mais vantajoso devido às seguintes circunstâncias.

1. As rotinas chave do algoritmo $1.408 + \varepsilon$, utilizadas para computarem soluções aproximadas para máximo conjunto de empacotamento com tamanho no máximo 3 em [22] e máximo conjunto independente com grau no máximo 4 em [6] são complexas e imprecisas em se tratando de detalhes de implementação. Por ser um algoritmo com um grau de complexidade elevada em se tratando de implementação, seria necessário dispor de muito tempo para estudar as rotinas citadas anteriormente e implementá-las adequadamente, além de outras partes que envolve um algoritmo aproximado como a prova do raio da qualidade das soluções, o qual não consta em [28]. Assim, considerou-se que não seria possível fornecer uma implementação adequada com soluções que apresentassem garantia matemática de precisão dentro do raio de $1408 + \varepsilon$ estabelecido teoricamente durante o curto espaço que se tem em um mestrado sem que isso afetasse o equilíbrio da pesquisa, uma vez que a pesquisa relacionada ao tema foi abordada unicamente durante os 2 anos de que se cursou o mestrado. Em contrapartida, ao analisar o algoritmo aproximado de raio $1.5 + \varepsilon$, a sua implementação se apresentava bem mais fácil de se alcançar em um espaço de

tempo aceitável (implementação levou cerca de 2 meses) quando comparado com o algoritmo de raio $1408+\varepsilon$, visto que os autores detalharam com os mínimos detalhes tanto a prova do raio da qualidade das soluções quanto os passos que o algoritmo possui, expondo todo o conhecimento teórico envolvido.

2. A qualidade das soluções providas pelo algoritmo aproximado $1.5 + \varepsilon$ é suficiente para os nossos objetivos. As respostas fornecidas por este algoritmo não estão tão distantes daquelas fornecidas do raio do algoritmo aproximado $1.408 + \varepsilon$, quando considerado o tamanho dos genomas utilizados nos experimentos (contendo no máximo 150 genes).

O capítulo possui a seguinte organização. A Seção 4.1 apresenta a fórmula fornecida por Hannehalli em [23] para computar a distância exata entre genomas com sinal. O algoritmo de raio $1.5+\varepsilon$ é uma continuação do algoritmo 1.75 (são providos pelos mesmos autores). Assim, na Seção 4.2 é descrito de forma resumida o algoritmo de raio 1.75 proposto em [13]. Na Seção 4.3 é apresentada a estratégia adotada no algoritmo de raio $1.5 + \varepsilon$ para prover resultados melhores que os fornecidos pelo algoritmo de raio 1.75. Na sequência, a Seção 4.4 descreve os passos necessários para implementar o algoritmo aproximado de raio $1.5 + \varepsilon$ reportado em [14], e apresenta a análise de complexidade de tempo do mesmo. Por fim, a Seção 4.5 apresenta a prova fornecida por Yun Cui et al. em [14], que as soluções fornecidas pelo algoritmo aproximado estão dentro de um raio de $1.5 + \varepsilon$ da solução ótima.

4.1 Fórmula de Hannenhalli

Por questão de compatibilidade com os autores, utiliza-se a equação fornecida por Hannenhalli para verificar a *DTC*. Tal fórmula faz parte da demonstração da qualidade das soluções providas pelo algoritmo aproximado de raio $1.5 + \varepsilon$ fornecida em [14] e apresentada na Seção 4.5.

A $d_s(\vec{A}, \vec{B})$ está fortemente relacionada com o número de ciclos e o número de *minSPs*. Se todas as *minSPs* no grafo de Pontos-de-Quebra $G_s(\vec{A}, \vec{B})$ estão em uma única *SP* e o número de *minSPs* é par, se tem uma isolação par. Claramente, existe uma única isolação em $G_s(\vec{A}, \vec{B})$.

Seja n o número de genes e N o número de cromossomos para os dois genomas $\vec{A}$ e $\vec{B}$. Seja c o número de ciclos e s o número de *minSPs* em $G_s(\vec{A}, \vec{B})$. O índice remanescente f é definido como segue:
1) $f = 1$ se s é ímpar.
2) $f = 2$ se existe uma isolação par.

3) $f = 0$ caso contrário.
O Lema a seguir, o qual está enunciado em [14], apresenta a fórmula demonstrada por Hannehalli em [23] para computar a distância ótima de translocação $d_s(\vec{A}, \vec{B})$ entre dois genomas com sinal.

Lema 4.1. *(Lema 2 em [14])* $d_s(\vec{A}, \vec{B}) = n - N - c + s + f$.

4.2 Algoritmo Aproximado de Raio 1.75

O algoritmo de raio aproximado 1.75 em [13] busca uma decomposição em ciclos do grafo de Pontos-de-Quebra sem sinal $G_u(A, B)$ que contenha o máximo número de 1-ciclos e 2-ciclos. Para alcançar soluções com raio aproximado $1.5+\varepsilon$, essa estratégia foi mantida. Assim, há a necessidade de uma breve revisão do algoritmo de decomposição em ciclos do raio 1.75.

Na busca por uma decomposição em ciclos, mantém-se o máximo número de 1-ciclos, que é, se dois vértices são unidos por uma aresta preta e uma aresta cinza em $G_u(A, B)$, então atribui-se sinal adequado aos dois genes para obter o 1-ciclo. Considere como exemplo, o grafo $G_u(A, B)$ (Figura 4.1(a)) construído a partir de dois genomas sem sinal A e B, onde
$A = \{(1, 2, 7, 5, 13, 9, 11, 10, 12, 8, 14, 4, 6, 3, 15), (16, 17, 23, 19, 21, 20, 22, 18, 24)\}$ e
$B = \{(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15), (16, 17, 18, 19, 20, 21, 22, 23, 24)\}$.
A Figura 4.1(b) ilustra como ficaria o genoma resultante $(\vec{A})$ após identificar todos os 1-ciclos.

Após obter o máximo número de 1-ciclos, busca-se uma máxima quantidade de 2-ciclos

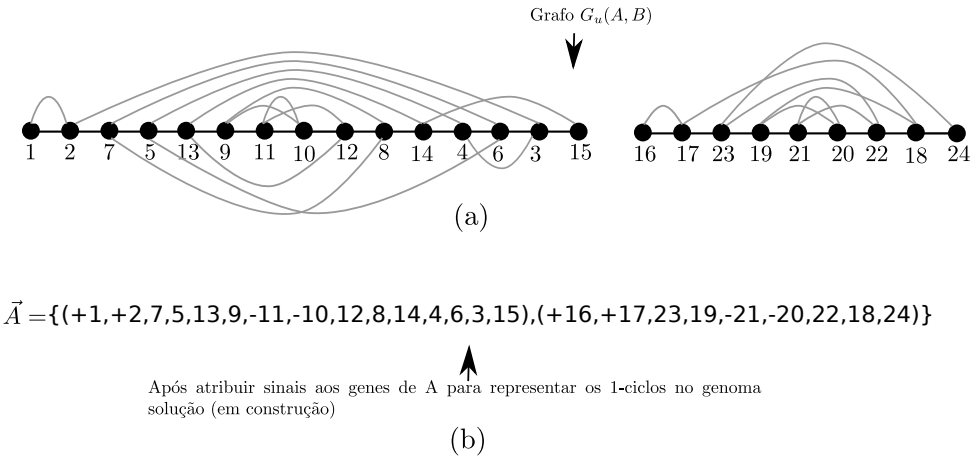

Figura 4.1: Mapeando os 1-ciclos em $G_u(A, B)$.

em tempo polinomial.

Um grafo de emparelhamento F_{AB} do grafo $G_u(A, B)$ é definido como segue:
1) Para cada dois genes em sequência no genoma solução (após identificar os 1-ciclos) com pelo menos um deles sem sinal, cria-se um vértice em F_{AB}. Figura 4.2 ilustra um exemplo.

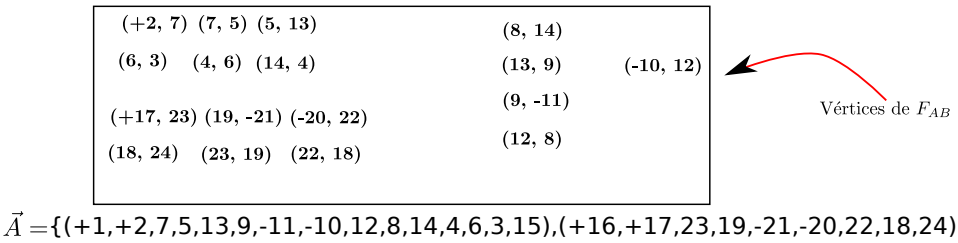

Figura 4.2: Vértices do grafo F_{AB} construídos a partir do genoma resultante $\vec{A}$ da Figura 4.1(a).

2) Para cada dois vértices em F_{AB}, cria-se uma aresta conectando-os em F_{AB} se as duas arestas pretas em $G_u(A, B)$ podem formar um 2-ciclo. A Figura 4.3 ilustra um exemplo considerando o conjunto de vértices apresentado na Figura 4.2.

Seja M o máximo emparelhamento de $G_u(A, B)$ (qualquer emparelhamento máximo é

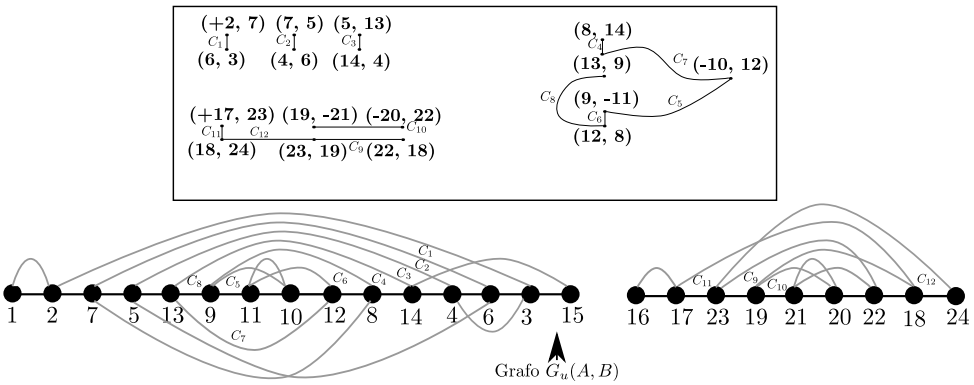

Figura 4.3: Arestas do Grafo F_{AB} construídas a partir do grafo $G_u(A, B)$ da Figura 4.1(a).

válido). Cada aresta em M representa um 2-ciclo em $G_u(A, B)$. Pela construção, dois 2-ciclos em M não podem compartilhar qualquer aresta preta de $G_u(A, B)$. Todavia, eles podem compartilhar uma aresta cinza em $G_u(A, B)$. Neste caso, os dois 2-ciclos não podem ser mantidos na decomposição em ciclos simultaneamente. Um 2-ciclo em M é isolado se não compartilha qualquer aresta com qualquer outro 2-ciclo. Caso contrário, o 2-ciclo é relacionado. Uma vez que um 2-ciclo tem 2 arestas cinzas, ele se relaciona no máximo com dois 2-ciclos.

Uma componente relacionada U consiste de 2-ciclos relacionados $c_1, c_2, \cdots, c_k$, onde c_i está relacionado com c_{i-1} $(2 \leq i \leq k)$, e cada 2-ciclo não se relaciona com qualquer outro

2-ciclo fora de U. Considere dois 2-ciclos relacionados c_i e c_{i-1} compartilhando uma aresta cinza em uma componente U. Os dois 2-ciclos também compartilham os genes finais da aresta cinza. Assim, tem-se:

1. Se as duas arestas pretas em c_i estão no mesmo cromossomo, o mesmo acontece para c_{i-1}.

2. Se as duas arestas pretas estão em dois cromossomos diferentes em c_i, então as duas arestas pretas em c_{i-1} também estão nos mesmos dois cromossomos.

O processo pode ser repetido no sentido esquerda para direita no genoma A para incluir mais ciclos na componente, onde tem-se como último 2-ciclo relacionado na componente o 2-ciclo mais interno em A. Portanto, uma componente relacionada envolve no máximo dois cromossomos e pode ser unicamente um dos quatro tipos, conforme Figura 4.4.

O algoritmo do raio de aproximação 1.75 consiste de 7 passos.
1) Construir o grafo $G_u(A, B)$.
2) Manter todos os 1-ciclos na decomposição do grafo $G_u(A, B)$.
3)Construir o grafo F_{AB} de $G_u(A, B)$, e computar o máximo emparelhamento M de F_{AB}. Computar todos os 2-ciclos isolados e respectivamente as componentes relacionadas.
4) Manter todos os 2-ciclos isolados na decomposição e selecionar 2-ciclos alternados (ao escolher C_{i-1} o 2-ciclo C_i que está compartilhando uma aresta cinza é desprezado) para cada componente relacionada U.
5) Arbitrariamente, fazer a distribuição de sinais nos vértices restantes completando a decomposição em ciclos do grafo $G_u(A, B)$.
6)Atribuir sinais positivos para todos os genes de B.
7) Computar a distância de translocação para o genoma solução.

Algoritmo 1 apresenta o pseudo-código do algoritmo aproximado de raio 1.75 guiado pelos passos apresentados acima.

4.3 Simples MinSPs Removíveis

Uma *minSP* contém pelo menos um ciclo longo. Uma *minSP* simples (*S-MSP*) é uma *minSP* contendo um 2-ciclo como sendo o único ciclo longo e pelo menos um 1-ciclo, além das arestas cinzas pertencentes ao 2-ciclo serem cruzadas. Na figura 4.5, o segmento de genes $x_l, \cdots, x_t$ formam uma *S-MSP*.

Seja $I_s = x_l, x_{l+1}, \cdots, x_{t-1}, x_t$ uma *S-MSP*. Uma *minSP* simples removível (*RS-MSP*) I_s é uma *minSP* simples tal que as duas arestas pretas $(R(x_{l-1}), L(x_l))$ e $(R(x_t), L(x_{t+1}))$ emolduram I_s pertencente a um ciclo cruzado e pelo menos uma das arestas cinzas $(R(x_{l-1}), R(x_t))$ ou $(L(x_l), L(x_{t+1}))$ estão no ciclo cruzado. As arestas cinzas $(R(x_{l-1}), R(x_t))$

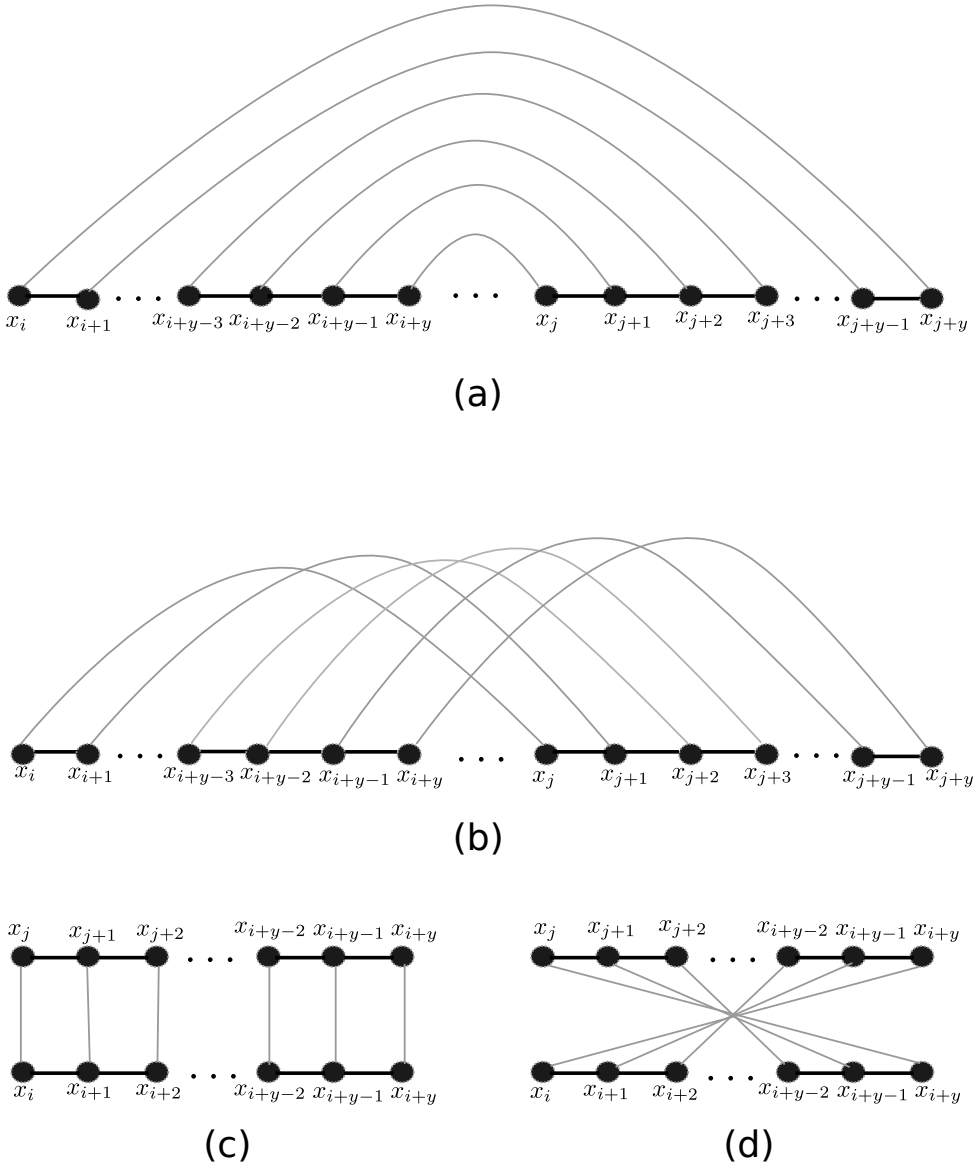

Figura 4.4: (a) e (b) As componentes estão em um único cromossomo. (c) e (d) Dois cromossomos estão envolvidos nas componentes.

ou $(L(x_l), L(x_{t+1}))$ são chamadas arestas chaves de I_s e tal ciclo cruzado é chamado ciclo removível de I_s. A figura 4.5(a) mostra um exemplo.

Dados genomas A e B, uma simples *minSP* candidata a ser removível (*CRS-MSP*) é definida como um intervalo $I_c = x_i, x_{i+1}, \cdots, x_{i+m}$ contendo pelo menos quatro genes em um cromossomo de A tal que:

1) Existe um intervalo de mesmo tamanho $y_j, y_{j+1}, \cdots y_{j+m}$ em algum cromossomo de B satisfazendo $x_i = y_j, x_{i+m} = y_{j+m}$ e $x_{i+k} = y_{j+m-k}$ para $(1 \leq k \leq m-1)$.

2) Pelo menos uma das arestas cinzas (x_{i-1}, x_{i+m}) ou (x_i, x_{i+m+1}) existe.

Assim, qualquer *CRS-MSP* pode-se tornar uma *RS-MSP*, como segue:

2.1) Atribuir adequadamente os sinais de todos os genes para criar uma *S-MSP*.

Algoritmo 1: Algoritmo 1.75 para DTS

Input: Genomas sem sinais A e B
Output: Distância aproximada

```
Construir o grafo de Pontos-de-Quebra G_u(A, B);
Computar todos os possíveis 1-ciclos em A;
Construir o grafo F_AB;
Computar o máximo emparelhamento M de F_AB;
Computar 2-ciclos isolados e componentes relacionados em M;
Distribuir apropriadamente sinais para os isolados 2-ciclos;
Distribuir apropriadamente sinais para as componentes;
Tomar os sinais distribuídos nos prévios passos e atribuir aos genes de A;
if Há genes sem sinais em A then
    Distribuir arbitrariamente sinais para os remanescentes genes em A;
end
Atribuir sinais positivos para cada genes em B;
Calcular a distância d_s(A⃗, B⃗);
```

2.2) Atribuir adequadamente sinais para x_{i-1} ou x_{i+m+1} tal que pelo menos uma das arestas cinzas $(R(x_{i-1}), R(x_{i+m}))$ ou $(L(x_i), L(x_{i+m+1}))$ esteja no grafo de Pontos-de-Quebra com sinal.

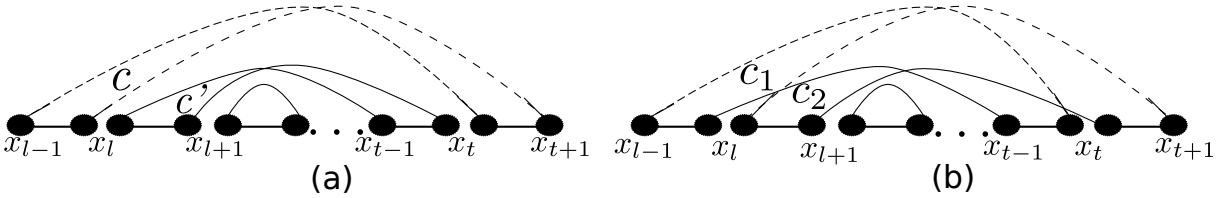

Figura 4.5: (a) A *RS-MSP*. (b) $G_s(\vec{A}, \vec{B})$ após remover a *RS-MSP* invertendo os sinais dos genes x_l e x_t .

Lema 4.2 (Modificando os genes extremos de uma *RS-MSP* Lema 5 em [14])**.** *Dado uma RS-MSP* $I_s = x_l, x_{l+1}, \cdots, x_{t-1}, x_t$*, ao modificar os sinais de* x_l *e* x_t*, tem-se:*

1. $I_s = -x_l, x_{l+1}, \cdots, x_{t-1}, -x_t$ *não é mais uma minSP.*
2. *O número de minSPs não é incrementado.*
3. *O número de i-ciclos no novo grafo de Pontos-de-Quebra permanece o mesmo.*

Demonstração. **Caso 1):** Ao modificar os sinais de x_l e x_t tem-se $I_s = -x_l, x_{l+1}, \cdots, x_{t-1}, -x_t$. Assim, I_s não é mais uma *minSP*. Uma vez que, a extremidade mais à esquerda

e mais à direita não é mais o menor e o maior valor respectivamente deste segmento, propriedade básica de uma *minSP*.

Caso 2): Seja I_s o segmento após modificar os sinais x_l e x_t, pelo caso 1 tem-se que I_s não é mais uma *minSP*, e o segmento I_s não contém qualquer outra *minSP*. Logo, a única possível nova *minSP* a ser criada vem de uma subpermutação *SP* que continha o segmento I_s, neste caso, a *SP* se torna uma nova *minSP*. Assim, o número de *minSPs* não é incrementado.

Caso 3): Se I_s é uma *RS-MSP*, então o grafo $G_s(\vec{A}, \vec{B})$ contém pelo menos uma das arestas cinzas chaves $(R(x_{l-1}), R(x_t))$ e $(L(x_l), L(x_{t+1}))$. Alterando os sinais de x_l e x_t tem-se dois novos ciclos c_1 e c_2. Veja figura 4.5(b). Assuma que o ciclo removível c é um i-ciclo ($i \geq 2$). Por definição, pelo menos uma das arestas cinzas chaves está em c. Assim, se há apenas uma delas, um dos dois i-ciclos c_1 e c_2 é um 2-ciclo e o outro é um i-ciclo ($i > 2$). Se c contém ambas as arestas, tanto c_1 e c_2 serão 2-ciclos, veja Figuras 4.5(a) e 4.5(b). Portanto, o número de i-ciclos no novo grafo de Pontos-de-Quebra $G_s(\vec{A}, \vec{B})$ permanecem os mesmos. □

4.4 Algoritmo de Decomposição aproximado de $1.5 + \varepsilon$

O algoritmo de raio $1.5 + \varepsilon$ é uma extensão do algoritmo de raio 1.75, onde se tem os passos utilizados no algoritmo de raio 1.75 mantidos e incorporados novos passos. Os novos passos incorporados em relação aos do algoritmo 1.75, estão destacados em itálico:
1) Construir o grafo $G_u(A, B)$.
2) Manter todos os 1-ciclos na decomposição do grafo $G_u(A, B)$.
3) Construir o grafo F_{AB} de $G_u(A, B)$, e computar o máximo emparelhamento M de F_{AB}. Computar todos os 2-ciclos isolados e respectivamente as componentes relacionadas.
4) *Para cada 2-ciclo isolado em M, se possível, criar uma RS-MSP utilizando o procedimento apresentado na Seção 4.4.1.*
5) *Selecionar todos os 2-ciclos isolados (exceto os conflitantes com as RS-MSPs criadas no passo anterior). Para cada componente relacionada, excluir 2-ciclos conflitantes com as RS-MSPs e selecionar os remanescentes 2-ciclos relacionados, alternando entre selecionar um e ignorar o próximo que compartilha uma aresta com tal 2-ciclo selecionado.*
6) Para as partes remanescentes, arbitrariamente fazer a decomposição.
7) *Seja $G_s^5(\vec{A}, \vec{B})$ o grafo de Pontos-de-Quebra obtido dos passos 1-6. Remover todas as RS-MSPs em $G_s^5(\vec{A}, \vec{B})$ conforme apresentado na Seção 4.4.1.*
8)Atribuir sinais positivos para todos os genes de B.

9)Computar a distância de translocação para o genoma solução.

A partir dos passos apresentados acima, foi abstraído o Algoritmo 2, de como computar soluções aproximadas de raio $1.5 + \varepsilon$.

Algoritmo 2: Algoritmo $1.5+\varepsilon$ para DTS

Input: Genomas sem sinais A e B

Output: Distância aproximada

```
1  Construir o grafo de Pontos-de-Quebras G_u(A, B);
2  Computar todos os possíveis 1-ciclos em A;
3  Construir o grafo F_AB;
4  Computar o máximo emparelhamento M de F_AB;
5  Computar 2-ciclos isolados e componentes relacionados em M;
6  Construir as S-MSPs a partir de M (Alg. 3);
7  Construir todas as possíveis RS-MSPs de isolados 2-ciclos (Alg. 5);
8  Distribuir apropriadamente sinais para os isolados 2-ciclos;
9  Distribuir apropriadamente sinais para as componentes;
10 Remover todas RS-MSPs (Alg. 4);
11 Tomar os sinais distribuídos nos prévios passos e atribuir aos genes de A;
12 if Há genes sem sinais em A then
13 |  Distribuir arbitrariamente sinais para os remanescentes genes em A;
14 end
15 Atribuir sinais positivos para cada genes em B;
16 Calcular a distância d_s(A⃗, B⃗);
```

4.4.1 Criando e Destruindo RS-MSPs

Após designar sinais para os genes internos ao genoma A para identificar todos 1-ciclos e construir o máximo emparelhamento M do grafo F_{AB}, o foco estará nos 2-ciclos isolados em M que podem ser únicos 2-ciclos para *S-MSPs*. Por definição tem-se que este tipo especial de 2-ciclo é um 2-ciclo cruzado, com todas as arestas pretas entre as suas duas arestas pretas envolvidas em 1-ciclos. O procedimento apresentado no Algoritmo 3 é utilizado pelo Algoritmo 2 para criar as *S-MSPs*.

Seja $I_s = x_l, x_{l+1}, \cdots, x_t$ uma *S-MSP*. Se existe aresta cinza (x_{l-1}, x_t) ou (x_l, x_{t+1}) no grafo $G_u(A, B)$, então pode-se criar aresta cinza $(R(x_{l-1}), R(x_t))$ ou $(L(x_l), L(x_{t+1}))$ no grafo de Pontos-de-Quebra com sinal $G_s(\vec{A}, \vec{B})$, transformando I_s em uma *RS-MSP*. Contudo, quando cria-se uma aresta cinza $(R(x_{l-1}), R(x_t))$ ou $(L(x_l), L(x_{t+1}))$ alguns 2-ciclos em M podem ser destruídos. Veja Figura 4.6(a) e 4.6(b), se aresta cinza $(L(x_i), L(x_{j+1}))$

Algoritmo 3: Procedimento para identificar *S-MSPs*

Input: Máximo emparelhamento M do grafo F_{AB}
Output: Estrutura X contendo todas as *S-MSPs*

```
1  X=∅;
2  índice=0;
3  for i até tamanho(M) do
4  |  if M[i] é um 2-ciclo isolado then
   |  |  /* I_s = x_l, x_{l+1}, ··· , x_t                          */
5  |  |  I_s=M[i];
6  |  |  if I_s contém unicamente 1-ciclos entre x_l e x_t then
7  |  |  |  X[índice]=I_s;
8  |  |  |  índice=índice+1;
9  |  |  end
10 |  end
11 end
```

é criada no grafo $G_s(\vec{A}, \vec{B})$ (ou seja atribuem-se sinais para os dois genes x_i e x_{j+1} de modo que a aresta $(L(x_i), L(x_{j+1}))$ seja criada), c_1 e c_2 não estarão no grafo $G_s(\vec{A}, \vec{B})$, pois serão destruídos, similarmente, se $(R(x_{l-1}), R(x_t))$ é criada, então c_1 e c_2 também serão destruídas, veja Figura 4.6(c) e 4.6(d).

Tem-se um 2-ciclo c em M conflitante com arestas cinzas $(R(x_{l-1}), R(x_t))$ ou $(L(x_l), L(x_{t+1}))$, se mantendo $(R(x_{l-1}), R(x_t))$ ou $(L(x_l), L(x_{t+1}))$, c é destruído. Portanto, um ciclo conflitante deve conter uma das duas arestas pretas $(L(x_{i-1}), L(x_i))$ e $(R(x_j), L(x_{j+1}))$.

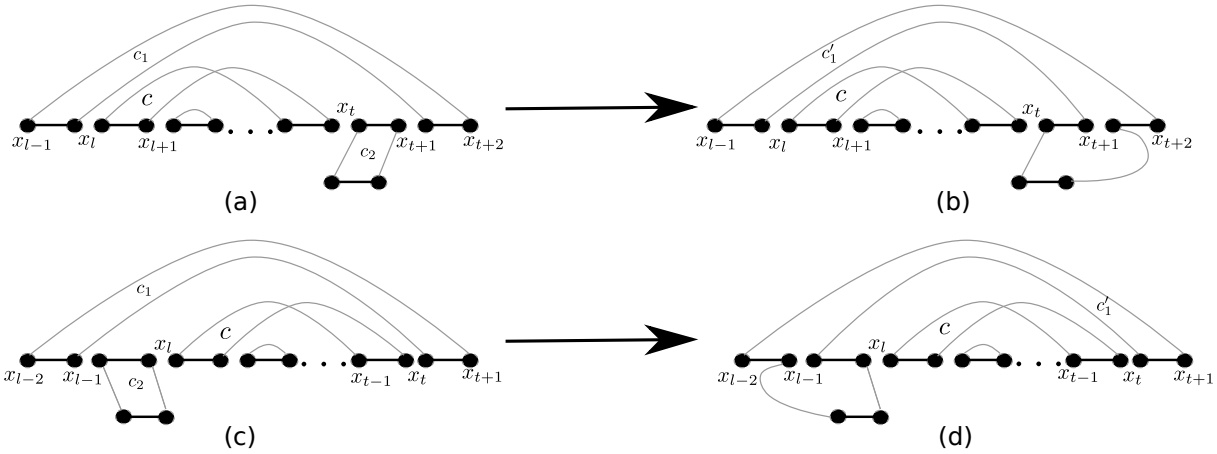

Figura 4.6: 2-ciclos conflitantes. (a) Antes de criar aresta cinza $(L(x_i), L(x_{j+1}))$. (b) Após criar aresta cinza $(L(x_i), L(x_{j+1}))$. (c) Antes de criar aresta cinza $(R(x_{i-1}), R(x_j))$. (d) Após criar aresta cinza $(R(x_{i-1}), R(x_j))$.

O procedimento apresentado no Algoritmo 5 é utilizado para criar as *RS-MSPs*. Uma *RS-MSP* não vai ser criada caso dois 2-ciclos conflitantes sejam destruídos com as arestas

cinzas $(R(x_{l-1}), R(x_t))$ e $(L(x_l), L(x_{t+1}))$.

Com relação a remoção de tais *RS-MSPs*, o Lema **4.2** mostra que ao remover uma *RS-MSP*, o número de i-ciclos não se altera no grafo de Pontos-de-Quebra resultante. Deste modo, o algoritmo de raio $1.5+\varepsilon$ remove todas as *RS-MSPs* encontradas utilizando o procedimento apresentado no Algoritmo 4. A ideia consiste em: Dado uma *RS-MSP* I_s, onde $I_s = (x_l, x_{l+1}, \cdots, x_t)$, altera-se os sinais de x_l e x_t conforme ilustrado na Figura 4.5.

Algoritmo 4: Procedimento para remover *RS-MSPs*

Input: Estrutura Q contendo todas as *RS-MSPs*

Output: Estrutura Q contendo todas as *RS-MSPs* removidas

```
1 for i até tamanho(Q) do
  |  /* I_s = x_l, x_{l+1}, ··· , x_t                    */
2 |  I_s=Q[i];
3 |  Inverter os sinais de x_l e x_t;
4 end
```

4.4.2 Implementação do Algoritmo de Aproximação

A seguir, serão dados os detalhes relacionados as representações das estruturas de dados utilizadas para representar: o grafo de Pontos-de-Quebra $G_u(A, B)$, o grafo F_{AB} utilizado para abstrair o máximo emparelhamento dos 2-ciclos bem como a representação das componentes apresentadas na Seção 4.2. Por fim é apresentada a complexidade do Algoritmo 2.

Estruturas de Dados Utilizadas

Ao iniciar a implementação de um algoritmo, a escolha adequada das estruturas de dados utilizadas é de grande impacto no tempo computacional.

Inicialmente, será descrita a estrutura utilizada para representar os genomas. Para facilitar os cálculos utilizados nos procedimentos do algoritmo, utiliza-se um arranjo de inteiros de tamanho n. A separação entre um cromossomo e outro é representada pelo

inteiro 0, veja Figura 4.8.

Algoritmo 5: Procedimento para criar *RS-MSPs*

Input: Grafo $G_u(A, B)$, máximo emparelhamento M e uma estrutura X contendo *S-MSPs*

Output: Estrutura Q contendo todas as *RS-MSPs*

```
1  Q=∅, índice=0;
2  for i até tamanho(X) do
3      marcador=0;
       /* I_s = x_l, x_{l+1}, ··· , x_t                                  */
4      I_s=X[i];
5      if Existe aresta cinza (x_{l-1}, x_t) then
6          marcador=1, cont=0;
7          Atribuir sinais adequados para criar aresta cinza (R(x_{l-1}), R(x_t));
8          for i até tamanho(M) do
9              if M[i] é um 2-ciclo isolado e é conflitante com (R(x_{l-1}), R(x_t)) then
10                 cont=cont+1;
11             end
12         end
13         if cont<2 then
14             Manter (R(x_{l-1}), R(x_t)) em I_S;
15         end
16     end
17     if Existe aresta cinza (x_l, x_{t+1}) em G_u(A, B) then
18         marcador=1, cont1=0;
19         Atribuir sinais adequados para criar aresta cinza (R(x_l), R(x_{t+1}));
20         for i até tamanho(M) do
21             if M[i] é um 2-ciclo isolado e é conflitante com (R(x_l), R(x_{t+1})) then
22                 cont1=cont1+1;
23             end
24         end
25         if cont1<2 then
26             Manter (R(x_l), R(x_{t+1})) em I_S;
27         end
28     end
29     if cont <2 ou cont1<2 e marcador igual a 1 then
30         Q[índice]=I_s, índice=índice+1;
31     end
32 end
```

O grafo $G_u(A, B)$ é representado por uma lista de adjacências. A lista é representada por uma estrutura de dados contendo um arranjo de tamanho 2 para armazenar as arestas cinzas e outro de tamanho 2 para armazenar as arestas pretas. A Figura 4.7 mostra a representação do grafo $G_u(A, B)$ da Figura 2.2.

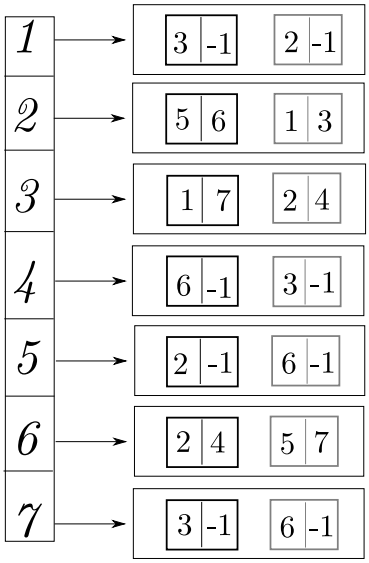

Figura 4.7: Representação do Grafo de Pontos-de-Quebra $G_u(A, B)$.

Um 2-ciclo é formado por exatamente 4 elementos de A e suas respectivas arestas pretas e cinzas envolvidas. Na representação do grafo F_{AB}, cada 2-ciclo em $G_u(A, B)$ representa um vértice, onde cada vértice contém um arranjo de inteiros de tamanho 4 para alocar os índices dos genes em A que constituim o 2-ciclo. Considere o seguinte exemplo com $A = \{X = (1, 2, 3), Y = (4, 6, 5, 7)\}$ e $B = \{X' = (1, 2, 3), Y = (4, 5, 6, 7)\}$ contendo apenas um único 2-ciclo na Figura 4.8. Note que, os genes em X são utilizados para os 1-ciclos e eventualmente possuem sinais atribuídos, sendo assim, são descartados na busca por 2-ciclos, conforme descrito na Seção 4.2.

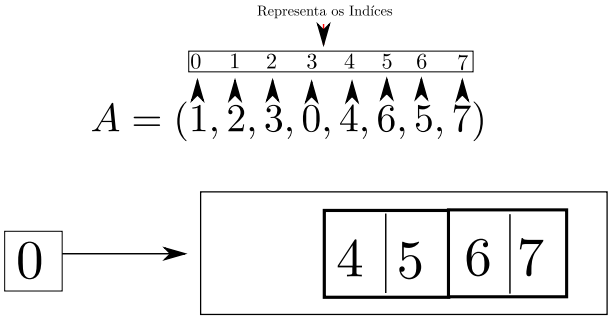

Figura 4.8: Representação do Grafo F_{AB}.

As componentes são representadas por uma lista de ponteiros de estruturas com tamanho igual à quantidade de 2-ciclos. Cada nodo na lista de ponteiros contém um arranjo

de duas posições com o índice do primeiro 2-ciclo e do último 2-ciclo, estes índices são herdados do grafo F_{AB} para representar uma determinada componente. Além disso, um marcador para cada componente é utilizada da seguinte forma: se a componente contém dois ou mais 2-ciclos, então tal componente é dita compartilhada, caso contrário (um único 2-ciclo) é marcada como não compartilhada e eventualmente representa um 2-ciclo isolado. Isto facilita o processo de busca por *S-MSPs*, uma vez que apenas as componentes não compartilhadas representando 2-ciclos isolados serão avaliadas.

Análise da Complexidade de Tempo do Algoritmo de Aproximação

O Algoritmo 2 foi implementado na linguagem `C++`. Será apresentada a análise da complexidade de tempo de execução, mostrando que a implementação tem a mesma complexidade dada em [14].

Na linha 1, o grafo $G_u(A, B)$ é construído usando listas de adjacências em $O(n^2)$.

Na linha 2, o processo de computar todos os 1-ciclos, e na linha 3 construir o grafo F_{AB} toma $O(n)$ cada um, uma vez que é necessário percorrer os n genes em A.

Na linha 4, para computar o máximo emparelhamento M de F_{AB}, foi utilizado a biblioteca `boost`, tal biblioteca possui grande prestígio na comunidade científica e, encontra-se implementada em `C++`, disponível em `http://www.boost.org/`. A computação do máximo emparelhamento tem complexidade de tempo $O(V^2)$ com V representando o número de vértices em F_{AB}.

Na linha 5, encontrar 2-ciclos isolados e 2-ciclos em componentes relacionadas de M, tem complexidade de tempo $O(m^2)$ com m sendo o número de vértices de M, uma vez que é necessário comparar se dois ciclos compartilham a mesma aresta em M.

Na linha 6, o procedimento para identificar as *S-MSPs* (Algoritmo 3) tem complexidade de $O(m.r)$, com $r = \frac{n}{N}$, onde n representa o número de genes e N o número de cromossomos em A.

Na linha 7, o procedimento para criar as *RMSPs* (Algoritmo 5) tem complexidade de $O(mp)$, com p representando a quantidade de *S-MSPs*.

Para as linhas 8 e 9, distribuir apropriados sinais para ambos 2-ciclos isolados e relacionados, o tempo necessário é $O(m^2)$.

Na linha 10, remover todas as *RMSPs* (Algoritmo 4) é executado em $O(m)$, uma vez que no pior caso pode existir $m/2$ *RMSPs* e cada *RMSP* pode ser removida em $O(1)$, visto que o procedimento unicamente reverte os sinais dos genes extremos de cada *RMSP*.

Na linha 11, tomar os sinais distribuídos para os 2-ciclos nos prévios passos e fixá-los para os genes de A tem complexidade de tempo $O(n)$.

Nas linhas 12, 13 o qual consiste nos seguintes procedimentos: verificar se existe genes sem sinais remanescentes em A e, distribuir sinais para tais genes tem complexidade $O(n)$ para cada procedimento respectivamente.

Por fim, na linha 15 são fixados sinais positivos para cada gene em B em tempo $O(n)$ e na linha 16, utilizando o algoritmo proposto em [5] para computar a distância de translocações entre genomas com sinal, o qual é linear em n, computa-se a distância entre $\vec{A}$ e $\vec{B}$.

Assim, se considerar unicamente os procedimentos com maiores pesos computacionais, que neste algoritmo seria o procedimento utilizado para construir o grafo $G_u(A, B)$, o qual tem complexidade quadrática em n, a implementação possui tempo de execução conforme sugerido em ([13] e [14]).

4.5 Apresentado a Prova que as soluções do Algoritmo Aproximado é de raio $1.5 + \varepsilon$

Seja $G_s^A(\vec{A}, \vec{B})$ o grafo de Pontos-de-Quebra construído a partir de $\vec{A}$ e $\vec{B}$ obtidos pelo algoritmo aproximado $1.5 + \varepsilon$. Seja c_i^A o número de *i-ciclos*, s_A o número de *minSPs* e f_A o índices remanescentes em $G_s^A(\vec{A}, \vec{B})$. O número de translocações requeridas para o algoritmo aproximado é denotado como $d_A(\vec{A}, \vec{B})$. Pelo Lema **4.1**,

$$d_A(\vec{A}, \vec{B}) = n - N - \Sigma_{i \geq 1} c_i^A + s_A + f_A. \quad (1)$$

onde $f_A \in \{0, 1, 2\}$. Uma *minSP* é uma *1-2minSP* se os ciclos contidos na *minSP* tem como único ciclo longo um 2-ciclo e os demais ciclos são 1-ciclos. Seja c_i^* o número de i-ciclos em $G_s^*(\vec{A}, \vec{B})$, o qual representa o grafo de Pontos-de-Quebra ótimo ($\vec{A}$ utilizado no $G_s^*(\vec{A}, \vec{B})$ representa a versão de A contendo uma atribuição de sinais que provê a menor distância de translocação). Pelo Lema **3** em [14],

$$d(\vec{A}, \vec{B}) \geq n - N - \sum_{i \geq 1} c_i^* + s^*. \quad (2)$$

Lema 4.3 (Inequação chave para a prova do raio, Lema **4** em [14])**.** *Assuma que $c_1^A = c_1^*$, que é, $G_s^A(\vec{A}, \vec{B})$ tem todos os possíveis 1-ciclos. Se*

$$c_2^A - s_2^A \geq \frac{c_2^* - 3s^*}{2}, \quad (3)$$

então $\frac{d_A(\vec{A}, \vec{B}) - f_A}{d(\vec{A}, \vec{B})} \leq 1.5$.

Demonstração. Seja $\Delta = \frac{3}{2}d(\vec{A},\vec{B}) - d_A(\vec{A},\vec{B}) + f_A$. Se $\Delta \geq 0$, então $\frac{d_A(\vec{A},\vec{B})-f_A}{d(\vec{A},\vec{B})} \leq 1.5$:

$$
\begin{aligned}
&\geq \tfrac{3}{2}\left(n - N - c_1^* - c_2^* - \sum_{i\geq 3} c_i^* + s^*\right) - \left(n - N - c_1^A - \sum_{i\geq 2} c_i^A + s_A + f_A\right) + f_A \qquad (4)\\
&\geq \left(\tfrac{3}{2}n - \tfrac{3}{2}N - \tfrac{3}{2}c_1^* - \tfrac{3}{2}c_2^* - \tfrac{3}{2}\sum_{i\geq 3} c_i^* + \tfrac{3}{2}s^*\right) - \left(n - N - c_1^A - \sum_{i\geq 2} c_i^A + s_A + f_A\right) + f_A\\
&\geq \left(\tfrac{1}{2}n - \tfrac{1}{2}N - \tfrac{1}{2}c_1^* - \tfrac{3}{2}c_2^* - \tfrac{3}{2}\sum_{i\geq 3} c_i^* + \tfrac{3}{2}s^*\right) + \sum_{i\geq 2} c_i^A - s_A\\
&\geq \left(\tfrac{1}{2}n - \tfrac{1}{2}N - \tfrac{1}{2}c_1^* - \tfrac{3}{2}c_2^* - \tfrac{3}{2}\sum_{i\geq 3} c_i^* + \tfrac{3}{2}s^*\right) + \sum_{i\geq 2} c_i^A - s_A\\
&\geq \tfrac{1}{2}\left(n - N - c_1^* - 2c_2^* - \sum_{i\geq 3} 3c_i^*\right) - \tfrac{1}{2}c_2^* + \tfrac{3}{2}s^* + \sum_{i\geq 2} c_i^A - s_A.
\end{aligned}
$$

A primeira inequação é de (2). Note que $n - N$ é o número de arestas pretas em $G_s^*(\vec{A},\vec{B})$. Por definição de c_i^*, tem-se $n - N = \sum_{i\geq 1} i c_i^*$. Assim, (4) torna-se

$$
\begin{aligned}
\Delta &\geq \frac{1}{2}\left(\sum_{i\geq 1} i c_i^* - c_1^* - 2c_2^* - \sum_{i\geq 3} 3c_i^*\right) - \frac{1}{2}c_2^* + \frac{3}{2}s^* + \sum_{i\geq 2} c_i^A - s_A\\
&\geq \frac{1}{2}\sum_{i\geq 4}(i-3)c_i^* - \frac{1}{2}c_2^* + \frac{3}{2}s^* + \sum_{i\geq 2} c_i^A - s_A\\
&\geq -\frac{1}{2}c_2^* + \frac{3}{2}s^* + \sum_{i\geq 2} c_i^A - s_A \qquad \left(\textit{uma vez que, } \sum_{i\geq 4}(i-3)c_i^* \geq 0\right).
\end{aligned}
$$

Para provar $\Delta \geq 0$, precisa-se provar

$$c_2^A + \sum_{i\geq 3} c_i^A - s_A \geq \frac{1}{2}c_2^* - \frac{3}{2}s^* \qquad (5)$$

O número de *minSPs* contendo pelo menos um i-ciclo para $i \geq 3$ é $S_A - S_2^A$. Uma vez que, S_A e S_2^A é o número de *minSPs* e o número de *1-2minSPs* em $G_s^A(\vec{A},\vec{B})$. Assim, $\sum_{i\geq 3} c_i^A \geq s_A - S_2^A$. Substituindo em (5) tem-se,

$$
\begin{aligned}
c_2^A + s_A - s_A - s_2^A &\geq \frac{1}{2}c_2^* - \frac{3}{2}s^*\\
c_2^A - s_2^A &\geq \frac{1}{2}c_2^* - \frac{3}{2}s^*.
\end{aligned}
$$

Implicando

$$\frac{d_A(\vec{A},\vec{B}) - f_A}{d(\vec{A},\vec{B})} \leq 1.5.$$

Logo, o lema é provado. □

Para mostrar que o raio do algoritmo é $1.5+\varepsilon$, mantém-se o foco na inequação (3).

4.5.1 Análise dos 2-ciclos

Por 2-ciclos selecionáveis, entende-se que foram atribuídos sinais para seus genes de modo a representar esses 2-ciclos (Passo 5 do algoritmo $1.5+\varepsilon$), e os ciclos arbitrários são os mesmos construídos no passo 6 do algoritmo $1.5+\varepsilon$, atribuindo sinais para os genes que ainda se encontram sem sinais.

Assuma que o máximo emparelhamento M de F_{AB} contém z 2-ciclos isolados. Uma componente relacionada de M é uma componente ímpar (par) se contém um número ímpar (par) de 2-ciclos. Seja r_0 denotando o número de componentes relacionadas ímpares, Δr_0 o número de componentes relacionadas ímpares onde cada uma tenha no mínimo um 2-ciclo conflitante que não esteja no grafo $G_s^5(\vec{A},\vec{B})$ (o grafo que ainda não teve as *RS-MSPs* destruídas), e Δr_e o número de componentes relacionadas pares onde cada uma tenha dois 2-ciclos que não esteja em $G_s^5(\vec{A},\vec{B})$. Seja Δz denotando o número de 2-ciclos isolados conflitantes (2-ciclos tal que ao remover uma *RS-MSP* eliminam-se outros 2-ciclos) que não estão em $G_s^5(\vec{A},\vec{B})$, e por fim c_2^a o número de 2-ciclos arbitrários em $G_s^5(\vec{A},\vec{B})$.

O grafo $G_s^A(\vec{A},\vec{B})$ usado no lema abaixo, é o grafo após remover as *RS-MSPs*.

Lema 4.4 (Quantidade de 2-ciclos na solução dada pelo algoritmo aproximado, Lema 9 em [14])**.** *O número total de 2-ciclos em* $G_s^A(\vec{A},\vec{B})$ *é*

$$c_2^A = \frac{|M| + r_0 + z}{2} - \Delta r_0 - \Delta r_e - \Delta z + c_2^a.$$

Demonstração. Pelo Lema **4.2**, tem-se que o número de 2-ciclos em ambos $G_s^5(\vec{A},\vec{B})$ e $G_s^A(\vec{A},\vec{B})$ é o mesmo. Para uma componente ímpar contendo pelo menos um 2-ciclo conflitante que não esteja em $G_s^5(\vec{A},\vec{B})$, o número de 2-ciclos selecionados é reduzido por um. Uma vez que, se tem o 2-ciclo eliminado na componente e alternância nas escolhas dos 2-ciclos restantes. Assim, o número total de 2-ciclos selecionáveis reduzidos será Δr_0.
Para uma componente par, contendo dois 2-ciclos conflitantes, o número de 2-ciclos selecionáveis é reduzido por um. Consequentemente, o número total de 2-ciclos selecionáveis

reduzidos será Δr_e. Há também Δz 2-ciclos conflitantes, que não estão em $G_s^5(\vec{A}, \vec{B})$. Portanto, o total de reduções de 2-ciclos selecionáveis é $\Delta r_0 + \Delta r_e + \Delta z$.

Se $\Delta r_0 + \Delta r_e + \Delta z = 0$, não existem 2-ciclos conflitantes com os z 2-ciclos isolados, e assim podem-se manter todos os z 2-ciclos na solução e

$$\frac{|M| + r_0 - z}{2}$$

2-ciclos relacionados. O total de 2-ciclos selecionáveis na solução será

$$\frac{|M| + r_0 - z}{2} + z = \frac{|M| + r_0 + z}{2}.$$

Somando com os c_2^a 2-ciclos arbitrários, $G_s^5(\vec{A}, \vec{B})$ contém

$$\frac{|M| + r_0 + z}{2} - \Delta r_0 - \Delta r_e - \Delta z + c_2^a.$$

□

4.5.2 Análise das 1-2minSPs

Uma *minSP* é uma isolada *minSP 1-2* se os ciclos contidos na *minSP* são compostos por 2-ciclos isolados e 1-ciclos. Seja s_m o número de isoladas *minSPs* do tipo *1-2* com no mínimo dois ciclos isolados e s_s o número de *minSPs* isoladas *1-2* contendo exatamente um 2-ciclo isolado em $G_s^5(\vec{A}, \vec{B})$. Por definição, todas as *S-MSPs* (definidas na Seção 4.3) em $G_s^5(\vec{A}, \vec{B})$ são isoladas *minSPs* do tipo *1-2*, contendo exatamente um 2-ciclo, e o número total de *S-MSPs* em $G_s^5(\vec{A}, \vec{B})$ também será s_s. Assim, o número total de isoladas *minSPs 1-2* é

$$s_m + s_s.$$

Seja s_a o número de *1-2minSPs* em $G_s^5(\vec{A}, \vec{B})$, onde cada uma delas contém pelo menos um 2-ciclo arbitrário.

Proposição 4.1(Relação entre 2-ciclos relacionados e ciclos arbitrários Prop. 1 em [14]). *Se uma minSP de $G_s^5(\vec{A}, \vec{B})$ contém um 2-ciclo relacionado, então a minSP contém no mínimo um ciclo arbitrário.*

Demonstração. Suponha uma *minSP* contendo um 2-ciclo relacionado selecionado C em uma componente U de $G_u^A(A, B)$, e U contendo 2-ciclos $c_l, c_{l+1}, \cdots, c_r$, onde c_i se relaciona com c_{i+1} para $l \leq i \leq r-1$. C pode ser unicamente uma componente relacionada do tipo (a) ou (b) na figura 4.4. Assuma que c_l contém o vértice mais à esquerda de U.

U é do tipo (a).
Se $c = c_i, l \leq i \leq r-1$, então cada aresta preta de c_{i+1} está em um ciclo arbitrário de I. Se $c = c_r$, então cada aresta preta de c_{r-1} pertence a um ciclo arbitrário. Uma vez que, existe uma aresta cinza conectando os vértices mais à esquerda e mais à direita de c_r, pelo menos uma aresta preta e de c_{r-1} (que está em um ciclo arbitrário) está em I. Uma vez que, I é uma *minSP*, por definição de subpermutação, I contém todo o ciclo em que a aresta preta e está contida.

U é do tipo (b).
Se $c = c_i, l \leq i \leq r-1$, então a aresta preta de c_{i+1} entre as duas arestas pretas de c_i está em um ciclo arbitrário de I. Se $c = c_r$, a aresta preta de C_{r-1} que está entre as duas arestas pretas de C_r está em um ciclo arbitrário de I. □

A partir da Proposição **1**, tem-se o seguinte Lema:

Lema 4.5 (Quantidade de *1-2minSPs* até o passo 6 do algoritmo aproximado, Lema 10 em [14])**.** *O número total de 1-2minSPs em* $G_s^5(\vec{A}, \vec{B})$ *é*

$$s_m + s_s + s_a.$$

Considere as s_s *S-MSPs* em $G_s^5(\vec{A}, \vec{B})$. Tais *S-MSPs* podem ser classificadas em três grupos:

1. Existem s_n *S-MSPs* que não contém aresta cinza chave. Estes tipos de *S-MSPs* não correspondem a qualquer *CRS-MSPs* no grafo $G_u(A, B)$.

2. Existem s_t *S-MSPs* em $G_s^5(\vec{A}, \vec{B})$ que correspondem a *CRS-MSPs* no grafo $G_u(A, B)$ e não são transformadas em *RS-MSPs* pelo procedimento apresentado no Algoritmo 3.

3. Existem s_r *S-MSPs* $G_s^5(\vec{A}, \vec{B})$ que correspondem a *CRS-MSPs* no grafo $G_u(A, B)$ e são transformadas em *RS-MSPs*. Estas *RS-MSPs* são removidas no passo 7 do algoritmo.

Assim, tem-se

$$s_s = s_n + s_t + s_r. \tag{6}$$

Seja s_r o número de *RS-MSPs* no $G_s^5(\vec{A}, \vec{B})$, onde se tem s_r^3 *RS-MSPs* com i-ciclos removíveis para $i \geq 3$ (*RS-MSPs* contendo unicamente uma das arestas cinzas chave).

Removendo as s_r^3 *RS-MSPs* não se criam quaisquer outras *1-2minSPs*. Logo, existe s_r^3 *1-2minSPs* em $G_s^5(\vec{A},\vec{B})$ que não estarão em $G_s^A(\vec{A},\vec{B})$(após o passo 7 do algoritmo). Assim, tem-se

$$s_2^A = s_m + s_s + s_a - s_r^3. \tag{7}$$

Considere as *R-SMSPs* em $G_s^5(\vec{A},\vec{B})$ que são removidas no passo 7 do algoritmo, tais ciclos removíveis são do tipo 2-ciclo. Pelo Lema 8 em [14], os 2-ciclos removíveis podem ser unicamente 2-ciclos isolados ou 2-ciclos arbitrários. Seja s_r^i o número de *RS-MSPs* com 2-ciclos removíveis isolados e s_r^a o número de *RS-MSPs* com removíveis 2-ciclos arbitrários. Tem-se

$$s_r = s_r^3 + s_r^a + s_r^i. \tag{8}$$

Seja z_{in} o número de 2-ciclos isolados internos a *minSPs* em $G_s^5(\vec{A},\vec{B})$ e z_{out} o número de 2-ciclos isolados externos a qualquer *minSPs* em $G_s^5(\vec{A},\vec{B})$. Por definição, o total de 2-ciclos isolados z é $z = \Delta z + z_{in} + z_{out}$.

Pelo lema 11 em [14] tem-se

$$z_{out} \geq s_t + s_r^i.$$

Onde s_t denota a quantidade de *CS-MSPs* em $G_s^5(\vec{A},\vec{B})$ que não são transformadas em *RS-MSPs.* Pela definição de s_m e s_s tem-se:

$$z_{in} \geq 2s_m + s_s \tag{9}$$

A partir das equações (6), (8) e (9) tem-se:

$$\begin{aligned}
z &= \Delta z + z_{in} + z_{out} \\
z &\geq s_t + s_r^i + z_{in} + \Delta z && \textit{substituindo } z_{out} \\
z &\geq s_s - s_n - s_r + s_r^i + z_{in} + \Delta z && \textit{por } (6) \\
z &\geq s_s - s_n - s_r + s_r - s_r^3 - s_r^a + z_{in} + \Delta z && \textit{por } (8) \\
z &\geq s_s - s_n - s_r^3 - s_r^a + 2s_m + s_s + \Delta z && \textit{por} (9) \\
z &\geq 2s_m + 2s_s - s_n - s_r^3 - s_r^a + \Delta z && (10)
\end{aligned}$$

Lema 4.6 (Quantidade de *RS-MSPs* com i-ciclos removíveis para $i \geq 3$, Lema 12 em [14])**.**

$$s_r^3 \geq \Delta r_0 + 2\Delta r_e + \Delta z - s_r^a.$$

Demonstração. Uma vez que, M contém Δr_0 componentes relacionadas ímpares cada uma com pelo menos um 2-ciclo conflitante, Δr_e componentes relacionadas pares cada uma com dois 2-ciclos conflitantes e Δz ciclos isolados conflitantes. Há no mínimo

$$\Delta r_0 + 2\Delta r_e + \Delta z$$

2-ciclos destruídos pelo procedimento de criação das *RS-MSPs*, e consequentemente $\Delta r_0 + 2\Delta r_e + \Delta z$ *RS-MSPs* em $G_s^5(\vec{A}, \vec{B})$. Pelo Lema 8 em [14], o número de *RS-MSPs* destruindo 2-ciclos conflitantes não é maior que $s_r^3 + s_r^a$. Assim,

$$s_r^3 + s_r^a \geq \Delta r_0 + 2\Delta r_e + \Delta z.$$

□

O seguinte lema faz a conexão entre os parâmetros c_2^* e s^* (número de 2-ciclos e *minSPs* respectivamente no grafo ótimo $G_s^*(\vec{A}, \vec{B})$) e os parâmetros $|M|$ e s_n (tamanho do máximo emparelhamento e número de *S-MSPs* que não contém arestas cinzas chaves) no algoritmo aproximado.

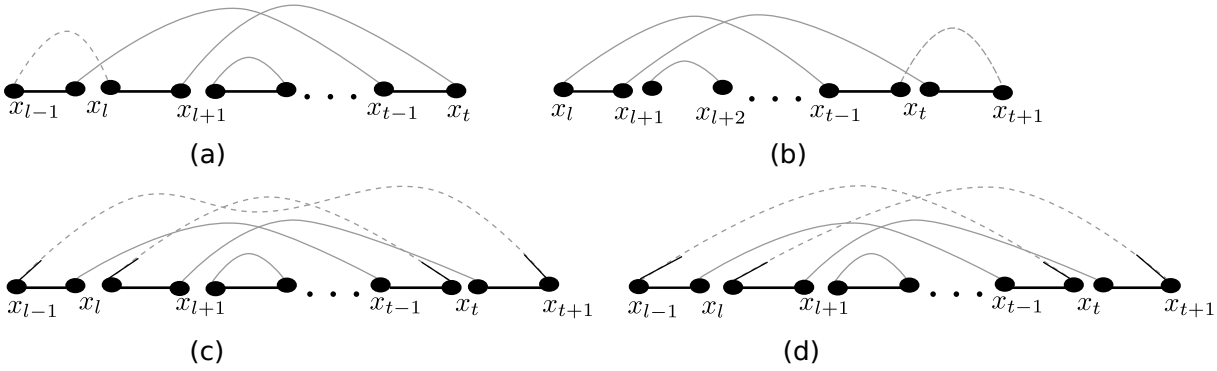

Figura 4.9: Cada uma das arestas pretas $(r(x_i), l(x_{i+1}))$ e $(r(x_{j-1}), l(x_j))$ está em um *i-ciclo* $(i \geq 3)$.

Lema 4.7 (Quantidade de subpermutações na solução aproximada é maior que em uma solução ótima, Lema 13 em [14])**.** $|M| - c_2^* \geq s_n - s^*$.

Demonstração. Seja $G_s^*(\vec{A}, \vec{B})$ o grafo de Pontos-de-Quebra ótimo contendo todos os possíveis 1-ciclos, onde $\vec{A}$ é versão com sinal de A contendo a menor distância de translocação possível. Pela construção de F_{AB}, tem-se $|M| - c_2^* \geq 0$. Assim, tem-se que considerar unicamente o caso, onde $s_n - s^* \geq 0$.

Pelas definições de s_n e s^*, $G_s^A(\vec{A},\vec{B})$ contém no mínimo $(s_n - s^*)$ *S-MSPs* que representam subpermutações não contidas em $G_s^*(\vec{A},\vec{B})$, tais que essas *S-MSPs* não produzem quaisquer *RS-MSPs*.
Seja $I_s = x_l, x_{l+1}, \cdots, x_{t-1}, x_t$ uma das $(s_n - s^*)$ *S-MSPs*, I_s não corresponde a qualquer *S-MSP* em $G_s^*(\vec{A},\vec{B})$. Porém, $G_s^*(\vec{A},\vec{B})$ mantém todos os 1-ciclos entre as arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$, além de conter pelo menos uma das arestas cinzas $(L(x_l), R(x_{t-1}))$ e $(L(x_{l+1}), R(x_t))$. Assim, há três possíveis casos a serem analisados.

Caso 1: Figura 4.9(a), aresta cinza $(L(x_l), R(x_{t-1}))$ está em $G_s^*(\vec{A},\vec{B})$, porém não se tem a aresta $(L(x_{l+1}), R(x_t))$ em $G_s^*(\vec{A},\vec{B})$. Todos os 1-ciclos entre as arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$ são mantidos, e tem-se a aresta cinza $(L(x_{l+1}), L(x_t))$ em $G_s^*(\vec{A},\vec{B})$. Uma vez que, a aresta cinza $(R(x_l), R(x_{t-1}))$ não está em $G_s^*(\vec{A},\vec{B})$, as duas arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$ pertencem a um i-ciclo, com $i \geq 3$.

Caso 2: Figura 4.9(b), aresta cinza $(L(x_{l+1}), R(x_t))$ está em $G_s^*(\vec{A},\vec{B})$, porém $(L(x_l), L(x_{t-1}))$ não está em $G_s^*(\vec{A},\vec{B})$. Uma vez que, todos os 1-ciclos entre as duas arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$ estão mantidos, a aresta cinza $(R(x_l), R(x_{t-1}))$ deve estar em $G_s^*(\vec{A},\vec{B})$. Uma vez que, a aresta cinza $(L(x_{l+1}), L(x_t))$ não está $G_s^*(\vec{A},\vec{B})$, as duas aresta pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$ devem estar em um i-ciclo com $i \geq 3$.

Caso 3: Figuras 4.9(c) e 4.9(d). Tem-se que ambas as arestas cinzas $(L(x_l), R(x_{t-1}))$ e $(L(x_{l+1}), R(x_t))$ estão em $G_s^*(\vec{A},\vec{B})$. Neste caso, tem-se que as arestas pretas $(R(x_{l-1}), L(x_l))$ e $(R(x_{t-1}), L(x_t))$ de um ciclo C em $G_s^A(\vec{A},\vec{B})$ e arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_t), L(x_{t+1}))$ de um ciclo C' em $G_s^A(\vec{A},\vec{B})$ estão mescladas em um único ciclo em $G_s^*(\vec{A},\vec{B})$, Figura 4.9(c), ou em dois diferentes ciclos (Figura 4.9(d)) em $G_s^*(\vec{A},\vec{B})$.
Uma vez que, as arestas cinzas (x_{l-1}, x_t) e (x_l, x_{t+1}) não estão em $G_u(A,B)$, tem-se que, $G_s^*(\vec{A},\vec{B})$ não contém arestas cinzas $(R(x_{l-1}), L(x_t))$ e $(R(x_l), L(xt+1))$, e portanto, tanto C e C' são i-ciclos com $i \geq 3$. Assim, as duas arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$ devem estar em um ou dois i-ciclos, com $i \geq 3$.

Alterando os sinais de pelo menos um dos dois genes x_l e x_t em $G_s^*(\vec{A},\vec{B})$ para obter uma *S-MSP*, destrói-se o ciclo onde se encontra pelo menos uma das arestas pretas $(R(x_l), L(x_{l+1}))$ e $(R(x_{t-1}), L(x_t))$. Isto incrementa em um o número de 2-ciclos em $G_s^*(\vec{A},\vec{B})$. Repetindo o procedimento, podem-se obter todos os $(s_n - s^*)$ 2-ciclos que representam as *S-MSPs*. Portanto, tem-se

$$|M| \geq c_2^* + s_n - s^*,$$

e o lema se conclui.

□

4.5.3 O Raio Aproximado

Para mostrar que o algoritmo possui soluções com raio de aproximação $1.5+\varepsilon$, faz-se uma análise para dois possíveis cenários. Caso $f(\vec{A}) = 0$ ($\vec{A}$ foi produzido pelo algoritmo aproximado), ou seja, o número de *minSPs*, não é ímpar e nem uma isolação par. Assim, o raio aproximado será 1.5, caso contrário ($f(\vec{A}) \neq 0$) tem-se o raio aproximado de $1.5+\varepsilon$.

Teorema 4.2 (Os índices remanescentes são irrelevantes, Teorema 2 em [14])**.** *Se* $f(\vec{A}) = 0$*, então o raio do algoritmo aproximado é* 1.5.

Demonstração. Seja c_1^A o número de 1-ciclos no grafo $G_s^A(\vec{A}, \vec{B})$ construído a partir de $\vec{A}$ e $\vec{B}$ produzido pelo algoritmo aproximado. Seja c_1^* o número de 1-ciclos no grafo de Pontos-de-Quebra ótimo $G_s^*(\vec{A}, \vec{B})$. Pelo passo 2 do algoritmo, $c_1^A = c_1^*$. Pelo Lema **4.3**, precisa-se unicamente provar que

$$c_2^A - s_2^A \geq \frac{c_2^* - 3s^*}{2}.$$

Pelo Lema **4.4** tem-se,

$$c_2^A = \frac{|M| + r_0 + z}{2} - \Delta r_0 - \Delta r_e - \Delta z + c_2^a.$$

Pela equação 7 tem-se

$$s_2^A = s_m + s_s + s_a - s_r^3.$$

Assim,

$$c_2^A - s_2^A = \frac{|M| + r_0 + z}{2} - \Delta r_0 - \Delta r_e - \Delta z + c_2^a - s_m + s_s + s_a - s_r^3. \tag{11}$$

Substituindo z em (11), com o lado direito da inequação em (9), tem-se

$$\begin{aligned} c_2^A - s_2^A &\geq \tfrac{|M|+r_0+(2s_m+2s_s-s_n-s_r^3-s_r^a+\Delta z)}{2} - (\Delta r_0 + \Delta r_e + \Delta z) + c_2^a - s_m + s_s + s_a - s_r^3 \\ &= \tfrac{|M|+r_0+\Delta z-s_n-s_r^a+s_r^3}{2} - (\Delta r_0 + \Delta r_e + \Delta z) + c_2^a - s_a. \end{aligned} \tag{12}$$

Pelo Lema **4.6**, ($s_r^3 \geq \Delta r_0 + 2\Delta r_e + \Delta z - s_r^a$), substituindo s_r^3 em (12), tem-se

$$\begin{aligned} c_2^A - s_2^A &\geq \tfrac{|M|-s_n+r_0+\Delta z+(\Delta r_0+2\Delta r_e+\Delta z)-2s_r^a}{2} - (\Delta r_0 + \Delta r_e + \Delta z) + c_2^a - s_a \\ &= \tfrac{|M|-s_n+r_0-2s_r^a}{2} - \tfrac{\Delta r_0}{2} + c_2^a - s_a \\ &= \tfrac{|M|-s_n}{2} + \tfrac{r_0-\Delta r_0}{2} + c_2^a - s_a - s_r^a \\ &\geq \tfrac{|M|-s_n}{2} + c_2^a - s_a - s_r^a. \end{aligned} \tag{13}$$

A última inequação vem do fato que $r_0 \geq \Delta r_0$. Uma vez que, $G_s^5(\vec{A}, \vec{B})$ tem pelo menos s_a arbitrários 2-ciclos em *minSPs* e s_r^a 2-ciclos arbitrários como ciclos removíveis das *RS-MSPs*. Logo, o número de 2-ciclos arbitrários c_2^a é limitado inferiormente como

$$c_2^a \geq s_r^a + s_a.$$

Substituindo c_2^a em (13) temos,

$$c_2^A - s_2^A \geq \frac{|M| - s_n}{2}.$$

Pelo Lema **4.7**, tem-se,

$$c_2^A - s_2^A \geq \frac{c_2^* - s^*}{2}.$$

□

Teorema 4.3 (Os índices remanescentes são relevantes, Teorema 3 em [14])**.** *Se* $f(\vec{A}) \neq 0$*, então o raio do algoritmo aproximado é* $1.5 + \varepsilon$*.*

Demonstração. Seja $d(A, B)$, a distância de translocação entre genomas A e B.
Para qualquer $\varepsilon \geq 0$, faz-se $k = \dfrac{2}{\varepsilon}$. Suponha que $d(A, B) \leq k$. Assim, o número de 1-ciclos em $G_u(A, B)$ não é menor que $n - N - 2k$, com n e N representando a quantidade de genes e cromossomos em A, respectivamente. Logo, pode-se encontrar uma solução ótima ([14]). Se o número de 1-ciclos em $G_u(A, B)$ é menor que $n - N - 2k$, tem-se $d(A, B) = t$, onde $t > k$.
Neste caso, computa-se uma solução aproximada com distância x. Pelo Teorema 4.2, tem-se que

$$x \leq 1.5t + f_A \leq 1.5t + 2,$$

onde f_A representa o índice remanescente de $G_s^A(\vec{A}, \vec{B})$ (grafo construído através dos genomas $\vec{A}$ e $\vec{B}$ produzido pelo algoritmo aproximado), e $(1.5t + 2)$, vem do fato que $f_A \in \{0, 1, 2\}$. Assim, a performance do raio do algoritmo é

$$
\begin{aligned}
\frac{x}{t} &\leq \frac{1.5t+2}{t} \\
&\leq 1.5 + \frac{2}{t} \\
&\leq 1.5 + \frac{2}{k} && t > k \\
&= 1.5 + \frac{2}{\frac{2}{\varepsilon}} && k = \frac{2}{\varepsilon} \\
&= 1.5 + \varepsilon.
\end{aligned}
$$

□

Capítulo 5

Algoritmo Genético para DTC

O capítulo possui a seguinte organização. A Seção 5.1 apresenta a descrição, o pseudo-código e a análise de complexidade de tempo do algoritmo genético (AG) proposto para o problema de DTS. Por fim, a Seção 5.2 apresenta respectivamente o pseudo-código e o arcabouço teórico do algoritmo utilizado como função de aptidão do AG proposto. O algoritmo foi publicado em [42].

5.1 Descrição do AG Proposto

Antes de enunciar o AG proposto, é apresentada a ideia chave utilizada pelo mesmo. Na busca por soluções de boa qualidade para o problema DTC, a seguinte estratégia é utilizada: dado um genoma sem sinal A (uma vez que B é considerado como o genoma identidade), construir todas as possíveis versões com sinal de tal genoma, atribuindo um sinal aleatoriamente positivo/negativo para cada gene em A. A menor distância de translocação dentre todas estas versões com sinal representará a solução ótima para a distância de translocação entre os genomas A e B sem sinal.

Note que, ao obter todas as versões com sinal de um dado genoma sem sinal, serão obtidos 2^n genomas com sinal, uma vez que para cada gene é possível tanto atribuir um sinal negativo ou positivo. A partir deste fato, surgiu a ideia de utilizar algoritmos genéticos para explorar o espaço de soluções. Todavia, nem sempre o algoritmo genético é capaz de encontrar uma solução ótima em tempo polinomial, visto que o espaço de busca é muito vasto, porém, espera-se que em um tempo aceitável, o proposto algoritmo genético forneça soluções próximas do ótimo, superando as providas pelo algoritmo aproximado $1.5 + \varepsilon$.

Sejam A e B genomas sem sinal satisfazendo a Propriedade 1 definida na Seção 2.1.4 do Capítulo 2, onde B é o genoma identidade. O AG proposto trabalha como segue: Inicialmente é construída uma população de tamanho $n \log n$, onde cada indivíduo repre-

senta um diferente genoma com sinal gerado a partir de A. Após isto, para cada geração, a reprodução é executada como segue: Selecionam-se dois indivíduos da população, tais indivíduos fazem parte dos melhores indivíduos (representam melhores soluções) para o qual o operadores de cruzamento e mutação serão aplicados a fim de produzir dois novos indivíduos. Então, os novos indivíduos são incorporados na população. Neste passo, escolhe-se o pior par de indivíduos na população para dar lugar a estes dois novos indivíduos. O AG finaliza após um número de gerações terem sido completadas, que é pré fixado como sendo n.

O pseudo-código do AG proposto é mostrado no Algoritmo 6.

Algoritmo 6: AG para computar DTS

Input: Genoma sem sinal A
Output: Número de translocações para transformar A numa identidade

```
1 Gerar a população inicial de A;
2 Computar a aptidão da população;
3 for i = 1 até Tamanho(A) do
4 |   Selecionar os melhores indivíduos encontrados;
5 |   Aplicar o operador de cruzamento;
6 |   Aplicar o operador de mutação sobre os descendentes ;
7 |   Computar a aptidão dos descendentes;
8 |   Substituir os piores indivíduos;
9 end
```

5.1.1 Análise da Complexidade do AG

O tamanho da população inicial é definida como sendo $n \log n$. Cada indivíduo da população é gerado a partir de A em tempo linear, aleatoriamente atribuindo um sinal positivo ou negativo para cada gene. Este passo tem complexidade de $O(n^2 \log n)$.

Uma vez que, para um único indivíduo a aptidão é computada em tempo linear, o processo de computar a aptidão para todos os indivíduos contidos na população tem complexidade de tempo $O(n^2 \log n)$.

No passo de seleção de indivíduos, para ordenar a população pelos seus respectivos valores de aptidão em ordem ascendente, usa-se o algoritmo *counting sort* que executa com complexidade $O(n + n \log n)$, com o valor de aptidão de cada indivíduo no intervalo de 1 até n, e com uma população de tamanho $n \log n$. Assim, a complexidade deste passo é $O(n \log n)$.

No passo de cruzamento, os melhores indivíduos classificados durante o passo se seleção são escolhidos para serem os pais. Para cada par de pais, aplica-se cruzamento entre eles, copiando os elementos a partir de um ponto aleatório à direita de um indivíduo para o

outro indivíduo e vice-versa, claramente tal passo pode ser computado em tempo linear ($O(n)$). Assim, o tempo de execução para a operação de cruzamento sobre um quantidade máxima de $n \log n$ indivíduos é $O(n^2 \log n)$.

No passo de mutação, tal operador é aplicado para cada descendente produzido pelo operador de cruzamento. Para cada gene de um dado indivíduo, uma checagem é realizada com o intuito de verificar quando aplicar ou não a mutação sobre estes genes, isto claramente toma tempo linear em n. Logo, o tempo total para a operação de mutação considerando toda a população ($n \log n$) é $O(n^2 \log n)$.

No passo de substituição, cada substituição de um indivíduo na população é realizado em $O(n)$, uma vez que necessita copiar todos os genes. Ao considerar toda a população, o tempo total requerido para essa operação tem complexidade de $O(n^2 \log n)$.

O algoritmo finaliza após n gerações, o tempo total é limitado em $O(n^3 \log n)$, ou de forma mais geral $O(gn^2 \log n)$, onde g representa o número de genes.

5.2 Função de Aptidão utilizada no AG

A função de aptidão utilizada no AG proposto, consiste da distância de translocação entre $\vec{A}$ e $\vec{B}$, onde $\vec{A}$ representa um indivíduo na população e $\vec{B}$ a versão de B contendo todos genes com sinal positivo.

Para computar a distância entre $\vec{A}$ e $\vec{B}$, o AG faz uso o algoritmo proposto por Bergeron et al em [5]. A seguir será apresentado o conteúdo teórico envolvido na implementação de tal algoritmo.

5.2.1 Intervalos Elementares e Ciclos

Seja $\vec{A}$ um genoma contendo n genes e N cromossomos. Considera-se P_A a permutação estendida com sinal definida por uma concatenação arbitrária de cromossomos de $\vec{A}$.

Definição 5.1 (Ponto-de-quebra [5])**.** *Um par $p \cdot q$ de elementos consecutivos em uma permutação com sinal é chamado um ponto. Um ponto é chamado uma adjacência, se é um ponto da forma $i.i+1$ ou $-(i+1).-i$ para $0 \leq i \leq n$, caso contrário, é chamado um ponto-de-quebra.*

Uma permutação P_A tem $n+1$ pontos, $N-1$ deles estão entre caudas, e dois outros pontos estão entre 0 e uma cauda e entre uma cauda e o valor representado por $n+1$. Os $N+1$ pontos definem a concatenação do genoma $\vec{A}$ e são chamados pontos brancos. Os outros pontos são chamados pontos pretos, sendo internos aos cromossomos. Por exemplo,

considere o genoma

$$\vec{A} = \{(+1, +3, +9), (+7, +8, +4, +5, +6), (+10, +2, +11, +12, +13)\},$$

A Figura 5.1 mostra como seria P_A.

$$P_A = (0 + 1 + 3 + 9 + 7 + 8 + 4 + 5 + 6 + 10 + 2 + 11 + 12 + 13 + 14)$$

Figura 5.1: Permutação P_A construída a partir do genoma $\vec{A}$.

P_A tem 14 pontos na figura 5.1; os pontos brancos internos a P_A marcam a concatenação de um cromossomo pertencente a $\vec{A}$.

Definição 5. 2 (Intervalo elementar [5]). *Para cada par de genes sem sinal $(k, k+1)$ para $0 \leq k \leq n+1$, defini-se o intervalo elementar I_K associado ao par $k.k+1$ como segue:*

1. *Ponto à direita de k, se k é positivo, caso contrário, a esquerda de k.*

2. *Ponto à esquerda de $k+1$, se $k+1$ é positivo, caso contrário, a direita de $k+1$.*

Definição 5.3 (Ciclos pretos em P_A [5]). *Um ciclo preto (ou branco) é uma sequência de pontos-de-quebra que estão ligados por intervalos elementares pretos (respectivamente, brancos). Um gene k tendo como adjacente $k+1$ (exemplo:+2 +3 ou -3 -2) define um ciclo trivial.*

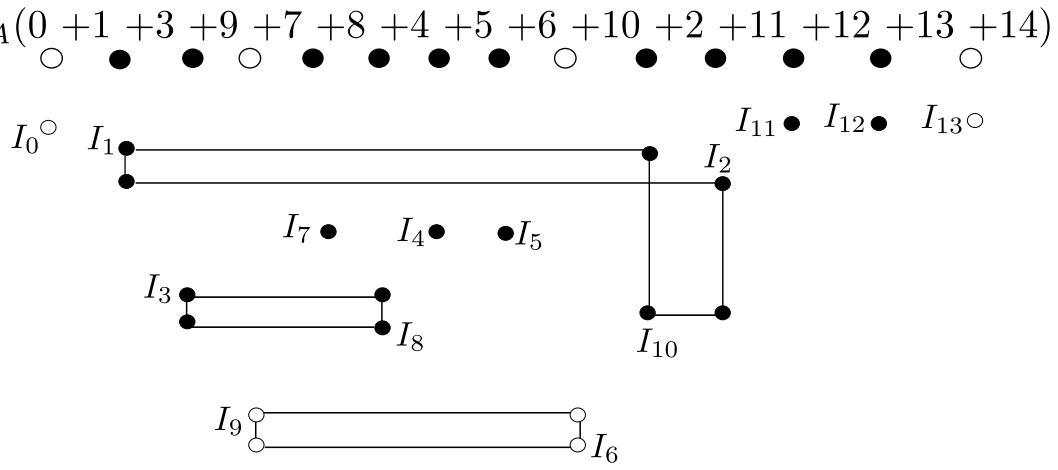

Figura 5.2: Intervalos elementares e ciclos da permutação P_A. Adjacências definem ciclos triviais.

Os intervalos elementares e ciclos de P_A são mostrados na Figura 5.2. Os ciclos brancos formados pelos $N+1$ pontos brancos dependem de como é feita a concatenação dos

cromossomos. Uma vez que, a ordem e a orientação dos cromossomos são irrelevantes para o problema de distância por translocação, o foco está nos ciclos pretos que são formados pelos $n - N$ pontos pretos (na figura 5.2 são os ciclos contendo vértices pretos). Pode-se observar que o número de ciclos pretos em P_A é máximo, e igual a $n - N$, unicamente se o genoma $\vec{A}$ está ordenado (P_A está ordenado em ordem incremental).

Lema 5.1 (Genoma após uma translocação [5])**.** *Uma translocação em um genoma $\vec{A}$ modifica o número de ciclos pretos no genoma resultante por 1, 0 ou -1.*

Demonstração. Sejam W e Z dois cromossomos em $\vec{A}$, particionando W em W_1 e W_2 e Z em Z_1 e Z_2, há três possíveis cenários.

O número de ciclos pretos é incrementado: Considere o caso (a) da figura 5.3, ao aplicar uma translocação produz-se $\vec{A}'$. Uma vez que, se tinha um ciclo em $\vec{A}$ considerando os 4 genes envolvidos na operação, $\vec{A}'$ terá um ciclo a mais no somatório de ciclos pretos, e será denominada uma translocação adequada.
Observe que esse será o melhor caso, pois em uma translocação só é possível juntar dois pares de genes.

O número de ciclos pretos não se altera: Considere o caso (b) da figura 5.3, ao aplicar uma translocação produz-se $\vec{A}'$. $\vec{A}'$ terá a mesma quantidade de ciclos pretos. Logo, essa translocação será inadequada.

O número de ciclos pretos é decrementado: Considere o caso (c) da figura 5.3, ao aplicar uma translocação produz-se o caso $\vec{A}'$. $\vec{A}'$ tem um ciclo a menos no somatório de ciclos pretos. Assim, essa translocação será denominada ruim (aplicou uma translocação entre pares adjacentes).

Com os três casos acima, tem-se as possíveis variações de quantidade de ciclos pretos após uma translocação entre segmentos de dois cromossomos, onde, uma translocação apropriada aumenta em um a quantidade de ciclos pretos, uma translocação inadequada mantém a quantidade de ciclos pretos e uma translocação ruim decrementa o número de ciclos pretos. □

Um intervalo elementar cujos pontos finais pertencem a diferentes cromossomos é chamado *intercromossomal*; caso contrário, é chamado *intracromossomal*. Dado um intervalo elementar *intercromossomal* I_K de P_A, pode-se assumir que elementos k e $k + 1$ podem criar uma nova adjacência em P_A e consequentemente em $\vec{A}$.
Assim, tem-se o seguinte lema.

Lema 5.2 ([5]). *Para cada intervalo intercromossomal em P_A existe uma translocação adequada no genoma $\vec{A}$.*

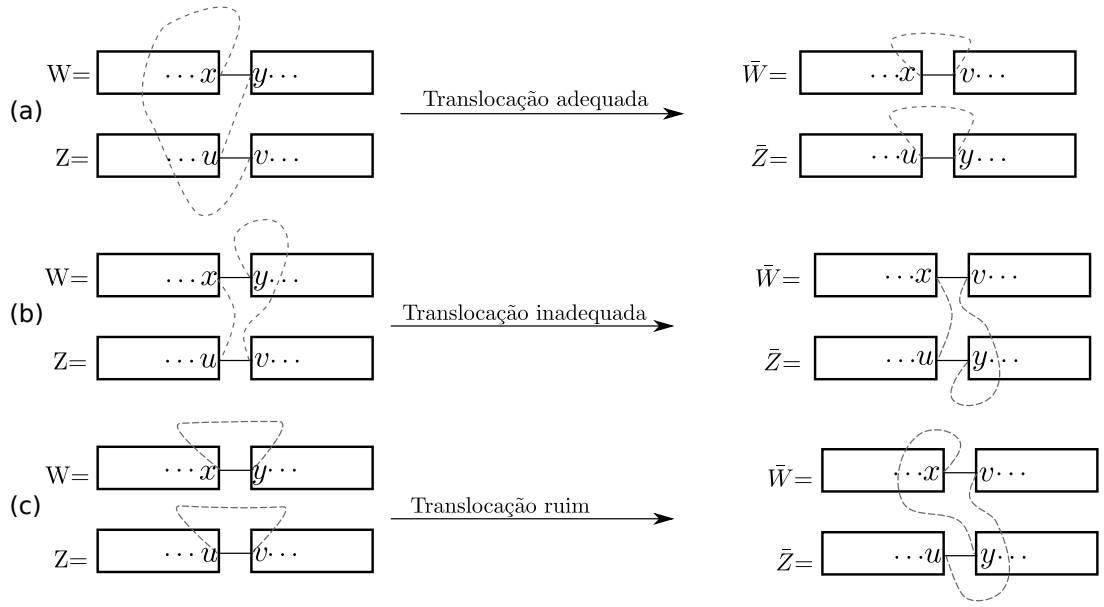

Figura 5.3: Cenário de translocações utilizando operação *Prefixo-Prefixo* sobre arestas pretas (u, v) e (x, y).

5.2.2 Componentes

Componentes foram primeiramente identificadas em [23] como subpermutações (SP). Para recordar, retorne ao Capítulo 2.

Definição 5.4 (Componentes em permutações estendidas [5]). *Uma componente de uma permutação estendida P_A é um intervalo de i até $i + j$ ou de $-(i + j)$ até $-i$, onde $j > 0$, tal que o conjunto de genes considerados sem sinal no intervalo $\{i, \cdots, i + j\}$ ou $\{-(i + j), \cdots, -i\}$ não contém qualquer outro intervalo menor.*

Quando os elementos de uma componente pertencem a um mesmo cromossomo, esta componente é dita *intracromossomal*. Uma componente *intracromossomal* é dita mínima se não contém qualquer outra componente *intracromossomal*. Uma componente formada por um par de elementos adjacentes (exemplo [2, 3] ou [-3, -2]) é denominada uma componente trivial, do contrário é não trivial.

Por exemplo, considere o genoma

$$\vec{A_1} = \{(+1 + 2 + 4 + 3 + 5 + 6 + 8 + 7 + 9), (+10 + 11 - 12 + 13 + 14 + 15)\}.$$

A permutação P_{A_1} tem 7 componentes *intracromossomais* $[+1+2]$, $[+2\cdots+5]$, $[+5+6]$, $[+6\cdots+9]$, $[+10+11]$, $[+11\cdots+13]$ e $[+13+14]$. Elas podem ser representadas pelos enquadramentos na Figura 5.4.

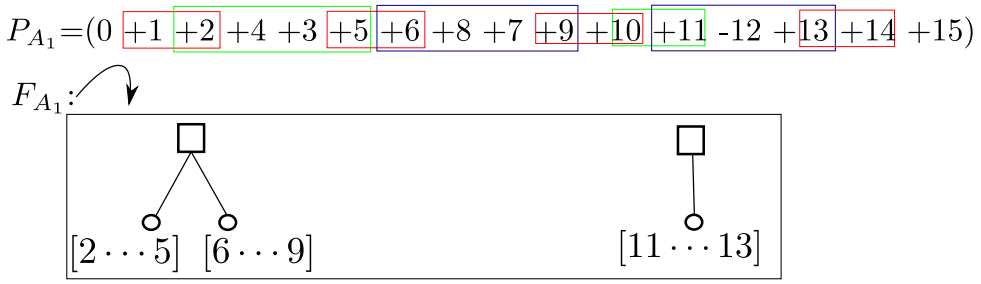

Figura 5.4: As componentes da permutação P_{A_1} e a floresta F_{A_1}.

As relações entre as componentes *intracromossomais* caracterizam uma importante propriedade no problema de distância de translocação entre genomas. Como mostrado por Bergeron e Stoye em [4], duas diferentes componentes *intracromossomais* podem ser classificadas tanto disjuntas, próximas com diferentes pontos ou sobrepostas em um elemento.

Quando duas componentes *intracromossomais* se sobrepõem em um elemento, é dito que elas estão ligadas. Sucessivas ligações formam uma cadeia. Uma cadeia que não pode ser estendida tanto para sua direita quanto para sua esquerda é considerada uma cadeia máxima. Representam-se as relações entre as componentes *intracromossomais* pertencentes a um cromossomo conforme a seguinte definição:

Definição 5.5 (Floresta de um cromossomo [5])**.** *Dado um cromossomo X e suas componentes intracromossomal, define-se a floresta F_X da seguinte forma:*

1. *Cada componente não trivial é representado por um nodo redondo.*
2. *Cada cadeia máxima contendo componentes não triviais é representada por um nodo quadrado e suas ligações (filhos) são representados por nodos redondos representando componentes não triviais pertencentes à cadeia.*
3. *Um nodo quadrado é o filho da menor componente que contém esta cadeia.*

A noção de floresta pode ser estendida para genomas combinando as florestas formadas a partir de todos os seus cromossomos.

Definição 5.6 (Floresta de um genoma [5])**.** *Dado um genoma $\vec{A}$ consistindo de cromossomos $\{X_1, X_2, \cdots, X_N\}$ a floresta F_A é o conjunto de florestas $\{F_{X_1}, F_{X_2}, \cdots, F_{X_N}\}$.*

Figura 5.4 mostra a floresta F_{A_1} consistindo de 2 árvores, onde o primeiro cromossomo contém uma floresta contendo uma única árvore com dois ramos e o segundo cromossomo contém uma floresta contendo uma árvore contendo um único ramo.

5.2.3 Relações entre Árvores e a Distância de Translocação

Lema 5.3 (Translocação adequada [5]). *Se um cromossomo X do genoma $\vec{A}$ contém mais que uma árvore, então existe uma translocação adequada envolvendo o cromossomo X.*

Demonstração. Considere o cromossomo $X = (x_1 \cdots x_m)$. Assuma que todos os intervalos elementares envolvidos no cromossomo X são *intracromossomal*. Logo, é necessário mostrar que o primeiro elemento do cromossomo é o menor e o último elemento é o maior, se ambos são positivos, e o contrário, se ambos são negativos, e todos os elementos entre o menor e maior estão contidos no cromossomo.

Seja i o menor elemento (desprezando o sinal positivo ou negativo) contido no cromossomo X. Suponha que i tenha sinal positivo e que $x_1 \neq i$. Assim, tem-se que o ponto à esquerda de i é um ponto do intervalo elementar I_{i-1}. Uma vez que, i é o menor elemento no cromossomo X o elemento $i-1$ está em outro cromossomo. Portanto, I_{i-1} é um intervalo *intercromossomais*, contrariando a suposição inicial de que todos os intervalos elementares são *intracromossomais*.

Seja j (desprezando o sinal) o maior elemento contido em X. Suponha que j tenha sinal positivo e que $x_m \neq j$. Logo, o ponto à direita de j é o ponto do intervalo I_j. Uma vez que, $x_m \neq j$, tem-se $j+1$ em outro cromossomo. Portanto, I_j é um intervalo *intercromossomal*, contrariando a hipótese de que todos os intervalos são *intracromossomais*.

Usando argumentos similares, pode-se mostrar que $x_1 = -j$, se j é o maior elemento e tem sinal negativo, e $x_m = -i$ se i tem sinal negativo. Além disso, todos os elementos entre i e j devem estar contidos no cromossomo X, do contrário tem-se intervalos *intercromossomais*. Assim, tem-se que X é uma componente *intracromossomal* e contém uma única árvore, o que leva a uma contradição. Portanto, existe um intervalo elementar *intercromossomal* contendo um elemento em X, pelo Lema **5.2**, a correspondente translocação é adequada.

□

Em um cenário de ordenação por translocações ótimo (utiliza o número minimo de translocações), quando se tem todas as árvores contidas em um único cromossomo, há a necessidade de separá-las (uma translocação é aplicada em dois cromossomos), isto significa mover elas para diferentes cromossomos através de translocações. O corolário a seguir mostra que tais separações é sempre possível com translocações que não modificam a topologia da floresta F_A.

Corolário 5.1 (Separando árvores [5]). *Se um cromossomo X de um genoma $\vec{A}$ contém mais que uma árvore em F_A, e nenhum outro cromossomo de $\vec{A}$ contém qualquer outra*

componente intracromossomal não trivial, então as árvores em X podem ser separadas por uma translocação adequada.

Demonstração. Pelo teorema 1 em [5], existe uma translocação que não muda F_A. Tal translocação tanto separa as árvores ou não. Se todas as árvores ainda continuam contidas no mesmo cromossomo, então, pelo mesmo argumento, existe uma outra translocação adequada que não muda o número de árvores. Assim, existem translocações adequadas que pode ou não separar as árvores de um cromossomo. Uma vez que, o número de translocações adequadas é finito, sempre existirá uma sequência de translocações adequadas que separam as árvores do cromossomo X sem modificar F_A. □

Lema 5.4 (Sequência ótima de translocações [5])**.** *Seja $\vec{A}$ um genoma, tal que F_A tem L folhas e T árvores. Se L é par e $T > 1$, então existe uma sequência de translocações adequadas, seguidas por uma translocação ruim, tal que o genoma resultante $\vec{A}$ tem $L' = L - 2$ folhas e $T' \neq 1$ árvores.*

Demonstração. Se todas as árvores estão contidas em um único cromossomo em $\vec{A}$, pelo Corolário **5.1** pode-se separá-las sem modificar a topologia de F_A utilizando translocações adequadas.

Assuma que existem árvores em diferentes cromossomos. Será mostrado que ao destruir um par de folhas representando componentes *intracromossomais* com uma translocação ruim reduz-se o número de folhas em 2, de formaf que o número de árvores no genoma resultante é 0 ou maior que 1.

Seja t_1 a árvore contida em F_A com o maior número de folhas e t_2 a segunda maior árvore em F_A não contida no mesmo cromossomo que t_1. Algumas observações podem ser feitas: Se t_1 contém uma única folha, obviamente todas as outras árvores também tem 1 folha, incluindo t_2. Assim, pode-se destruir t_1 e t_2 por uma translocação ruim. Neste caso, tinha-se $T = L$ antes de aplicar a translocação, e após a translocação tem-se $T' = L'$. Logo, pode-se afirmar que $T \neq 1$. Uma vez que, $L' = L - 2$ é par.

Se t_1 possui mais de uma folha, escolhe-se a segunda folha a partir da esquerda, emparelhando com qualquer folha de t_2. Executando uma translocação ruim, destrói-se as duas folhas criando uma nova árvore com a primeira folha à esquerda de t_1. Se $T' = 1$, então esta única árvore contém apenas uma folha, porém isso é impossível, já que o número de folha antes de aplicar a translocação ruim era par, e destruindo duas folhas em F_A tem-se $L' = L - 2$ sendo par, ou seja, o número de árvores $T' > 1$.

□

O Lema **5.4** implica que, quando o número de folhas é par e $T > 1$, tem-se um cenário ótimo para destruir toda a floresta. Assim, podem-se usar translocações adequadas para

separar a floresta (todas as folhas em um único cromossomo) e remover duas folhas com uma translocação ruim (isso equivale a destruir subpermutações). Eventualmente, todas as árvores serão destruídas, ou seja, ao final do procedimento tem-se $T = 0$. A ideia básica é reduzir todos os outros casos ao simples caso do Lema **5.4** .

Teorema 5.1 (Fórmula da distância de translocação [5]). *Seja $\vec{A}$ um genoma com c ciclos, n genes, N cromossomos e F_A a floresta associada com $\vec{A}$. Então*

$$d(\vec{A}) = n - N - c + t$$

onde

$$t = \left\{ \begin{array}{ll} L+2 \quad se\, L\ é\ par\ e\ T = 1 & (1) \\ L+1 \quad se\, L\ é\ impar & (2) \\ L+1 \quad se\, L\ é\ par\ e\ T \neq 1 & (3) \end{array} \right\}$$

Demonstração. Primeiramente, será mostrado que $d(\vec{A}) \geq n - N - c + t$. Considere um cenário de ordenação ótima de tamanho d contendo p translocações e b translocações ruins. Assim,

$$d = p + b \qquad\qquad (1),$$

com b translocações removendo ciclos e p translocações adicionando p ciclos, precisa-se ter

$$c - b + p = n - N,$$

ao final da ordenação, que seria ter todos os pares adjacentes como em $\vec{B}$. somando 2b aos dois lados da equação tem-se:

$$\begin{aligned} &c - b + 2b + p = n - N + 2b \\ &c + b + p = n - N + 2b \\ &c + d = n - N + 2b \quad utilizando\,(1) \\ &d = n - N - c + 2b \end{aligned}$$

Vamos mostrar que $2b \geq t$, implica $d \geq n - N - c + t$.

Uma vez que, uma translocação ruim remove no máximo duas folhas, tem-se $b \geq \frac{L}{2}$, se L é par, ou $b \geq \frac{(L+1)}{2}$, se L é ímpar. Portanto, casos (2) e (3): $b \geq \frac{t}{2}$.

Se existe uma única árvore com um número par de folhas, então existe uma translocação ruim no cenário ótimo que tem um ponto contido na árvore e outro não contido na árvore. Se esta translocação não destrói qualquer folha, então $b \geq 1 + \frac{L}{2}$.

Se esta translocação destrói uma componente mínima, o número de translocações ruins necessárias para destruírem as folhas remanescentes é $\frac{((L-1)+1)}{2}$, implicando que $b \geq 1+\frac{L}{2}$. Assim, no caso (1): $b \geq \frac{t}{2}$.

No caminho para mostrar que $d(\vec{A}) \leq n - N - c + t$, será mostrado uma sequência de translocações ruins que alcança o limite $n - N - c + t$.

A análise é por casos:

Caso (2): Se L é ímpar e $T = 1$, pode-se destruir uma folha do meio da árvore. Então $L - 1$ será par, e $T > 1$ ou $T = 0$. Se $T > 1$, tem-se as precondições do Lema **5.4** aplicáveis, e o número total de translocações ruins será $1 + \frac{(L-1)}{2}$.
Se L é ímpar e $T > 1$, destrói-se uma única folha de alguma árvore que contenha no mínimo duas folhas, se tal árvore existe. Do contrário, tem-se $T \geq 2$. Uma vez que, o número de folhas é ímpar, destrói-se qualquer folha. Em ambos os casos, tem-se $T' > 1$. Novamente, o número de translocações ruins será $1 + \frac{(L-1)}{2}$.

Caso (3): Se L é par e $T \neq 1$, tem-se as precondições do Lema **5.4** aplicáveis, e o número total de translocações ruins será $\frac{L}{2}$.

Caso (1): Se L é par e $T = 1$, destrói-se qualquer folha e aplica-se o caso (2). O número total de translocações ruins será $1 + \frac{L}{2}$. □

5.2.4 Algoritmo para computar $d(\vec{A}, \vec{B})$

O pseudo-código apresentado no Algoritmo 7 utilizado para computar a distância de translocação ótima é descrito como segue: assuma que os genomas $\vec{A}$ e $\vec{B}$, e uma permutação estendida P_A são fornecidas como entrada. Os ciclos de P_A são computados escaneando P_A da esquerda para direita sem levar em consideração os pontos brancos, os quais representam início e fim de um cromossomo em $\vec{A}$. Para computar as as componentes *intracromossomais* em cada cromossomo de $\vec{A}$ é utilizado uma adaptação do algoritmo proposto por Bergeron et al em [4], cujo tempo é linear em n. A floresta F_A é construída a partir das componentes *intracromossomais* em $O(n)$. Finalmente, a distância de translocação é computada utilizando a fórmula do Teorema **5.1** .

Algoritmo 7: Computação da Distância de Translocação entre Genomas

Input: Genomas $\vec{A}$ e $\vec{B}$ e a permutação estendida P_A
Output: A distância exata de translocações entre genomas $\vec{A}$ e $\vec{B}$

```
if Ā e B̄ são co-caudais then
    Armazene em c os ciclos em P_A;
    foreach cromossomo Crom em Ā do
        Guarde em Comp as componentes intracromossomais de Crom;
        Armazene em F_A a(s) árvore(s) construídas de Comp;
    end
    if L é par then
        if T=1 then
            t ← L + 2;
        end
        else
            t ← L;
        end
    end
    else
        t ← L + 1;
    end
    d ← n − N − c + t;
end
else
    Genomas Ā e B̄ não são co-caudais;
end
```

Capítulo 6

AG Aprimorado com Técnicas: Meméticas e Aprendizagem por Oposição

O capítulo possui a seguinte organização. A Seção 6.1 apresenta a metodologia utilizada nos algoritmos meméticos, os quais são conhecidos na área de computação evolutiva como extensões dos algoritmos genéticos. A Seção 6.2 apresenta um algoritmo memético para resolver o problema de *DTS*, juntamente com a sua análise de complexidade de tempo. Na sequência, a Seção 6.3 apresenta os termos e definições com relação a técnica de aprendizagem por oposição, para fechar, a Seção 6.4 apresenta um algoritmo genético melhorado com a sua respectiva análise de complexidade de tempo. Os *AGs* aprimorados com técnicas meméticas e aprendizagem por oposição foram publicados em [43].

6.1 Algoritmos Meméticos

Algoritmos Meméticos (AMs) são tidos como extensões dos algoritmos genéticos, com ênfase na exploração de uma dada população (como nos algoritmos genéticos) com uma ou mais fases de busca local. Tais algoritmos surgiram no fim da década de 80, após quase duas décadas do surgimento dos algoritmos genéticos, introduzido por Moscato em seu trabalho "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts"[34], baseados no conceito de meme.

A caracterização de meme o qual originou o termo memético, foi relatada por Dawkins em [15] como sendo similar ao gene, todavia no contexto da evolução cultural. Dawkins define meme como:

"Exemplos de memes são melodias, ideias, frases de efeito, modos de fabricação de arcos de edifício. Similar aos genes que se propagam na piscina de genes saltando de corpo em corpo por espermas ou ovos, memes se propagam na piscina de memes saltando de cérebro em cérebro mediante um processo que, no sentindo real, é tido como processo de imitação" [15].

Em [31] é definido que, a diferença entre genes e memes está na forma como é realizada a transmissão de informação aos descendentes. Ao transmitir informações através dos genes aos descendentes, tem-se que eles herdam tanto as habilidades quanto as características dos seus respectivos progenitores. Em contrapartida, os memes são adaptados aos indivíduos baseando-se no conhecimento adquirido para atender as suas necessidades. Assim, para Moscato ([35]), os *AMs* quando fazem uso de heurísticas tais como: métodos de aproximação, técnicas de busca local, etc, estão incorporando a ideia principal em relação à transmissão dos memes. Buriol em [8] também levanta outro ponto importante o qual diferencia memes e genes. A transmissão de informação para os genes seria somente através da criação de uma nova geração, enquanto que, para os memes isso pode ocorrer sem a necessidade de se ter uma nova geração. Levando em consideração os dois pontos de divergência entre *memes* e genes, base dos *AMs* e *AGs* respectivamente, tem-se que *AMs* além da evolução genética, utilizam o conceito de evolução cultural, enquanto que *AGs* procuram emular o processo de evolução biológica.

Na literatura, o mecanismo para transmitir a informação memética é a introdução de um ou mais passos de otimização local incluídos a um algoritmo evolutivo. Esse processo consiste em, dado um AG, adicionam-se fases de busca local transformando-o num AM, também conhecido como AG híbrido ([35]).

6.2 Algoritmo Memético para DTS

O algoritmo memético proposto aqui, denotado como AM para o problema DTS é baseado no AG apresentado no Capítulo 5.

O AM é um híbrido do AG, onde são incluídas fases de busca local nos seguintes estágios do AG:

- Após a geração da população inicial.
- Após o reinício da população.
- Após a geração de cada ciclo de reprodução.

O estágio de reinício da população é executado sempre que a população converge para um estado degenerado, que seria quando todos os indivíduos tem um alto grau de semelhança, o qual é mensurado utilizando a entropia de Shannon [40] que é definida como

segue:

$$H(S) = -\sum_{i} p_i \log_2 p_i,$$

onde S é o conjunto de diferentes elementos da população corrente, e p_i é a probabilidade de ocorrência do elemento i na população corrente. Quando a entropia tem um alto valor é dito que a população tem uma boa diversidade (indivíduos distantes um do outro), porém quando a entropia possui um baixo valor, tem-se uma população contida ou prestes a convergir em um estado degenerado.

A função de aptidão utilizada no *AM* é a mesma usada pelo *AG*. O pseudo-código do *AM* é ilustrado no Algoritmo 8.

Algoritmo 8: *AM* para DTS

Input: Genoma sem sinal A
Output: Número de translocações para transformar A numa identidade

```
1  Gerar a população inicial de A;
2  Computar a aptidão da população;
3  Aplicar busca Local sobre a população (Alg. 9);
4  for i = 1 até Tamanho(A) do
5  |  Selecionar os melhores indivíduos encontrados;
6  |  Aplicar operador de cruzamento;
7  |  Aplicar operador de mutação sobre os descendentes;
8  |  Computar a aptidão dos descendentes;
9  |  Substituir os piores indivíduos;
10 |  Aplicar busca Local sobre a população (Alg. 9);
11 |  if Limite de entropia é atingido then
12 |  |  Reiniciar a população;
13 |  |  Computar a aptidão da população;
14 |  |  Aplicar busca Local sobre a população (Alg. 9);
15 |  end
16 end
```

O método de otimização por busca local utilizado no *AM* consiste em um passo muito simples, o qual envolve modificar o sinal de um gene de um indivíduo em uma posição aleatória , e verificar se esta alteração traz alguma melhora na aptidão do indivíduo em questão. No caso em que o valor de aptidão é melhorado, mantém-se o novo sinal para este gene e o indivíduo retorna para a população. Para cada indivíduo, o número de possíveis modificações foi restringido para dois. O pseudo-código do procedimento de busca local é apresentado no Algoritmo 9.

Algoritmo 9: Técnica busca local

Input: $\vec{A}$ e sua aptidão
Output: $\vec{A'}$ com seu atual valor de aptidão

```
1  for i até numero_Iterações do
2  |  Gerar aleatoriamente uma posição k em Ā;
3  |  Inverter o sinal do gene na posição k, obtendo Ā;
4  |  if aptidão( Ā') > aptidão(Ā) then
5  |  |  Substituir Ā por Ā' na população;
6  |  |  Atualizar aptidão de Ā' na população;
7  |  end
8  |  else
9  |  |  Recuperar o último estado de Ā';
10 |  end
11 end
```

6.2.1 Análise da Complexidade de Tempo

Os estágios pertencentes ao AM nas linhas 1, 2, 6, 7, 8, 9, 13 possuem complexidade de tempo $O(n^2 \log n)$ e na linha 5 $O(n \log n)$, conforme argumentado na Seção 5.1.1 do Capítulo 5. Na linha 12, onde a população é reiniciada, a complexidade é de $O(n^2 \log n)$. Nas linhas 3, 10, 14 que é o Algoritmo 9, onde se tem o número de iterações restrito a 2, e consequentemente tendo a necessidade de calcular o valor de aptidão unicamente duas vezes, a complexidade é linear em n, uma vez que na análise de complexidade constantes são descartadas. Na linha 11, onde é computado o valor de entropia, implementa-se uma tabela hash com índices para cada diferente valor de aptidão encontrado na população corrente, e então faz-se o cálculo do valor de entropia. A complexidade exigida é de $O(n^2)$. Assim, se considerar unicamente o procedimento com maior complexidade, que seria o que contém complexidade quadrática, o AM possui complexidade de tempo de $O(n^3 \log n)$ (na linha 4 a complexidade é de $O(n)$).

6.3 Método de Aprendizagem por Oposição

Ao lidar como um problema de otimização, aprendizagem (adaptação ao cenário de possíveis soluções), é um fato de suma importância para a resolução do problema. Com esta base de pensamento, tem-se que, sempre que estiver procurando uma solução x de um dado problema, é preferível de alguma forma fazer uma estimativa de x. Tal estimativa não necessariamente estará representando a solução exata e pode ser obtida com base em uma suposição aleatória, ou provida por algum outro algoritmo que fornece uma

estimativa dentro de um fator próximo da solução ótima. Este último geralmente envolve a resolução de problemas complexos.

Em muitos casos, a aprendizagem começa num ponto aleatório e se desloca em direção a uma solução existente. Se tal suposição não se encontra muito longe da solução ótima, este ponto pode convergir rapidamente à solução existente. No entanto, ao iniciar de um ponto aleatório, o qual se encontra distante da solução ótima, pensando no pior caso onde a solução está em direção oposta, achar uma solução pode tomar muito mais tempo ou mesmo se tornar impossível. Todavia, se não há qualquer conhecimento a priori, não é possível fugir deste problema. Assim, a saída é olhar em todas as direções ao mesmo tempo o que para alguns problemas se torna inviável, ou de um maneira mais simples, na direção oposta.

Se beneficiando dos argumentos descritos acima, Tizhoosh em [46] propôs um método inovador de otimização, denominado aprendizado por oposição (*OBL-Opposition-based Learning*). A principal ideia sobre *OBL* é que, quando se está buscando uma solução em uma direção, pode ser uma boa ideia tomar a direção oposta. Isto incrementa as chances de melhorar a solução final especialmente quando se encontra em um cenário onde as soluções estão piorando. Em [47] e [1] foram definidos dois tipos de pontos de oposição, denotados como: tipo-I e tipo-II, os quais são descritos como segue:

Definição 6.1 (Número Oposto [47]). *Seja x um número real no intervalo $[a, b]$, o número oposto $\tilde{x}$ é definido da seguinte forma: $\tilde{x} = a + b - x$.*

Definição 6.2 (Ponto Oposto de Tipo-I [47]). *Seja $P = (x_1, x_2, \ldots, x_n)$ um ponto n-dimensional com x_i sendo um número real no intervalo $[a_i, b_i]$. O ponto oposto de tipo-I é definido por $\tilde{P} = (\tilde{x_1}, \tilde{x_2}, \ldots, \tilde{x_n})$, onde cada coordenada é definida como: $\tilde{x_i} = a_i + b_i - x_i$, com $i = 1, 2, \ldots, n$.*

Definição 6.3 (Ponto Oposto de Tipo-II [47]). *Seja f uma função arbitrária $\mathbb{R}^n \to \mathbb{R}$ com imagem no intervalo $[y_{min}, y_{max}]$. Para cada ponto n-dimensional $P = (x_1, \ldots, x_n)$, o ponto oposto de tipo-II é definido por $\tilde{f}(x_1, \ldots, x_n) = y_{min} + y_{max} - f(x_1, \ldots, x_n)$.*

Em simples palavras, oposição de tipo-I refere-se ao cálculo de uma posição oposta de um dado ponto P, que seria $\tilde{P}$, onde $f(P)$ e $f(\tilde{P})$ representam seus valores de aptidão respectivamente. Dada uma função, oposição de tipo-II refere-se ao cálculo oposto, que é uma função $\tilde{f}(P)$. Neste trabalho, foi considerado apenas oposição do tipo-I.

6.4 AG Melhorado com OBL para DTS

O algoritmo genético com *OBL* do tipo-I (*OBL-AG*) proposto para o problema *DTS* é baseado no *AG* apresentado no Capítulo 5, e tem similaridade com o *AM* se tratando de onde a técnica *OBL* é aplicada. Tal técnica é aplicada nos seguintes estágios do *AG*:

- Após gerar a população inicial.
- Após o término de uma geração do ciclo de reprodução.
- Após reiniciar a população.

Similarmente como no *AM* a convergência da população é controlada utilisando a entropia de Shannon [40]. Se tratando da função de aptidão usada pelo *OBL-AG*, utiliza-se a mesma função proposta para o *AG*. O pseudo-código do *OBL-AG* é apresentado no Algoritmo 10.

Algoritmo 10: *OBL-AG* para *DTS*

Input: Genoma sem sinal A
Output: Número de translocações para transformar A numa identidade

```
1  Gerar a população inicial de A;
2  Computar a aptidão da população;
3  Aplicar técnica OBL sobre a população (Alg. 11);
4  for i = 1 até Tamanho(A) do
5  |  Selecionar os melhores indivíduos encontrados;
6  |  Aplicar operador de cruzamento;
7  |  Aplicar operador de mutação;
8  |  Computar a aptidão dos descendentes;
9  |  Substituir os piores indivíduos;
10 |  Aplicar técnica OBL sobre a população (Alg. 11);
11 |  if Limite de entropia é atingido then
12 |  |  Reiniciar a população;
13 |  |  Computar a aptidão da população;
14 |  |  Aplicar técnica OBL sobre a população (Alg. 11);
15 |  end
16 end
```

A técnica *OBL* inclusa no *OBL-AG* consiste em aplicar oposição do tipo-I para um dado indivíduo representando um genoma com sinal, que seria inverter o sinal de cada gene interno aos cromossomos sem alterar os seus genes extremos. Este novo indivíduo oposto é adicionado à população sempre que o valor de sua aptidão seja melhor quando comparado com o indivíduo original. Caso contrário, este indivíduo oposto é descartado. O pseudo-código da técnica *OBL* é apresentando no Algoritmo 11.

Algoritmo 11: Técnica *OBL*

Input: $\vec{A}$ e seu valor de aptidão
Output: $\vec{A'}$ com seu atual valor de aptidão
1 $\vec{A'}$ = gerar oposição I para $\vec{A}$;
2 **if** *aptidão*($\vec{A'}$) > *aptidão*($\vec{A}$) **then**
3 | Substituir $\vec{A}$ por $\vec{A'}$ na população;
4 | Atualizar aptidão de $\vec{A'}$ na população;
5 **end**

6.4.1 Análise da Complexidade de Tempo

A análise da complexidade do *OBL-AG* é dada como segue. Uma vez que n representa o tamanho do genoma A fornecido como entrada, onde a população possui tamanho fixo em $n \log n$. O estágio de seleção na linha 5 tem complexidade de tempo $O(n \log n)$. Os estágios de cruzamento, mutação, função de aptidão, e substituição nas linhas 6, 7, 2, 8, 13 e 9 respectivamente, possuem complexidade de tempo $O(n^2 \log n)$. Na linha 11, onde se computa o valor de entropia, a complexidade é de $O(n^2)$. Para as linhas 1 e 12, onde se gera e reinicia a população respectivamente, a complexidade é de $O(n^2 \log n)$. Nas linhas 3, 10 e 14 (Algoritmo 11), onde se aplica a técnica de otimização *OBL*, a complexidade é limitada em $O(n^2 \log n)$. Assim, a complexidade total do *OBL-AG* é $O(n^3 \log n)$, uma vez que linha 4 (número de gerações), possui complexidade linear em n.

Capítulo 7

Versões em Paralelo do *AM*

O capítulo possui a seguinte organização. A Seção 7.1 apresenta um breve resumo relacionado a computação paralela. Na Seção 7.2 são abordadas duas métricas populares para avaliar programas em paralelo. Na sequência, a Seção 7.3 apresenta uma descrição sobre o padrão de troca de mensagens *Message Passing Interface* utilizado em computação paralela, na Seção 7.4 são apresentados conceitos relevantes sobre paralelização envolvendo algoritmos genéticos. Por fim, a Seção 7.5 apresenta 3 versões em paralelo do *AM*.

7.1 Computação Paralela

Computação paralela é um estratégia designada em computação, onde um programa é constituído por múltiplas tarefas que cooperam estreitamente para resolver um dado problema [38]. Assim, um programa em paralelo geralmente executa várias tarefas simultaneamente em núcleos que estão fisicamente perto uns dos outros onde tais núcleos podem compartilhar a mesma memória ou manter a sua própria memória. Ao utilizar paralelismo, tem-se em pauta um dos seguintes objetivos:

- Reduzir o tempo de processamento.
- Resolver grandes desafios computacionais.
- Aumentar a acurácia de soluções aproximadas para problemas não pertencentes a classe $\mathcal{P}$.

7.1.1 Classificação de Arquiteturas Paralelas

Arquiteturas paralelas são classificadas de acordo com a semelhança de suas características. A classificação mais popular é a proposta por Flyn em [18], que agrupa as

arquiteturas levando em considerações os seus fluxos de instruções e dados. A classificação é dada como segue:

Single Instruction, Single Data (SISD) Um único fluxo de instrução para um único fluxo de dados. Tal organização representa as máquinas sequenciais disponíveis no mercado atual baseadas na arquitetura de *Vonn Neumann*.

Single Instruction, Multiple Data (SIMD) Um único fluxo de instrução para múltiplos fluxos de dados. A classe *SIMD* corresponde aos computadores matriciais e vetoriais, os quais executam em um dado instante de tempo uma mesma instrução sobre um conjunto de dados.

Multiple Instruction, Single Data (MISD) Múltiplos fluxos de instruções para um único fluxo de dados. Máquinas que se encontram nesta classe, contém um pipeline de múltiplas unidades funcionais, o qual executa instruções independentes uma das outras sobre um único fluxo de dados. Todavia, não existe nenhuma implementação real desse tipo de arquitetura reportado na literatura [2].

Multiple Instruction, Multiple Data (MIMD) Múltiplos fluxos de instrução para múltiplos fluxos de dados. Máquinas inclusas nesta classe, consistem tipicamente de um conjunto de núcleos de processamentos independentes, no qual cada um dos núcleos possui tanto a sua própria unidade de controle quando a sua própria unidade lógica aritmética. A maior parte dos sistemas paralelos atuais estão inclusos nessa classe, abrangendo tanto máquinas multiprocessadores as quais mantém memória compartilhada quanto sistemas multicomputadores os quais mantém memória distribuída.

Após introduzir o conceito de paralelismo a nível de hardware, a próxima seção aborda o assunto do ponto de vista de software.

7.1.2 Ambiente Paralelo

Ao pensar em um ambiente apto a executar programas em paralelo quatro pontos devem ser levados em consideração. **1)** máquina que ofereça recursos para explorar paralelismo, **2)** sistema operacional que suporte paralelismo, **3)** uma linguagem de programação que ofereça suporte a um modelo de programação paralela e **4)** o modelo de programação paralela, os quais estão divididos em dois: memória compartilhada e *Message Passing*.

Os pontos citados acima, são detalhados como segue:

1. máquinas que se encontram na classe de arquitetura *MIMD*, uma vez que os recursos disponíveis são muito mais viáveis quando comparado com máquinas pertencentes as outras classes apresentadas na seção anterior.

2. Sistemas operacionais que implementam além do conceito de multitarefa o qual possibilita a execução concorrente de processos, o conceito de *multithreading*[1]. Os sistemas operacionais mais populares atuais: `UNIX`, `Windows` e `MAC OS` implementam tal conceito.

3. Há duas abordagens populares na literatura para explorar paralelismo através de linguagens de programação: paralelismo automático e extensões paralelas para linguagens.

    - Paralelismo automático está ligado ao conceito do desenvolvimento de compiladores, os quais são capazes de analisar o código sequencial de um dado programa e gerar uma versão em paralelo. A tarefa de paralelização consiste de três estágios, no primeiro identifica-se regiões no código sequencial propícias a aplicar paralelismo, em um estágio posterior, a partir das regiões identificadas é feito um mapeamento do paralelismo para a arquitetura da máquina alvo e por fim o código paralelo é gerado e otimizado. Compiladores paralelizadores são populares entre as linguagens `Fortran` e `C`, como exemplo tem-se: *SUIF* [49], *Parafrase-2* [39], *PGI* [33] entre outros. Embora essa abordagem seja cômoda ao desenvolvedor, visto que não precisa alterar o código fonte do programa sequencial. Todavia, a performance obtida através desta abordagem não é muito satisfatória. Atualmente, é uma das áreas mais cobiçada pelos pesquisadores de computação paralela.
    - A abordagem através de bibliotecas contendo extensões paralelas (exemplo: *MPI*, *OpenMP*) em linguagens de programação (`C`, `Fortram` e etc) requer um esforço maior do programador, uma vez que necessita de um embasamento teórico relacionado a computação paralela. A vantagem com tal abordagem, é que o desenvolvedor pode utilizar uma linguagem que esteja familiarizado e explorar o paralelismo através dos acréscimos de comandos e estruturas específicas para funções paralelas. A performance está intimamente ligada com a estratégia utilizada pelo desenvolvedor e o conhecimento semântico de paralelismo respectivamente.

4.

[1] sistema operacional possui a capacidade de executar várias threads (partes de um programa) simultaneamente.

- **Memória compartilhada:** todos os processos compartilham o mesmo espaço de memória, tal que o acesso é feito de maneira assíncrona. A vantagem de tal modelo está na comunicação entre os processos o qual é realizada de maneira implícita. A desvantagem consiste que, tal modelo pode ocasionar condição de corrida, onde dois ou mais processos estão acessando dados compartilhados de forma concorrente, e o resultado final do processamento vai depender da política de execução utilizada nos processos para controlar o acesso a estes dados compartilhados. Dentre implementações de biblioteca conhecidas que utilizam este modelo tem-se: *Posix-Threads* (somente em `C`) e *OpenMP* (`C`, `C++` e `Fortran`) e etc.
- ***Message Passing*:** modelo normalmente associado com ambiente de memória distribuída. Neste modelo, cada processo mantém o próprio espaço de memória e a comunicação entre eles é feita via troca de mensagens. Estas trocas de mensagens podem ser síncronas ou assíncronas. Um sistema de passagem de mensagens síncronas requer que o emissor e o receptor esperem um pelo outro durante a transferência da mensagem. Na comunicação assíncrona, emissor e receptor não esperam um pelo outro e podem continuar a execução enquanto a transferência da mensagem está sendo realizada. A Vantagem deste modelo consiste no número ilimitado de processadores que podem ser utilizados, além disso cada processador acessa sem interferência e rapidamente a sua própria memória.

  A Desvantagem se dá tanto pelo overhead imposto pela troca de mensagens quanto pela responsabilidade dada ao desenvolvedor pelo o envio e recebimento de dados o que de certa forma ocasiona muito mais chances de se ter erros semânticos. Como exemplo de bibliotecas que segue este modelo tem-se: *PVM* e *MPI* ambas disponíveis para as linguagens `C`, `C++` e `Fortran`.

7.2 Métricas de Avaliação para Programas em Paralelo

O principal objetivo ao escrever programas em paralelo, geralmente é desempenho. Deste modo, necessita-se de métricas para avaliar o quão bons são estes programas.

7.2.1 *Speedup* e Eficiência

Ao utilizar paralelismo, o melhor cenário possível consiste em dividir igualmente o trabalho entre os núcleos sem introduzir nenhum *overhead* adicional. Se isto é possível para um dado programa, onde se tem p núcleos disponíveis para executá-lo, este programa

é executado paralelamente p vezes mais rápido do que a versão sequencial. Em uma representação matemática, seja T_{serial} o tempo da versão sequencial, $T_{paralelo}$, o tempo da versão em paralelo é obtido pela seguinte equação:

$$T_{paralelo} = \frac{T_{serial}}{p}.$$

Quando se obtém um cenário conforme discutido acima, tem-se um programa em paralelo com *speedup* linear.

Na prática, é difícil obter um *speedup* linear, porque múltiplos processos acabam introduzindo alguma sobrecarga. Por exemplo, programas de memória compartilhada quase sempre utilizam seções críticas, o que por sua vez exige que algum mecanismo de exclusão mútua seja aplicado, tal como um mutex [2]. Ao utilizar chamadas para o mutex tem-se sobrecargas que não se encontram na versão sequencial do programa, além disso a utilização do mutex exige que o programa paralelo execute sequencialmente uma dada seção crítica. Em se tratando de memória distribuída, os programas quase sempre necessitam trocar mensagens através da rede, ocasionando um gargalo quando comparado com acessos a memória local. Por outro lado os problemas citados anteriormente não se encontram em programas sequenciais. Logo, é muito incomum que programas paralelos obtenham um *speedup* linear e, é provável que os custos tendem a aumentarem à medida que se incrementa o número de processos. Levando em consideração cada ponto citado acima, ([38]) a equação utilizada para mensurar o *speedup* de um programa paralelo é

$$S = \frac{T_{serial}}{T_{paralelo}}$$

Outra métrica para mensurar programas paralelos, denominada como eficiência, tem valor dado pela equação abaixo ([38])

$$E = \frac{S}{p}$$

.

Quanto mais próximo de 1 o valor de E, melhor será a eficiência de um programa paralelo, todavia um valor igual a 1 só será alcançado caso o programa tenha *speedup* linear. Em comum, ambas as métricas *speedup* e eficiência são dependentes do número de processos utilizados.

[2]Mutex é uma técnica usada em programação concorrente para evitar que dois processos obtenham acesso simultaneamente a um recurso compartilhado.

7.3 *Message Passing Interface*

A seguir será dado um introdução de *Message Passing Interface* (*MPI*), levando em consideração alguns aspectos importantes do ambiente bem como algumas rotinas que serão utilizadas na Seção 7.5 onde são propostas versões em paralelo do *AM*. A escolha por *MPI* se deve a facilidade e flexibilidade comparada com as outras biblioteca de meu conhecimento, tais como *OpenMP* e *Posix-Threads*. O conteúdo teórico em relação ao *MPI* apresentado a seguir foi capturado da documentação oficial do *MPI* que se encontra disponível em `http://www.open-mpi.org/`.

MPI é uma biblioteca de troca de mensagens para computação paralela criado na década de 90 por um grupo composto por universidades, órgãos governamentais e representantes de indústrias.

Neste padrão, uma aplicação é constituída por dois ou mais processos, onde estes processos possuem um identificador único atribuído no instante em que o processo é criado.

Cada processo mantém o seu próprio espaço de memória, de forma que a comunicação entre diferentes processos é realiza através de protocolos específicos de comunicação. Isto é feito definindo coleções de processos (grupos), que poderão se comunicar entre si de forma segura.

Em se tratando dos tipos de dados, *MPI* suporta todos os tipos primitivos (*float,double, int, char, etc*), além disso fornece facilidade para os usuários definirem suas próprias estruturas de dados baseando-se nos tipos de dados primitivos.

As rotinas do *MPI* necessitam que sempre seja especificado um comunicador como argumento, que é o responsável por identificar um grupo de processos e o contexto em relação ao qual uma determinada operação deve ser executada. Deste modo, o *MPI* fornece um comunicador pré definido chamado
MPI_ COMM_ WORLD, que por padrão inclui todos os processos criados pelo usuário ao iniciar a aplicação.

Há diferentes tipos de envio de mensagens providos pelo *MPI* na forma com e sem bloqueio, tais como:

- Envio síncrono, só é completado quando o recebimento é confirmado.
- Envio bufferizado, sempre é completado (caso não ocorra um erro), independentemente do recebimento ser confirmado.
- Envio pronto, sempre é completado (caso não ocorra um erro), mesmo que o recebimento não seja confirmado.

Quando ocorre com bloqueio, a rotina que enviou a mensagem necessita de confirmação de recebimento por parte do processo designado ao recebimento para continuar o fluxo corrente. Na forma sem bloqueio, não há a necessidade de confirmar o recebimento da mensagem, assim a rotina continua normalmente após o envio de uma dada mensagem.

Na sequência será apresentado as rotinas providas pelo *MPI* que serão utilizadas para o nosso propósito na Seção 7.5.

7.3.1 Rotinas *MPI*

Antes de enunciar as rotinas, serão apresentados os argumentos básicos que tais rotinas necessitam (lembrando que são apenas nomes fictícios, podem ser usados quaisquer outros).

- **buf:** local onde armazenar os dados de envio ou recebimento.
- **count:** quantidade de dados a ser enviado ou recebido.
- **dataype:** o tipo de dado (tipo primitivo ou definido).
- **dest:** especifica o processo que irá receber a mensagem.
- **source:** específica de qual processo irá receber a mensagem.
- **tag:** uma maneira de identificar os tipos de mensagens. Assim, um processo de recebimento pode usar a **tag** para decidir quais mensagens quer receber em um determinado momento.
- **comm:** indica o contexto da comunicação, ou um conjunto de processos para os quais os campos de origem ou de destino são válidos. A menos que o programador crie explicitamente novos comunicadores, o comunicador *MPI_COMM_WORLD* é geralmente utilizado.

MPI_Send Operação básica de envio bloqueante. Tal rotina somente retorna após o buffer da aplicação estiver livre para reutilização. Esta rotina pode ser implementada de forma diferente em sistemas operacionais diferentes. A sintaxe utilizada é:
MPI_Send (buf,count,datatype,dest,tag,comm)

MPI_Recv Recebe uma mensagem e bloqueia até que os dados requisitados estejam no buffer de recepção. A sintaxe utilizada é:
MPI_Recv (buf,count,datatype,source,tag,comm,status)

MPI_ Barrier Rotina que força todos os processos a esperarem em um mesmo ponto até que todos os processos tenham alcançado tal ponto. A sintaxe utilizada é:
MPI_ Barrier(comm)

7.4 Paralelização com Algoritmos Genéticos

A ideia de paralelizar algoritmos evolutivos existe desde a década de 80 com Pettey et al. em [45]. Nesse trabalho, foi utilizado um algoritmo genético contendo dois operadores (cruzamento e mutação) com uma única população de tamanho estático para análise de cadeias de Markov.

De acordo com os trabalhos ([11] [21], [9]), existem três modelos de paralelização de algoritmos genéticos:

Granularidade Grossa Neste modelo, cada processo mantém a sua própria população isolada evoluindo em paralelo, periodicamente há uma migração dos melhores indivíduos entre as populações, o que de certa forma, leva a uma elitização das populações.

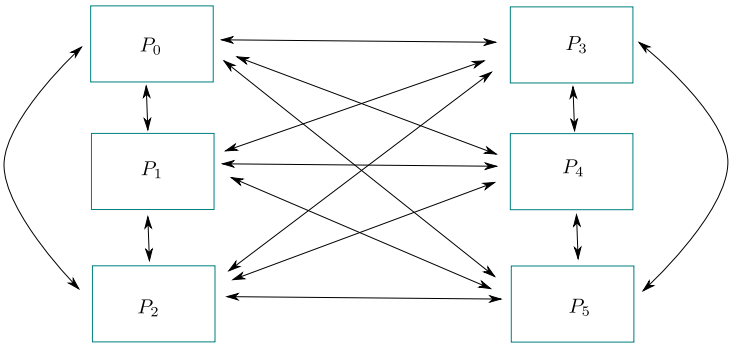

Figura 7.1: Esquema de um algoritmo genético utilizando paralelismo de granularidade grossa.

Granularidade Fina Neste modelo, cada processo mantém uma população separada e ao mesmo tempo sobreposta com outros processos, de forma que na fase de seleção, somente indivíduos pertencentes a mesma população podem reproduzir. Devido a sobreposição, muitos indivíduos fazem parte de mais de uma população, permitindo que o material genético destes indivíduos sejam utilizados em mais de um ciclo evolutivo.

Global Nesse modelo, dado um conjunto de processos, um processo tido como mestre mantém toda a população, e envia blocos de um ou mais indivíduos desta população aos demais processos(escravos), os quais podem realizar tanto o ciclo de reprodução quanto

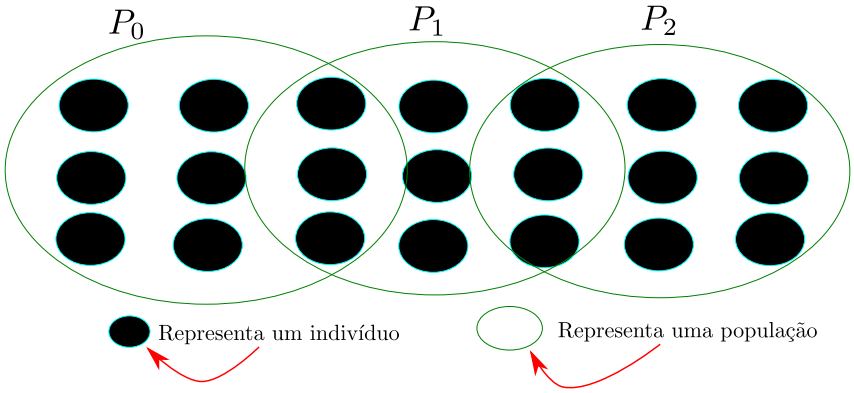

Figura 7.2: Esquema de um algoritmo genético utilizando paralelismo de granularidade fina.

o cálculo do valor de aptidão, ficando responsáveis por retornar as entradas manipuladas ao processo mestre.

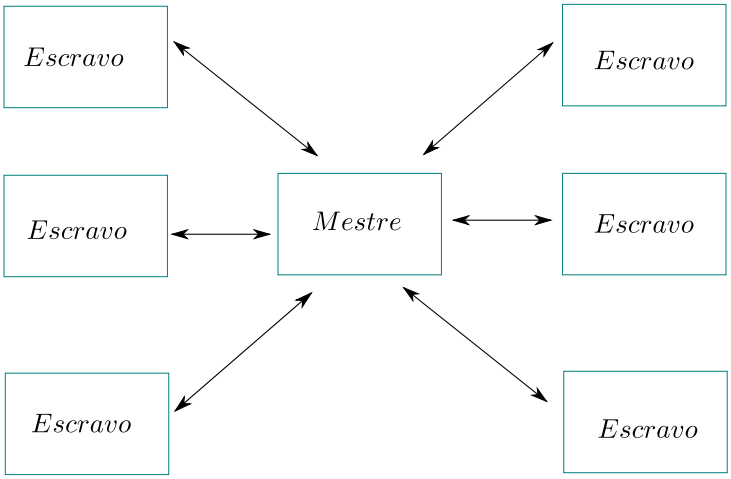

Figura 7.3: Esquema de um algoritmo genético utilizando paralelismo global.

7.5 Contribuição

Apresentam-se duas versões em paralelo do algoritmo *AM* proposto no Capítulo 6. **1)** Paralelização do cálculo da função de aptidão do *AM*, onde a busca é por aprimorar o desempenho de tempo de execução. **2)** *AM* com múltiplas populações, onde a busca é por melhoria na qualidade das soluções.

As versões em paralelo do *AM* fazem uso da biblioteca *MPI* implementada na linguagem `C`. Para a troca de mensagens foram utilizados as rotinas *MPI_ send* and *MPI_ receive*.

7.5.1 Paralelizando a Função de Aptidão do *AM*

Esta paralelização (chamada AM_F) é baseada no modelo global (mestre-escravo) contendo uma única população. O AM_F mantém tal população no processo mestre e, os processos escravos são usados para computar a função de aptidão em paralelo. A ideia principal desta paralelização é acelerar o tempo de execução do *AM*, todavia mantendo a precisão dos resultados (número de translocações). Como forma de avaliação serão usadas as duas métricas discutidas na Seção 7.2, juntamente com a avaliação da quantidade de indivíduos processados por segundo. O Algoritmo 12 mostra o pseudo-código executado pelo processo mestre e o pseudo-código executado pelos processos escravos, enquanto o Algoritmo 13 apresenta o procedimento de envio de indivíduos e recebimento dos valores de aptidão em paralelo.

Algoritmo 12: AM_F para DTS com o cálculo da aptidão em paralelo

Input: Genoma sem sinal A
Output: Número de translocações para transformar A numa identidade

```
   /* Pseudo-código do processo mestre:                                    */
 1 Gerar a população inicial de A;
 2 Computar aptidão da população em paralelo (Alg. 13);
 3 Aplicar busca local sobre a população (Alg. 9);
 4 Computar aptidão da população em paralelo (Alg. 13);
 5 for i = 1 até Tamanho(A) do
 6 |   Selecionar os melhores indivíduos encontrado;
 7 |   Aplicar operador de cruzamento;
 8 |   Aplicar operador de mutação;
 9 |   Computar aptidão dos descendentes em paralelo (Alg. 13);
10 |   Substituir os piores indivíduos;
11 |   Aplicar busca local sobre a população (Alg. 9);
12 |   if Limite de entropia é atingido then
13 |   |   Reiniciar a população;
14 |   |   Computar a aptidão da população em paralelo (Alg. 13);
15 |   |   Aplicar busca local sobre população (Alg. 9);
16 |   end
17 end
   /* Pseudo-código do processo escravo:                                   */
18 while 1 do
19 |   Receber um indivíduo a partir do processo mestre (MPI_Send);
20 |   Computar aptidão do indivíduo ;
21 |   Enviar o resultado para o processo mestre (MPI_Recv);
22 end
```

O cálculo do valor de aptidão realizado na busca local também é feito em paralelo (linhas 3, 11 e 15 no Algoritmo 12), onde o procedimento apresentado no Algoritmo 13

Algoritmo 13: Procedimento Executado pelo Processo Mestre

```
1  for todos os processos escravos do
2  |  Enviar um indivíduo da população (MPI_Send);
3  end
4  while existir individuo na população sem valor aptidão do
5  |  Receber resultado de um processo escravo (MPI_Recv);
6  |  Enviar para um processo escravo um indivíduo da população (MPI_Send);
7  end
8  for todos os processos escravos do
9  |  Receber resultado de um processo escravo (MPI_Recv);
10 end
```

envia um indivíduo para um dado processo escravo, o qual fica responsável por aplicar a busca local e retornar um indivíduo melhor adaptado com seu respectivo valor de aptidão (quando se alcança um indivíduo melhor adaptado).

Note que, o processo de computar o valor de aptidão em paralelo é bem simples. Primeiramente, é fornecido a todos os processos um indivíduo (representando um genoma), no segundo laço, sabe-se que todos os processos estão ocupados, logo há necessidade de receber os resultados para que possa alocar novos indivíduos a estes processos. No fim, tem-se processos contendo resultados que ainda não foram devolvidos, assim o último laço certifica-se que todos os indivíduos da população tenham tido seus valores de aptidão computados e devolvidos ao processo mestre.

7.5.2 *AM* com Múltiplas Populações com Troca de Indivíduos

Esta paralelização é denotada como AM_{MPT} (algoritmo memético com múltiplas populações com troca de indivíduos) e está baseada no modelo de granularidade grossa. O AM_{MPT} mantém multi populações, onde cada população está alocada em um processo diferente. Em cada geração é executado um estágio no qual todos os processos trocam seus melhores indivíduos, todos os processos continuam seus ciclos de reprodução após todos eles terem alcançado uma barreira (rotina *MPI_Barrier*), o qual é executado para sincronizar os processos. Tal paralelização visa unicamente melhorar a precisão dos resultados (número de translocações) explorando uma população maior.

No Algoritmo 14 é mostrado o pseudo-código do AM_{MPT} e no Algoritmo 15 é descrito o procedimento para realizar a troca de indivíduos entre as populações em paralelo.

Além disso uma variante do AM_{MPT} é proposta, chamada AM_{MPS} (algoritmo memético com multi populações sem troca de indivíduos). A principal característica desta variante em relação ao AM_{MPT} é que não há nenhuma troca de indivíduos entre as popu-

lações, implicando que cada população evoluirá de forma independente. O pseudo-código do AM_{MPS} é obtido ao eliminar as linha 6 e 7 do Algoritmo 14.

Algoritmo 14: *MA* paralelo com multi populações para UTD (AM_{MPT}), onde cada processo mantém uma diferente população inicial

Input: Genoma A
Output: Número de translocações para transformar A numa identidade

```
1  Gerar a população inicial de A;
2  Computar a aptidão da população;
3  Aplicar busca Local sobre a população (Alg. 9);
4  for i = 1 até Tamanho(A) do
5  |  Selecionar os melhores indivíduos encontrados;
6  |  Envia o melhor indivíduo, e receber indivíduos (Alg. 15);
7  |  MPI_Barrier;
8  |  Aplicar operador de mutação sobre os descendentes;
9  |  Computar a aptidão dos descendentes;
10 |  Substituir os piores indivíduos;
11 |  Aplicar busca Local sobre a população (Alg. 9);
12 |  if Limite de entropia é atingido then
13 |  |  Reiniciar a população;
14 |  |  Computar a aptidão da população;
15 |  |  Aplicar busca Local sobre a população (Alg. 9);
16 |  end
17 end
18 Retornar o valor de aptidão do melhor indivíduo dentre todas as populações;
```

O parâmetro p que aparece no Algoritmo 15 é utilizado para controlar quando substituir um indivíduo ruim (baixa aptidão) na população corrente por um recebido de outra população. Este parâmetro será definido no próximo capítulo.

Algoritmo 15: Procedimento para enviar e receber indivíduos em paralelo

```
1  for i = 1 to numeroProcesssos do
2  |  if i não é processo corrente then
3  |  |  Enviar o melhor indivíduo para processo i (MPI_Send);
4  |  end
5  end
6  for i = 1 to numeroProcessos do
7  |  if i não é processo corrente then
8  |  |  Receber melhor indivíduo a partir do processo i (MPI_Recv);
9  |  |  Gerar um número float r entre 0 e 1;
10 |  |  if r < p then
11 |  |  |  Substitui o pior indivíduo na população pelo indivíduo recebido;
12 |  |  end
13 |  end
14 end
```

Capítulo 8

Experimentos e Resultados

O capítulo possui a seguinte organização. A Seção 8.1 apresenta o ambiente utilizado para realizar os experimentos. A Seção 8.2 apresenta um roteiro para realizar tanto a definição dos parâmetros quanto os experimentos para os algoritmos genéticos propostos em capítulos anteriores. Por fim, a Seção 8.3 apresenta a discussão dos resultados obtidos.

8.1 Ambiente de Execução

Para os experimentos, foi utilizada uma máquina de 64 GB de memória RAM com sistema operacional Ubuntu-Server contendo dois processadores Intel Xeon E5-2620. Cada processador contém 6 núcleos (cada núcleo com velocidade de 2.4 GHZ) com *Hyper-threading* habilitado apresentando dois níveis de cache acoplado a cada núcleo (L1 32Kb e L2 256 Kb) e uma cache externa (L3) compartilhada entre todos os núcleos de 15 MB.

8.2 Experimentos

A fim de validar os AGs propostos, testes foram realizados. Para estes testes utilizou-se tanto genomas sintéticos quanto genomas biológicos. Para construir os genomas sintéticos foi proposto um algoritmo como segue: produz-se um genoma identidade contendo n genes e N cromossomos. Então, sobre este genoma identidade, aplica-se um número fixo de translocações e reversões. O pseudo-código deste procedimento é apresentado no Algoritmo 16.

Em uma fase inicial os parâmetros de cada algoritmo evolutivo proposto foram ajustados via experimentos utilizando possíveis bons valores conforme comentado no Capítulo 2, Seção 2.2.1, tal procedimento visa obter resultados de maior acurácia quando comparado com as soluções providas pelos algoritmos evolutivos sem ajustes nos parâmetros.

Algoritmo 16: Gerador de genomas sintéticos

Input: Número de genes n com N cromossomos
Output: Um genoma sintético A

```
1  Gerar um genoma identidade A com n genes e N cromossomos;
2  j ← 0;
3  while j ≤ n do
4  |  Escolher randomicamente um cromossomo C de A;
5  |  Selecionar randomicamente um intervalo em C;
6  |  Aplicar uma reversão sobre este intervalo;
7  |  Escolher randomicamente dois cromossomos C e C' de A;
8  |  Applicar uma translocação Prefixo-Prefixo entre segmentos de C e C';
9  |  j ← j + 1;
10 end
```

8.2.1 Definição dos Parâmetros dos Algoritmos

A seguir será apresentado o procedimento utilizado em cada algoritmo genético proposto para fixar os valores dos seus respectivos parâmetros.

Definindo os Parâmetros do *AG*

O AG foi executado dez vezes para cada grupo contendo genomas com n genes, para $n \in \{30, 50, 100, 150\}$, com 25% de cada grupo contendo N cromossomos, com $N \in \{2, 3, 4, 5\}$. Para ajustar cada um dos parâmetros, valores foram estimados sobre um cenário de possíveis bons valores, e valores aleatórios foram fixados para os demais parâmetros que não obtiveram um bom valor atribuído através dos experimentos.

No fim dos experimentos os parâmetros que forneceram os melhores resultados para o AG foram tomados. Os parâmetros com seus respectivos valores são apresentados na Tabela 8.1.

Tabela 8.1: Valores correspondentes aos respectivos parâmetros do AG.

Parâmetro	Valores Estimados	Melhor Valor
Probabilidade de cruzamento	$\{30\%, 40\%, \cdots, 90\%, 100\%\}$	90%
Probabilidade de mutação	$\{1\%, 2\%, 3\%, 4\%, 5\%\}$	2%
Num. de pontos de cruzamento	$\{1, 2, 3\}$	1
% para seleção	$\{30\%, 40\%, \cdots, 90\%, 100\%\}$	80%
% para substituição	$\{30\%, 40\%, \cdots, 90\%, 100\%\}$	70%

Definição dos Parâmetros para *AM* e *OBL-AG*

Os parâmetros utilizados no *AG* são mantidos e acrescenta-se um novo grupo para ambos *AM* e *OBL-AG* contendo três novos parâmetros: porcentagem da população para o qual busca local ou *OBL* é aplicada, porcentagem da população corrente que será preservada após reiniciar a população e o limite de entropia.

Os parâmetros de ambos algoritmos foram divididos em 3 grupos.

- G_1**:**
  - p_1: probabilidade de cruzamento
  - p_2: probabilidade de mutação
  - p_3: Número de pontos de cruzamento
- G_2**:**
  - p_4: porcentagem de seleção
  - p_5: porcentagem de substituição sobre a população corrente.
- G_3**:**
  - p_6: porcentagem da população corrente para o qual busca loca ou *OBL* é aplicada
  - p_7: porcentagem da população corrente que vai ser preservada após reiniciar a população
  - p_8: Máximo limite de entropia

Seja p_i um parâmetro de um grupo G_l, um intervalo discreto foi gerado para este parâmetro e os outros parâmetros foram fixados com um valor estimado ou com o seu valor já computado via experimentos. Em seguida, testes exaustivos foram executados para os possíveis valores de p_i. No fim dos testes experimentais, o melhor valor discreto encontrado foi tomado. Este processo é executado para cada parâmetro p_i com $0 < i < 9$ contido em cada grupo G_l, para $0 < l < 4$. No fim do processo, os melhores valores obtidos para os parâmetros do *AM* e do *OBL-AG* são apresentados nas Tabelas 8.2 e 8.3, respectivamente.

Definição dos Parâmetros dos Algoritmos Paralelos

O AM_F (Algoritmo 12) visa unicamente acelerar a execução do *AM*, deste modo, foram utilizados os mesmos parâmetros.

Tabela 8.2: Valores correspondentes aos respectivos parâmetros do AM.

	Parâmetro	Valores Estimados	Melhor Valor
p_1	Probabilidade de cruzamento	$\{30\%, 40\%, \cdots, 90\%, 100\%\}$	90%
p_2	Probabilidade de mutação	$\{1\%, 2\%, 3\%, 4\%, 5\%\}$	2%
p_3	Num. de pontos de cruzamento	$\{1, 2, 3\}$	1
p_4	% para seleção	$\{30\%, 40\%, \cdots, 90\%, 100\%\}$	80%
p_5	% para substituição	$\{20\%, 30\%, \cdots, 90\%, 100\%\}$	70%
p_6	% para busca local	$\{20\%, 30\%, \cdots, 90\%, 100\%\}$	80%
p_7	% de preservação	$\{20\%, 30\%, \cdots, 80\%, 90\%\}$	60%
p_8	Max. limite entropia	$\{0.1, 0.2, \cdots, 0.9, 1.0\}$	0.3

Tabela 8.3: Valores correspondentes aos respectivos parâmetros do OBL-AG.

	Parâmetro	Valores Estimados	Melhor Valor
p_1	Probabilidade de cruzamento	$\{0.3, 0.4, \cdots, 0.9, 1.0\}$	90%
p_2	Probabilidade de mutação	$\{0.01, 0.02, 0.3, 0.4, 0.5\}$	2%
p_3	Num. de pontos de cruzamento	$\{1, 2, 3\}$	1
p_4	% para seleção	$\{0.3, 0.4, \cdots, 0.9, 1.0\}$	80%
p_5	% para substituição	$\{0.2, 0.3, \cdots, 0.9, 1.0\}$	70%
p_6	% para OBL	$\{0.2, 0.3, \cdots, 0.9, 1.0\}$	70%
p_7	% de preservação	$\{0.2, 0.3, \cdots, 0.9, 1.0\}$	50%
p_8	Max. limite entropia	$\{0.1, 0.2, \cdots, 0.9, 1.0\}$	0.1

No segundo algoritmo proposto, o qual busca melhorar a precisão dos resultados do *AM*, foi inserido um novo parâmetro (P) para controlar a quantidade de inserções que são realmente feitas em uma dada população, esse parâmetro representa a probabilidade que um indivíduo tem de ser adicionado em uma determinada população. O experimento para determinar tal parâmetro foi feito da seguinte forma: foi gerado um intervalo discreto de possíveis valores para p, onde $p \in \{0.1, 0.2, \cdots, 0.9, 1.0\}$. Para cada valor de p, AM_{MPT} (Algoritmo 14) foi executado para grupos de 100 genomas de n genes, com $n \in \{100, 110, 120, 130, 140, 150\}$, onde 33.3% contém N cromossomos, para $N \in \{3, 4, 5\}$. Ao fim do experimento, foi identificado que os melhores resultados são providos com $p = 0.2$, ou seja apenas 20% de todos os indivíduos que chegam para eventuais inserções, realmente serão incorporados a população. Para a variante proposta na qual não há troca de indivíduos (AM_{MPS}) $p = 0.0$.

8.2.2 Experimentos com Genomas Sintéticos

Para facilitar o entendimento, os experimentos foram divididos em duas partes. A primeira trata unicamente das execuções com algoritmos sequenciais e, a segunda, unicamente com execuções com algoritmos paralelos.

Execuções com Algoritmos Sequenciais para Grupos de Genomas

Experimentos com *AG*, *OBL-AG*, *AM* e o algoritmo aproximado de raio 1.5+ε foram conduzidos como segue: Inicialmente, grupos contendo 100 genomas sem sinais com n genes, para $n \in \{30, \cdots, 140, 150\}$, e com N cromossomos, para $N \in \{2, 3, 4, 5\}$ foram gerados. Para cada genoma o algoritmo aproximado foi executado uma única vez e, os três algoritmos evolutivos foram executados dez vezes e a média destas dez execuções é tomada como resultado para cada algoritmo. Finalmente, para cada família de 100 genomas de tamanho (n, N) a média dos resultados foi calculada. Veja Tabelas 8.4 e 8.6 (a relação de tempo desses resultados é apresentada nas Tabelas 8.5 e 8.7).

Tabela 8.4: Média de translocações para genomas sintéticos com 2 e 3 cromossomos.

	2 cromossomos				**3 cromossomos**			
n	1.5Apr	*AG*	*OBL-AG*	*AM*	1.5Ap	*AG*	*OBL-AG*	*AM*
30	18.53	16.34	16.32	16.25	18.03	15.83	15.81	15.76
40	26.65	23.07	23.05	22.91	26.40	22.70	22.68	22.52
50	34.75	30.08	30.03	29.75	33.60	29.39	29.36	29.14
60	43.12	37.22	37.18	36.80	41.75	36.22	36.16	35.85
70	51.56	44.34	44.28	43.80	50.34	43.62	43.59	43.09
80	59.80	51.97	51.92	51.28	58.05	50.16	50.14	49.58
90	68.26	59.20	59.14	58.42	66.08	57.49	57.38	56.76
100	76.16	66.35	66.30	65.49	75.17	65.23	65.16	64.40
110	85.32	74.10	73.99	73.14	83.71	72.65	72.58	71.68
120	93.65	81.59	81.55	80.60	91.04	79.14	79.12	78.18
130	102.00	88.89	88.80	87.72	100.39	87.55	87.42	86.43
140	110.55	96.66	96.68	95.58	109.16	95.21	95.13	94.04
150	118.59	103.83	103.76	102.70	116.35	101.76	101.76	100.60

Execuções com Algoritmos Sequenciais para Genomas *Benchmarks*

Nove genomas *Benchmarks* foram propostos em [43]. Os genomas possuem a seguinte nomenclatura *GenomaLxCy*, onde L representa o padrão para tamanho e C o padrão para o número de cromossomos. Para cada genoma, *AG*, *AM* e *OBL-AG* foram executados 50 vezes, e então as seguintes medidas foram calculadas: média, mediana, mínimo e máximo. Veja as Tabelas 8.8 e 8.9, onde os melhores valores estão destacados em negrito.

Execuções com Versões em Paralelo do *AM*

Dois experimentos foram realizados. Um para calcular o tempo de execução do AM_F (Algoritmo 12) confrontando a quantidade de indivíduos que podem ser processados por

Tabela 8.5: Tempo de execução (em segundos) dos algoritmos evolutivos e do algoritmo aproximado de raio 1.5+ε para genomas sintéticos com 2 e 3 cromossomos.

	2 cromossomos				**3 cromossomos**			
n	1.5Apr	*AG*	*OBL-AG*	*AM*	1.5Ap	*AG*	*OBL-AG*	*AM*
30	0.09	0.09	0.12	0.23	0.07	0.10	0.14	0.25
40	0.09	0.24	0.32	0.56	0.07	0.24	0.34	0.61
50	0.10	0.46	0.63	1.12	0.08	0.47	0.66	1.18
60	0.09	0.74	1.03	1.91	0.07	0.78	1.10	2.03
70	0.10	1.12	1.59	3.01	0.08	1.17	1.71	3.20
80	0.10	1.69	2.39	4.51	0.09	1.74	2.49	4.77
90	0.09	2.37	3.37	6.44	0.09	2.45	3.51	6.76
100	0.11	3.18	4.55	8.80	0.08	3.27	4.72	9.21
110	0.10	4.23	6.06	11.76	0.08	4.36	6.28	12.28
120	0.11	5.43	7.81	15.29	0.12	5.60	8.10	15.95
130	0.11	6.83	9.85	19.45	0.09	7.04	10.23	20.30
140	0.12	8.52	12.32	24.41	0.10	8.78	12.74	25.47
150	0.12	10.39	15.10	30.24	0.11	10.71	15.65	31.59

Tabela 8.6: Média de translocações para genomas sintéticos com 4 e 5 cromossomos.

	4 cromossomos				**5 cromossomos**			
n	1.5Apr	*AG*	*OBL-AG*	*AM*	1.5Ap	*AG*	*OBL-AG*	*AM*
30	16.76	14.83	14.82	14.81	15.45	13.88	13.88	13.87
40	24.09	20.93	20.91	20.84	23.01	20.29	20.28	20.24
50	31.73	27.71	27.66	27.53	29.62	26.03	26.01	25.94
60	39.76	34.50	34.44	34.21	37.49	32.59	32.56	32.39
70	47.42	41.41	41.31	40.99	44.69	39.01	38.97	38.74
80	55.33	47.88	47.87	47.42	51.40	44.85	44.80	44.48
90	63.35	54.86	54.82	54.27	59.46	51.75	51.69	51.26
100	71.15	61.89	61.80	61.17	66.68	58.40	58.37	57.85
110	78.91	68.85	68.86	68.05	74.77	65.48	65.42	64.81
120	86.29	75.80	75.79	74.85	81.80	71.69	71.66	70.97
130	95.25	83.17	83.15	82.25	89.67	78.76	78.71	77.91
140	102.86	90.47	90.37	89.42	97.29	85.34	85.25	84.39
150	109.71	96.18	96.21	95.10	104.24	91.70	91.64	90.69

segundo com a versão sequencial (AM). O outro experimento, consiste em computar o número de translocações necessárias para transformar um genoma em outro aumentando o número de populações utilizando duas variantes, uma que implementa o modelo de granularidade grossa realizando trocas de indivíduos entre populações AM_{MPT}, e a outra

Tabela 8.7: Tempo de execução (em segundos) dos algoritmos evolutivos e do algoritmo aproximado de raio 1.5+ε para genomas sintéticos com 4 e 5 cromossomos.

	4 cromossomos				**5 cromossomos**			
n	1.5Apr	*AG*	*OBL-AG*	*AM*	1.5Ap	*AG*	*OBL-AG*	*AM*
30	0.10	0.10	0.14	0.28	0.07	0.12	0.16	0.31
40	0.10	0.25	0.27	0.65	0.10	0.26	0.38	0.71
50	0.09	0.50	0.68	1.26	0.09	0.51	0.83	1.45
60	0.10	0.80	1.14	2.13	0.09	0.93	1.40	2.37
70	0.11	1.21	1.75	3.37	0.10	1.46	2.05	3.78
80	0.12	1.81	2.61	5.02	0.10	2.06	2.90	5.44
90	0.11	2.52	3.65	7.10	0.09	2.80	4.01	7.64
100	0.14	3.38	4.94	9.69	0.11	3.80	5.44	10.43
110	0.12	4.51	6.54	12.89	0.10	4.95	7.30	13.81
120	0.13	5.79	8.43	16.71	0.11	6.27	9.06	17.78
130	0.12	7.31	10.69	21.33	0.11	7.96	11.51	22.70
140	0.10	9.13	13.35	26.76	0.13	9.14	13.58	27.69
150	0.13	11.07	16.24	32.86	0.13	11.44	16.88	34.29

em que as populações não realizam qualquer troca de indivíduo, AM_{MPS}.

No primeiro experimento, para genomas de tamanho n, onde $n = 150$ com N cromossomos, para $N \in \{3, \cdots, 8, 9, 10\}$ apenas um genoma foi gerado, e então ambos AM_F e AM foram executados 10 vezes, tal que a média do tempo destas 10 execuções foi tomada como resultado para os respectivos algoritmos. O número de processos P utilizados na versão paralela chegou a 24 no total, com $P \in \{3, 4, \cdots, 23, 24\}$. O número de processos foi tomado a partir de 3, uma vez que, como algoritmo paralelo utiliza a topologia mestre-escravo, com apenas 2 processos, teríamos unicamente um processo encarregado de computar a função de aptidão. Resultados são apresentados nas Tabelas 8.10, 8.11, 8.12 e 8.13 (o tempo é dado em segundos). O tempo e quantidade de indivíduos processados por segundo para o AM é apresentado na legenda de cada tabela.

No segundo experimento, o número de processos foi fixado como sendo 24, e grupos de 100 genomas foram gerados. Cada grupo contém genomas com n genes, para $n \in \{100, 110, 120, 130, 140, 150\}$, possuindo N cromossomos, para $N \in \{3, 4, 5\}$. É importante enfatizar que cada grupo não diversifica a quantidade de genes e cromossomos, ou seja, geram-se grupos a partir do produto cartesiano de n x N. Então para cada genoma em cada um destes grupos, AM, AM_{MPT} e AM_{MPS} foram executados 10 vezes e a média destas 10 execuções foi tomada como resultado para cada um dos algoritmos envolvidos. Esta média representa o resultado para um dado genoma. Além disso, a média dos resultados e dos tempos de execução de todos os conjuntos de 100 genomas de igual

Tabela 8.8: Resultados para *AG*, *AM* e *OBL-AG* utilizando genomas *Benchmarks*.

	Média			**Mediana**		
	AG	AM	OBL-AG	AG	AM	OBL-AG
GL150C2	102.52	**101.58**	102.48	102.50	**102.00**	**102.00**
GL150C3	109.54	**108.82**	109.40	**109.00**	**109.00**	109.50
GL150C4	100.94	**100.04**	101.08	101.00	**100.00**	101.00
GL150C5	96.92	**96.32**	96.82	97.00	**96.00**	97.00
GL150C6	84.80	**84.06**	84.58	85.00	**84.00**	**84.00**
GL150C7	91.08	**90.38**	91.14	91.00	**90.00**	91.00
GL150C8	77.52	**76.88**	77.46	77.50	**77.00**	78.00
GL150C9	78.80	**78.22**	78.74	79.00	**78.00**	79.00
GL150C10	77.52	**77.10**	77.48	**77.00**	**77.00**	**77.00**

Tabela 8.9: Resultados para *AG*, *AM* e *OBL-AG* utilizando genomas *Benchmarks*.

	Mínimo			**Máximo**		
	AG	AM	OBL-AG	AG	AM	OBL-AG
GL150C2	101.00	**100.00**	**100.00**	106.00	**104.00**	106.00
GL150C3	108.00	**107.00**	**107.00**	**111.00**	**111.00**	112.00
GL150C4	99.00	**98.00**	99.00	104.00	**102.00**	104.00
GL150C5	**95.00**	**95.00**	**95.00**	99.00	**98.00**	99.00
GL150C6	**84.00**	**84.00**	**84.00**	87.00	**85.00**	86.00
GL150C7	**90.00**	**90.00**	**90.00**	93.00	**92.00**	94.00
GL150C8	**76.00**	**76.00**	**76.00**	79.00	**78.00**	**78.00**
GL150C9	**78.00**	**78.00**	**78.00**	81.00	**79.00**	81.00
GL150C10	**77.00**	**77.00**	**77.00**	81.00	**78.00**	79.00

tamanho foram calculados. Os resultados do segundo experimento são apresentados nas Tabelas 8.14, 8.15 e 8.16, lembrando que o tempo e o resultado médio apresentado para cada algoritmo é o tempo e o resultado médio respectivamente para cada grupo contendo 100 genomas de n genes.

Tabela 8.10: *AM* obteve tempo de 29.14 segundos com 1111 indivíduos processados por segundo (p/s) com 3 cromossomos, e tempo de 29.98 com 10860 indivíduos p/s para 4 cromossomos.

	3 crom.				**4 crom.**			
N. de Processos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos
3	16.43	1.77	0.59	20542	17.63	1.70	0.57	20035
4	11.83	2.46	0.62	30975	12.40	2.42	0.61	29163
5	9.83	2.96	0.59	38880	10.34	2.90	0.58	36979
6	8.50	3.43	0.57	45041	8.90	3.37	0.56	45199
7	7.54	3.86	0.55	52775	8.05	3.72	0.53	51219
8	6.99	4.17	0.52	62003	7.13	4.20	0.52	57846
9	6.23	4.68	0.52	63056	6.56	4.57	0.51	61128
10	5.96	4.89	0.49	71227	6.09	4.92	0.49	68865
11	5.88	4.96	0.45	72950	6.01	4.99	0.45	70066
12	5.34	5.46	0.46	78852	5.51	5.44	0.45	78850
13	5.59	5.21	0.40	79076	5.58	5.37	0.41	78128
14	5.36	5.44	0.39	78481	5.68	5.28	0.38	77551
15	5.57	5.23	0.35	76591	5.81	5.16	0.34	74411
16	5.59	5.21	0.33	75056	5.91	5.07	0.32	70035
17	5.60	5.20	0.31	74840	5.70	5.26	0.31	74982
18	5.51	5.29	0.29	75198	5.58	5.37	0.30	79224
19	5.42	5.38	0.28	78983	5.78	5.19	0.27	70191
20	5.49	5.31	0.27	78072	5.59	5.36	0.27	77050
21	5.39	5.41	0.26	79334	5.62	5.33	0.25	75203
22	5.61	5.19	0.24	75821	5.61	5.34	0.24	76749
23	5.74	5.08	0.22	73264	5.62	5.33	0.23	76248
24	5.72	5.09	0.21	74850	5.77	5.20	0.22	70134

Tabela 8.11: *AM* obteve tempo de 31.53 segundos com 10542 indivíduos processados por segundo (p/s) com 5 cromossomos, e tempo de 32.60 com 10269 indivíduos p/s para 6 cromossomos.

	5 crom.				**6 crom.**			
N. de Processos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos
3	18.48	1.71	0.57	19290	19.26	1.69	0.56	18988
4	12.95	2.43	0.61	29253	13.45	2.42	0.61	27693
5	10.67	2.96	0.59	35843	11.32	2.88	0.58	33690
6	9.28	3.40	0.57	43098	9.78	3.33	0.56	40159
7	8.35	3.78	0.54	50354	8.78	3.71	0.53	46280
8	7.43	4.24	0.53	55564	7.71	4.23	0.53	52958
9	6.76	4.66	0.52	64090	7.21	4.52	0.50	55194
10	6.23	5.06	0.51	65194	6.75	4.83	0.48	60956
11	6.20	5.09	0.46	73387	6.56	4.97	0.45	66070
12	5.76	5.47	0.46	79680	6.12	5.33	0.44	73110
13	5.93	5.32	0.41	74214	6.46	5.05	0.39	71453
14	5.87	5.37	0.38	75625	6.48	5.03	0.36	70154
15	5.86	5.38	0.36	75794	6.52	5.00	0.33	69802
16	5.77	5.46	0.34	77537	6.49	5.02	0.31	68831
17	5.76	5.47	0.32	79420	6.07	5.37	0.32	67009
18	5.86	5.38	0.30	75879	6.32	5.16	0.29	71930
19	5.78	5.46	0.29	78020	6.17	5.28	0.28	72085
20	5.80	5.44	0.27	77282	6.15	5.30	0.27	72719
21	5.82	5.42	0.26	76691	6.53	4.99	0.24	70222
22	5.79	5.45	0.25	77761	6.23	5.23	0.24	71162
23	5.85	5.39	0.23	76102	6.25	5.22	0.23	68281
24	6.12	5.15	0.21	74025	6.13	5.32	0.22	67678

Tabela 8.12: *AM* obteve tempo de 33.92 segundos com 10106 indivíduos processados por segundo (p/s) com 7 cromossomos, e tempo de 35.61 com 9913 indivíduos p/s para 8 cromossomos.

	7 crom.				**8 crom.**			
N. de Processos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos
3	19.82	1.71	0.57	18250	20.37	1.75	0.58	18132
4	14.30	2.37	0.59	26206	14.51	2.45	0.61	26750
5	11.74	2.89	0.58	33438	12.20	2.92	0.58	32969
6	10.32	3.29	0.55	39730	12.19	2.92	0.49	38474
7	9.15	3.71	0.53	46361	9.39	3.79	0.54	43614
8	8.44	4.02	0.50	50743	8.61	4.14	0.52	50760
9	7.83	4.33	0.48	57563	7.99	4.46	0.50	55320
10	7.27	4.67	0.47	59846	7.40	4.81	0.48	62015
11	7.02	4.83	0.44	64037	7.19	4.95	0.45	62814
12	6.45	5.26	0.44	74023	6.38	5.58	0.47	70654
13	6.83	4.97	0.38	74397	6.74	5.28	0.41	67211
14	6.97	4.87	0.35	71616	6.90	5.16	0.37	66217
15	6.65	5.10	0.34	72613	6.89	5.17	0.34	66345
16	6.68	5.08	0.32	71955	6.82	5.22	0.33	66484
17	6.89	4.92	0.29	65617	6.64	5.36	0.32	67869
18	6.55	5.18	0.29	73731	6.55	5.44	0.30	68110
19	6.95	4.88	0.26	71177	6.42	5.55	0.29	69595
20	6.75	5.03	0.25	68321	6.68	5.33	0.27	67595
21	7.00	4.85	0.23	70938	6.39	5.57	0.27	70313
22	6.76	5.02	0.23	72071	6.51	5.47	0.25	69876
23	6.53	5.19	0.23	73744	6.55	5.44	0.24	69547
24	7.01	4.84	0.20	66362	7.04	5.06	0.21	65606

Tabela 8.13: *AM* obteve tempo de 36.86 segundos com 9561 indivíduos processados por segundo (p/s) com 9 cromossomos, e tempo de 39.27 com 9345 indivíduos p/s para 10 cromossomos.

	9 crom.				**10 crom.**			
N. de Processos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos	Tempo AM_F	Speedup	Eficiência	N.Indivíduos
3	21.32	1.73	0.58	17744	22.80	1.72	0.57	16919
4	14.84	2.48	0.62	25556	16.43	2.39	0.60	24207
5	12.68	2.91	0.58	32203	14.06	2.79	0.56	29833
6	10.76	3.43	0.57	37314	12.19	3.22	0.54	35033
7	9.50	3.88	0.55	41680	10.86	3.62	0.52	40234
8	9.03	4.08	0.51	48389	10.02	3.92	0.49	43779
9	8.22	4.48	0.50	54636	9.17	4.28	0.48	47253
10	7.80	4.73	0.47	55221	8.78	4.47	0.45	50026
11	7.69	4.79	0.44	60480	8.63	4.55	0.41	52858
12	7.09	5.20	0.43	63831	8.12	4.84	0.40	54585
13	7.56	4.88	0.38	62726	8.36	4.70	0.36	55401
14	7.67	4.81	034	62345	8.26	4.75	0.34	54477
15	7.65	4.82	0.32	62499	8.34	4.71	0.31	53594
16	7.40	4.98	0.31	62591	8.60	4.57	0.29	53868
17	7.10	5.19	0.31	63294	8.48	4.63	0.27	52319
18	7.49	4.92	0.27	62102	8.45	4.65	0.26	52448
19	7.29	5.06	0.27	63088	8.19	4.79	0.25	54052
20	7.53	4.90	0.25	61962	8.34	4.71	0.24	53139
21	7.15	5.16	0.25	63185	8.26	4.75	0.23	53468
22	7.29	5.06	0.23	63135	8.58	4.58	0.21	52550
23	7.48	4.93	0.21	62643	8.61	4.56	0.20	52298
24	7.30	5.05	0.21	62840	8.53	4.60	0.19	52652

Tabela 8.14: Número médio de translocações utilizando um conjunto de 100 genomas para cada n de 3 cromossomos.

	3 crom.					
n	Média AM	Tempo AM	Média AM_{MPT}	Tempo AM_{MPT}	Média AM_{MPS}	Tempo AM_{MPS}
100	64.30	8.26	64.81	21.02	63.79	20.80
110	71.43	11.05	72.10	27.46	70.74	26.76
120	78.62	14.46	79.50	35.19	77.73	34.94
130	86.45	18.49	87.44	44.60	85.33	43.79
140	94.15	23.19	95.45	54.74	92.89	54.42
150	100.55	28.75	102.08	66.57	99.16	66.21

Tabela 8.15: Número médio de translocações utilizando um conjunto de 100 genomas para cada n de 4 cromossomos.

	4 crom.					
n	Média AM	Tempo AM	Média AM_{MPT}	Tempo AM_{MPT}	Média AM_{MPS}	Tempo AM_{MPS}
100	61.24	8.64	61.53	21.96	60.72	21.95
110	68.07	11.55	68.64	28.83	67.49	28.73
120	75.00	15.06	75.73	36.83	74.30	36.76
130	81.53	19.32	82.41	46.57	80.76	46.48
140	88.85	24.35	90.12	57.44	87.90	57.41
150	95.42	30.16	96.83	69.79	94.40	69.55

Tabela 8.16: Número médio de translocações utilizando um conjunto de 100 genomas para cada n de 5 cromossomos. O tempo é dado em segundos.

	5 crom.					
n	Média AM	Tempo AM	Média AM_{MPT}	Tempo AM_{MPT}	Média AM_{MPS}	Tempo AM_{MPS}
100	57.95	9.15	58.16	23.25	57.69	23.21
110	64.78	13.17	65.12	30.38	64.35	30.29
120	72.15	16.87	72.62	38.74	71.62	38.75
130	78.48	20.33	79.25	48.70	77.82	48.81
140	85.25	25.60	86.23	59.92	84.58	59.81
150	91.89	31.53	93.07	73.08	91.20	72.94

8.2.3 Experimentos com Genomas Baseados em Dados Biológicos

Três diferentes espécies de mamíferos foram escolhidas: gato, rato e humano. Seus genomas foram tomados de [7], considerando unicamente o material genético em comum. Modificações nestes genomas foram feitas para satisfazer a Propriedade 1 (Seção 2.1.4 no Capítulo 2): genes 82, 83 e 88 foram removidos no mapeamento original proposto em [7]; além disso, genes auxiliares foram adicionados no extremos de cada cromossomo para cada um dos genoma em ordem para obter genomas *co-caudais*, no fim cada um dos genomas permaneceu com 146 genes e 18 cromossomos.

Inicialmente, o genoma humano foi fixado como o genoma identidade e, correspondentes mapeamentos de genes sobre os genomas do gato e do rato foram executados para dar origem aos genomas Humano-Gato e Humano-Rato. Em seguida, o genoma do gato foi fixado como o genoma identidade e o correspondente mapeamento de genes foi aplicado sobre o genoma do rato para gerar o genoma Gato-Rato. Com o método descrito acima é possível mensurar a distância evolutiva entre humano $\times$ gato, humano $\times$ rato, gato $\times$ rato.

Os experimentos foram conduzidos como segue: *AG*, *AM* e *OBL-AG* foram executados dez vezes para cada um destes genomas (e o algoritmo aproximado de raio $1.5+\varepsilon$ uma única vez), e a média das dez execuções foi tomada como resultado. Veja Tabela 8.17.

Tabela 8.17: Resultados utilizando dados biológicos com tempo de execução fornecido em segundos.

Genoma	1.5-Ap	1.5-Ap-Tempo	*AG*	*AG*-Tempo	AM	AM-Tempo	OBL-AG	OBL-AG-Tempo
Humano-Gato	43	2.34	43	17.61	43	54.78	43	30.10
Humano-Rato	53	3.37	51	18.16	51	56.86	51	31.80
Gato-Rato	53	2.71	50	17.82	50	55.04	50	30.79

8.3 Discussão

Algumas considerações são necessárias antes de discutir os resultados. No procedimento de geração de genomas sintéticos (Algoritmo 16), aplicam-se reversões sobre um único cromossomo e, unicamente translocações *Prefixo-Prefixo* entre os cromossomos. Ao incluir reversões, foi possível obter instâncias do problema bem mais complexa. Translocações *Prefixo-Sufixo* não foram consideradas, pois uma operação *Prefixo-Sufixo* pode ser simulada combinando uma reversão seguida de uma operação *Prefixo-Prefixo* [5].

É importante enfatizar que o algoritmo proposto em [5], o qual é usado como função de aptidão, encontrava-se implementado em `Java` pelos seus autores como uma ferramenta inclusa no framework `UniMoG` [25]. Esta implementação foi traduzida em `C` neste trabalho. Afim de manter o padrão de qualidade, exaustivos testes foram executados para validar os resultados desta implementação comparando os resultados de ambas as versões (`Java` e `C`).

Afim de facilitar a leitura, a discussão será particionada em duas partes. A primeira parte será restrita aos algoritmos evolutivos implementados de forma serial, e a segunda parte será relacionada aos algoritmos paralelos.

Algoritmos Seriais

Os resultados apresentados nas Tabelas 8.4, 8.6 e 8.17 também são apresentados em [43]. Ao olhar para os resultados em ambas as tabelas é notório que o algoritmo aproximado apresenta pouquíssimas variações no tempo de execução, mesmo para genomas contendo 150 genes, isto se deve ao fato que as soluções providas pelo algoritmo aproximado são baseadas nos cálculos dos 2-ciclos e os genomas gerados tanto dos dados biológicos quanto do Algoritmo 16 possuem um baixo número de 2-ciclos. Logo, a execução do algoritmo aproximado torna-se muito rápida.

A partir dos experimentos usando grupos de genomas sintéticos (Tabelas 8.5 e 8.7) pode-se observar que quando o número de genes é incrementado, os algoritmos evolutivos tendem a aumentar seus respectivos tempos de execução continuamente "como esperado". Isto pode ser claramente observado na Figura 8.1, para entradas com 2 cromossomos. Outro fator que influência o tempo, é o número de cromossomos, o tempo de execução

também sofre aumento quando o número de cromossomos é incrementado. Isto se deve ao fato que, tendo mais cromossomos, o processo de construção da floresta pertencente ao algoritmo utilizado como função de aptidão se torna mais complexa (identificações de *SPs* e *MinSPs* nos cromossomos).

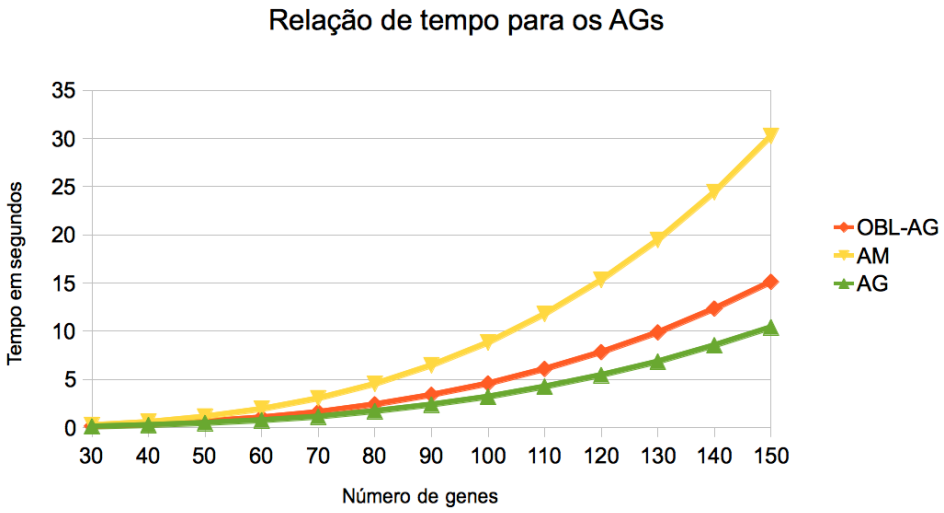

Figura 8.1: Tempos de execução dos algoritmos evolutivos considerando diferentes quantidades de genes para 2 cromossomos.

Ao observar o tempo necessário para computar genomas grandes (contendo 150 genes), os experimentos mostram que *AM* e *OBL-AG* tomam aproximadamente 184% e 47% do tempo de execução do *AG*, respectivamente. Todavia, tanto *AM* quanto *OBL-AG* são de interesse prático uma vez que o primeiro algoritmo leva pouco menos 1 minuto para computar genomas com 150 genes e 10 cromossomos em um modesto OS X de 3,1 GHz com processador Intel Core i5 de 4 cores sem *Hyper-threading*.

A partir das Tabelas 8.4 e 8.6, pode-se concluir que o *AG* computa em média melhores resultados que aqueles providos pelo algoritmo aproximado. Além disso, para genomas contendo 50 genes ou mais, o *AG* fornece resultados aproximadamente 12% melhores que o algoritmo aproximado (menos translocações são usadas para transformar um genoma em outro). Para as versões do *AG* utilizando métodos de otimização, pode-se observar que *AM* fornece melhores resultados comparados tanto com o *AG* (aproximadamente 1%) quanto com o *OBL-AG* (cerca de 0.87%), e o *OBL-AG* apresenta resultados muito semelhantes aos providos pelo *AG* (melhoria de aproximadamente 0.09%).

Ao analisar os experimentos utilizando específicos genomas (Tabelas 8.8 e 8.9), o *AM* provê melhores resultados quando comparado com *AG* e *OBL-AG* respectivamente. Ao considerar as mesmas medidas, o *OBL-AG* apresenta uma pequena melhoria comparado ao AG, e para certas instâncias os resultados são os mesmos.

A partir dos experimentos usando dados biológicos, pode-se observar que *AG*, *OBL-AG* e *AM* apresentam os mesmos resultados. Isto é explicado porque estes genomas são instâncias muito fáceis de resolverem, uma vez que, gato, humano e rato, possuem uma sequência genética muito similar. Além disso, os piores resultados são aqueles computados pelo algoritmo aproximado, o qual é utilizado como mecanismo de controle mínimo de qualidade.

Algoritmos Paralelos

Nos primeiros ciclos da pequisa onde foram propostos algoritmos genéticos para lidar com o problema *DTS* ([42] e [43]) não nos preocupamos com o tempo de execução, e sim na precisão das soluções fornecidas por estes algoritmos. Esta foi a principal razão para propor a implementação do Algoritmo 12 (AM_F). Com relação aos resultados dos experimentos, pode-se concluir que utilizando o modelo de paralelização global, onde os processos escravos realizam o cálculo da função de aptidão, reduz-se substancialmente o tempo de execução do *AM*, mantendo a precisão dos resultados. Como pode ser visto nas respectivas Tabelas 8.10, 8.11, 8.12 e 8.13, é notável que, utilizando 12 processos se tem os melhores tempos de execução (aproximadamente 5x mais rápido), mesmo a máquina tendo condições de utilizar até 24 processos simultaneamente. A explicação mais plausível se deve ao fato que o tempo de execução de cada genoma é baixo (entre 30 e 40 segundos) para que, ao aumentar a quantidade de processos, continue obtendo ganho de desempenho, uma vez que se utiliza *MPI* para a implementação, ao aumentar o número de processos, tem-se o aumento do número de troca de mensagens e eventualmente, o aumento do gargalo. Além disso, a técnica de *Hyper-threading* não oferece desempenho igual a uma máquina que contenha o mesmo número de núcleos físicos, uma vez que tal técnica aproveita os ciclos ociosos de cada núcleo real para realizar processamento, isto dá um ganho de desempenho entre 20% a 30%, o que acaba não compensando o custo do gargalo nas trocas de mensagens ao colocar mais de 12 processos (a máquina utilizada contém 12 núcleos reais). Outro ponto a destacar é que ao utilizar mais de 12 processos não se obtém incremento no *speedup*, porém estes valores obtidos são superiores aos demais *speedups* utilizando um número de processos inferior a 12.

Ao observar a eficiência do AM_F, pode-se concluir que a medida que se incrementa o número de processos nos experimentos, salve algumas exceções para 3 e 4 processos, onde a eficiência aumenta ou para alguns outros casos onde se mantém, a eficiência decresce em aproximadamente 3%. Com relação ao número de indivíduos processados por segundo, os experimentos mostram que o maior número é obtido justamente com 12 processos o que de certa forma é normal visto que os melhores *speedups* são alcançados com 12 processos

simultâneos. Para mais de 12 processos, o número de indivíduos processados por segundo é superior em todos os experimentos realizados com o número de processos inferior a 11.

Os resultados obtidos pelo AM foram relatados para superar a precisão das soluções providas pelo algoritmo genético proposto em [42] bem como os resultados fornecidos pelo algoritmo aproximado de raio $1.5+\varepsilon$. Assim, o segundo experimento foi realizado com AM_{MPT} e AM_{MPS} (versões em paralelo do AM), com o intuito de aumentar a o número de indivíduos e consequentemente melhorar a precisão dos resultados.

A partir das Tabelas 8.15 e 8.16 pode-se observar que o AM_{MPS} sobrepõem a qualidade dos resultados providos por ambos algoritmos AM_{MPT} e AM. Comparado com os resultados do AM isto já era esperado, visto que AM_{MPS} faz uso de uma amostra do espaço de busca bem mais extensa (24 vezes maior). Entretanto, não era esperado que o AM_{MPT} fornecesse resultados inferiores ao AM, visto que o espaço de busca da versão em paralelo é bem mais extenso que o da versão sequencial, contudo na literatura, os grandes autores que lidam com algoritmo genético deixam claro que a estratégia de elitização tende a prover indivíduos mais aptos, todavia, há o risco de, ao manter uma população totalmente elitizada, possa-se perder genes que poderiam estar contidos em indivíduos menos aptos, que por sua vez poderiam ser usados para convergir a resultados mais expressivos. Outra observação, é que foi utilizado no AM_{MPT} os mesmos valores dos parâmetros definidos para o AM, em uma eventual continuação deste trabalho é desejável definir os parâmetros utilizados pelo AM_{MPT}, na tentativa de melhorar os resultados providos pelo mesmo. Além disso, é importante enfatizar que mesmo com a paralelização do AM há apenas uma pequena vantagem (aproximadamente 1.07%) do AM_{MPS} para com o AM em se tratando da qualidade das soluções providas por ambos com genomas de até 150 genes para 3, 4 e 5 cromossomos. Contudo, ao olhar os resultados na Figura 8.2 (diferença entre o resultados providos pelo AM serial e o AM_{MPS}) é possível constatar que a medida que se incrementa a quantidade de genes e consequentemente a amostra do espaço de busca do problema se torna maior, a precisão dos resultados providos pelo AM_{MPS} se tornam cada vez melhores. Em relação ao tempo, tem-se que, além do AM_{MPT} fornecer os piores resultados, ele também consome mais tempo de execução, cerca de 242% e 0.3% mais tempo de processamento quando comparado com AM e AM_{MPS} respectivamente. Como considerações finais, foi decepcionante o AM_{MPT} ter sido um fracasso, visto que se tinha muitas expectativas em relação ao seu desempenho. O AM_{MPS} fornecer resultados melhores que o AM era óbvio, já que, uma execução de si, equivale a realizar 24 experimentos diferentes com o AM. Todavia, há um ponto positivo que deve ser apontado com relação ao AM_{MPS}, o tempo de execução destes 24 experimentos simultâneos tem um tempo bem mais baixo quando comparado com o tempo de realizar os mesmos 24 experimentos sequencialmente (aproximadamente 916%), ou mesmo fazendo uso do MA_F com

12 processos (aproximadamente 89%). Os dados utilizados para calcular a porcentagem de ganho de tempo do AM_{MPS} foram tomados das Tabelas 8.10, 8.11, 8.14, 8.15 e 8.15.

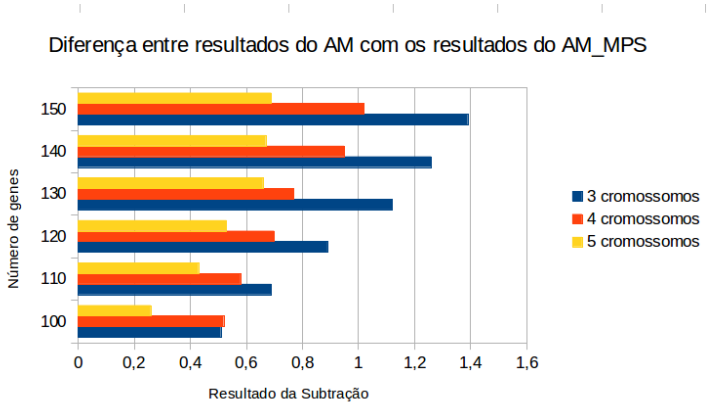

Figura 8.2: Diferença entre os resultados providos pelo AM e os resultados fornecidos pelo AM_{MPS}.

8.3.1 Análise Estatística

A avaliação estatística tem sido considerada uma parte essencial na validação da qualidade dos resultados providos por algoritmos baseados em aprendizagem. A ideia consiste em a partir de um conjunto de entradas fornecidas para ambos algoritmos envolvidos, avaliar estatisticamente se há alguma diferença nos resultados providos por tais algoritmos.

Aqui, a análise estatística tem como objetivo verificar se existe alguma diferença nos resultados providos pelo *OBL-AG* e AM quando comparado com os resultados do AG; e as soluções do melhor algoritmo em paralelo AM_{MPS} com relação as soluções do AM e AM_{MPT}.

As amostras utilizadas para os testes são genomas sem sinal denotados como *benchmarks* propostos em [43] os quais também foram utilizados e apresentados na Seção 8.2.2. A análise é realizada seguindo a metologia discutida em [16, 17, 36]; que é, inicialmente, aplicar o teste de *Kolgomorov-Smirnov* para determinar a distribuição não normal das amostras (50 execuções do AG, *OBL-AG*, AM, AM_{MPS} e AM_{MPT}). Então, aplica-se o teste de *Wilcoxon* considerando um nível de confiabilidade de 95%, o qual representa uma significância de 5% ou valor inferior a 0.05; para comparar a mediana de pares de amostras (genomas) entre dois algoritmos. Os resultados do teste de *Wilcoxon* são mostrados respectivamente nas Tabelas 8.18, 8.19 e 8.20. Duas colunas são incluídas: uma com a mediana e a outra com o resultado do teste estatístico, onde símbolo "$s+$" indica que dois algoritmos apresentam diferenças nos resultados e "$s-$" indica que não há qualquer

diferença estatística nos resultados providos (valor > 0.05) entre eles. A seguir, ao se referir como desempenho superior e inferior, tem-se que a mediana possui um valor menor e maior, respectivamente.

Avaliando primeiramente os algoritmos sequenciais, pode-se observar na Tabela 8.18 que em muitos casos, o *AM* apresenta desempenho superior com relação ao *AG* e *OBL-AG*. A partir da Tabela 8.18, pode-se observar que o *OBL-AG* possui desempenho diferente somente com relação ao *AM*, e nenhuma diferença quando comparado com o *AG*; o que significa que estatisticamente as soluções providas pelo *OBL-AG* não possuem nenhuma significância estatística quando comparadas com as soluções fornecidas pelo *AG*. Além disso, é importante destacar que mesmo o *OBL-AG* apresentando desempenho diferente do *AM*, em muitos casos, o *OBL-AG* possui desempenho inferior.

Para os algoritmos paralelos, foram realizados testes estatísticos somente com relação ao AM_{MPS} (sem troca de indivíduos), visto que o AM_{MPT} (com troca de indivíduos) apresenta resultados piores que o *AM*. A partir da Tabela 8.20 pode se observar que na maioria das amostras os resultados providos pelo AM_{MPS} são diferentes que aqueles fornecidos tanto pelo *AM* e AM_{MPT}. Em se tratando de desempenho, o AM_{MPS} supera os resultados do *AM* na maioria dos experimentos. Em relação ao AM_{MPT}, é ainda mais evidente, o AM_{MPS} apresenta melhor desempenho para todo o conjunto de amostras (valor da mediana é menor).

Tabela 8.18: *AM* versus outros algoritmos utilizando o teste de *Wilcoxon*.

Genomas	AM	AG		OBL-AG	
GL150C2	102.00	102.50	$s+$	102.00	$s+$
GL150C3	109.00	109.00	$s+$	109.50	$s+$
GL150C4	100.00	101.00	$s+$	101.00	$s+$
GL150C5	96.00	97.00	$s+$	97.00	$s+$
GL150C6	84.00	85.00	$s+$	84.00	$s+$
GL150C7	90.00	91.00	$s+$	91.00	$s+$
GL150C8	77.00	77.50	$s+$	78.00	$s+$
GL150C9	78.00	79.00	$s+$	79.00	$s+$
GL150C10	77.00	77.00	$s+$	77.00	$s+$

Tabela 8.19: *OBL-AG* versus outros algoritmos utilizando o teste de *Wilcoxon*.

Genomas	OBL-AG	AG		AM	
GL150C2	102.00	102.50	$s-$	102.00	$s+$
GL150C3	109.50	109.00	$s-$	109.00	$s+$
GL150C4	101.00	101.00	$s-$	100.00	$s+$
GL150C5	97.00	97.00	$s-$	96.00	$s+$
GL150C6	84.00	85.00	$s-$	84.00	$s+$
GL150C7	91.00	91.00	$s-$	90.00	$s+$
GL150C8	78.00	77.50	$s-$	77.00	$s+$
GL150C9	79.00	79.00	$s-$	78.00	$s+$
GL150C10	77.00	77.00	$s-$	77.00	$s+$

Tabela 8.20: AM_{MPS} versus outros algoritmos utilizando o teste de *Wilcoxon*.

Genomas	AM_{MPS}	AM		AM_{MPT}	
L150C2	100.00	102.00	$s+$	104.00	$s+$
L150C3	106.00	108.00	$s+$	109.00	$s+$
L150C4	99.00	100.00	$s+$	103.00	$s+$
L150C5	95.00	96.00	$s+$	97.00	$s+$
L150C6	84.00	84.00	$s-$	86.00	$s+$
L150C7	90.00	90.00	$s+$	92.00	$s+$
L150C8	76.00	77.00	$s+$	77.00	$s+$
L150C9	78.00	78.00	$s+$	78.00	$s-$
L150C10	77.00	77.00	$s-$	78.00	$s+$

Capítulo 9

Conclusão e Trabalhos Futuros

Neste trabalho foi proposto um algoritmo genético (AG), para resolver o problema de distância evolutiva entre genomas sem sinal por translocações. Este AG atua em uma população de genomas com sinal gerados a partir de um genoma fornecido como entrada e após cada geração a população evolui com a inserção de descendentes contendo as menores distâncias de translocação. De fato, a ideia chave do AG, está no uso da função de aptidão baseada no cálculo de soluções exatas para genomas com sinal, usando o algoritmo de tempo linear no número de genes do genoma proposto em [5]. Em seguida, dois algoritmos evolucionários híbridos para o problema de distância de translocação entre genomas sem sinal foram propostos, baseando-se no AG. As principais características destes novos algoritmos, chamados de AM e $OBL\text{-}AG$, é a aplicação das técnicas de otimização, de: memético e aprendizagem baseada em oposição (*OBL-Opposition-based Learning*), respectivamente. Tais técnicas são acrescentadas ao iniciar a população, no reinício da população sempre que o limite de entropia é alcançado, e como um novo estágio após o ciclos de reprodução. Em ambos algoritmos, a convergência da população é controlada utilizando um método conhecido como entropia de *Shannon*.

Para o desenvolvimento do método e execução de testes experimentais que permitiram verificar a qualidade das respostas fornecidas foi necessário realizar o seguinte.

- Implementação do algoritmo aproximado de raio $1.5+\varepsilon$ proposto em [14]. Dessa forma foi realizado um minucioso estudo teórico a fim de alcançar um algoritmo que computasse corretamente as soluções dentro do raio proposto.
- Na literatura há poucos genomas baseados em dados biológicos contendo mais de um cromossomo. Assim, foi implementado um algoritmo para a geração de genomas sintéticos baseados em translocações e reversões para que testes mais específicos pudessem ser aplicados.

- Dos genomas baseados em dados biológicos com mais de um cromossomo que se encontram mapeados, escolheram-se três (gato, rato e humano). Todavia, para mensurar a distância evolutiva entre essas espécies utilizando translocações foi necessário realizar um mapeamento sobre o original [7], para satisfazer a propriedade, onde dados dois genomas os mesmos devem possuir o mesmo conjunto de genes finais dos seus respectivos cromossomos.

- Para o cálculo da função de aptidão do AG foi utilizado o código fonte do algoritmo de tempo linear para o problema de distância de translocações entre genomas com sinal, disponibilizado em [25]. Tal algoritmo encontrava-se implementado em `Java` como parte de um framework (`UniMoG`) utilizado para computar distâncias evolutivas entre espécies através de várias métricas. Deste modo, houve a necessidade de isolar a parte referente à distância de translocação entre genomas com sinal e abstrair para uma implementação em `C`.

- Duas versões em paralelo do AM foram propostas para resolver o problema de distância de translocação entre genomas sem sinal. A primeira AM_F busca melhor desempenho de execução conservando a precisão dos resultados fazendo uso do modelo global (mestre-escravo), onde o processo mestre mantém os estágios do AM sobre uma única população, enquanto que os processos escravos ficam encarregados de computar o valor de aptidão de cada indivíduo. A segunda versão busca incrementar a precisão dos resultados utilizando múltiplas populações, onde cada processo mantém uma instância do AM com sua população. Nesta versão, duas variantes foram propostas, uma com troca dos melhores indivíduos entre as populações (AM_{MPT}), e uma sem quaisquer trocas de indivíduos entre as populações (AM_{MPS}).

- Para obter um máximo desempenho dos AGs, os seguintes parâmetros foram ajustados via experimentos.

  - AG: probabilidade de cruzamento, probabilidade de mutação, número de pontos de cruzamento, % da população para seleção e % da população para substituição.

  - OBL-AG: utiliza os parâmetros do AG, incorporando os parâmetros % da população para aplicar OBL, % de preservação da população ao reiniciar e limite de entropia.

- AM, AM_F e AM_{MPS}: utilizam os parâmetros do AG, incorporando os parâmetros % da população para busca local, % de preservação da população ao reiniciar e limite de entropia.

- AM_{MPT}: utiliza os parâmetros do AM, incorporando o parâmetro % troca de indivíduos.

Experimentos e Resultados:

- Experimentos com genomas sintéticos utilizando algoritmos seriais:
  - O AG fornece resultados com precisão de cerca de 12 % melhor que o algoritmo aproximado de raio $1.5+\varepsilon$.
  - O AM apresenta melhores precisões nos resultados comparados com AG e *OBL-AG* respectivamente, e o *OBL-AG* apresenta soluções muito semelhantes quando comparado com AG.
  - Uma análise estatística para os algoritmos evolutivos AG, *OBL-AG*, AM foi realizada utilizando o teste de *Wilcoxon*, onde são considerados como amostras genomas *benchmark* propostos em [43]. Desta análise, concluiu-se que AM apresenta melhor desempenho nos resultados comparados com AG e *OBL-AG* respectivamente, e que os resultados providos pelo *OBL-AG* não apresentam nenhuma diferença significativa de desempenho quando comparados com os resultados fornecidos pelo AG.

- Experimentos com genomas sintéticos utilizando algoritmos paralelos:
  - AM_F reduz o tempo de execução do AM em aproximadamente 5 vezes.
  - Em se tratando das variantes com múltiplas populações, o AM_{MPS} obtém melhora na precisão dos resultados quando comparado com AM e o AM_{MPT}. Todavia, a versão na qual há troca de indivíduos AM_{MPT} não fornece qualquer melhora na precisão dos resultados.
  - Através de uma análise estatística baseada no teste de *Wilcoxon*, considerando como amostras, os genomas *benchmark* propostos em [43], para o AM_{MPS}, AM_{MPT} e AM; tem-se que o AM_{MPS} provê tanto resultados diferentes quanto melhores que o AM_{MPT} e AM. Quando comparado com o AM, os resultados (valor da mediana) providos pelo AM_{MPS} superam em 77.7% do conjunto de amostras adotado e com relação ao AM_{MPT} os resultados (valor da mediana) superam em 88.8%.

- Experimentos com genomas biológicos utilizando algoritmos seriais:
  - três genomas baseados em dados biológicos (gato, rato, e humano) foram processados, onde os experimentos usando estes dados mostraram que o AG, AM e $OBL\text{-}AG$ computam os mesmos resultados, em compensação, todos superam os resultados providos pelo algoritmo aproximado de raio $1.5+\varepsilon$.

É necessário enfatizar, que o cálculo da função de aptidão está restrito unicamente a distância de translocação sem necessariamente calcular a sequência de translocações que transforma (ou ordena) um genoma em outro. Do ponto de vista biológico, isto não é um problema, uma vez que não é necessário se ter em mãos a sequência de ordenação para realizar uma reconstrução filogenética; além disso, esta sequência não detêm muita significância, já que, existe mais de uma sequência, a qual fornece uma ordenação ótima.

Como possíveis trabalhos futuros, propõe-se o seguinte:

- Estudar e implementar o algoritmo de raio de aproximação $1.408+\varepsilon$, já que ainda não foi provida uma implementação para este algoritmo.
- Adicionar novos genomas baseados em dados biológicos.
- Estudar possíveis adaptações para aprimorar o AG com a técnica de otimização OBL do tipo-II.
- Estudar possibilidades de aprimorar o AG mesclando ambas as técnicas memético e OBL.
- Estudar outras possibilidades de explorar paralelismo para aprimorar a precisão dos resultados. O fato do algoritmo AM_{MPS} ter fornecido melhores soluções em comparação com o AM, já era esperado, visto que o AM_{MPS} unicamente simula a execução de 24 (número de processos utilizados nos experimentos) instâncias do AM, onde cada uma mantém uma população diferente, o que de certa forma consiste em unicamente aumentar o espaço de amostragem do problema em questão. Com relação ao AM_{MPT}, foi decepcionante os resultados fornecidos pelo mesmo, já que o tamanho do espaço de amostragem é o mesmo considerado no AM_{MPS}, era esperado que pelo menos os resultados pudessem competir com os do AM. Dois pontos passaram despercebidos, e que por falta de tempo não puderam ser reparados, e que por ventura podem explicar o fracasso do AM_{MPS}: **1)** avaliar com mais cuidado a escolha de quais indivíduos são compartilhados e **2)** refazer os testes para definir quais os melhores valores a serem fixados para os parâmetros utilizados pelo algoritmo, uma vez que foram utilizados os mesmos valores tomados pelo AM e o processo de escolha do melhor valor foi realizado unicamente para o novo parâmetro

incorporado ao AM_{MPT}. Assim, tem-se em mente que como uma contribuição mais pertinente perante a comunidade científica, explorar o paralelismo com granularidade grossa, utilizando outros métodos para realizar a troca de indivíduos entre as populações envolvidas e refazer os experimentos para fixar os parâmetros utilizados no AM_{MPT}, possa ser um dos meios de se obter uma precisão nos resultados que pelo menos se equipare com as providas pelo AM_{MPS}.

Referências

[1] F. S. Al-Qunaieer, H. R. Tizhoosh, and S. Rahnamayan. Opposition based computing—a survey. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–7. IEEE, 2010. 75

[2] G. S. Almasi and A. Gottlieb. *Highly parallel computing*. Menlo Park, CA (USA); Benjamin-Cummings Publishing Company, United States, 1988. 80

[3] D. A. Bader, B. M. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Computational Biology*, 8(5):483–491, 2001. 20

[4] A. Bergeron, J. Mixtacki, and J. Stoye. *Combinatorial Pattern Matching: 15th Annual Symposium, CPM 2004, Istanbul, Turkey, July 5-7, 2004. Proceedings*, chapter Reversal Distance without Hurdles and Fortresses, pages 388–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 65, 69

[5] A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *Computational Biology*, 13(2):567–578, 2006. 2, 3, 7, 19, 20, 47, 61, 62, 63, 64, 65, 66, 67, 68, 104, 111

[6] P. Berman and M. Fürer. Approximating maximum independent set in bounded degree graphs. In *Proceedings of the fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'94, pages 365–371, 1994. 33

[7] G. Bourque and P. A. Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome research*, 12(1):26–36, 2002. 103, 112

[8] L. S. Buriol. Algoritmo memético para o problema do caixeiro viajante assimétrico como parte de um framework para algoritmos evolutivos. Master's thesis, Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação, Campinas, SP, 2000. 72

[9] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998. 86

[10] E. Cantu-Paz. *Designing Efficient and Accurate Parallel Genetic Algorithms (Parallel Algorithms)*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1999. AAI9952979. 16

[11] E. Cantu-Paz and D. E. Goldberg. Efficient parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2):221 – 238, 2000. 86

[12] A. Caprara. Sorting by reversals is difficult. In *Proc. of the first annual international conference on Computational molecular biology*, pages 75–83, New York, NY, USA, 1997. ACM. 9, 21, 22, 23, 24

[13] Y. Cui, L. Wang, and D. Zhu. A 1.75-approximation algorithm for unsigned translocation distance. *Computer and System Sciences*, 73(7):1045–1059, 2007. 31, 32, 34, 35, 47

[14] Y. Cui, L. Wang, D. Zhu, and X. 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. 3, 32, 33, 34, 35, 39, 46, 47, 49, 50, 51, 52, 53, 55, 56, 111

[15] R. Dawkins. The selfish gene. 1976. *Revised. Oxford*, 1989. 71, 72

[16] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Machine Learning Research*, 7:1–30, 2006. 108

[17] J. J. Durillo, J. García-Nieto, A. J. Nebro, C. A. Coello, F. Luna, and E. Alba. Multi-objective particle swarm optimizers: An experimental comparison. In *Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization*, EMO, pages 495–509, Berlin, Heidelberg, 2009. Springer-Verlag. 108

[18] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966. 79

[19] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley & Sons, New York, USA, 1st edition, 1966. 12

[20] E. A. Gonçalves. Evolução adaptativa em populações estruturadas. Master's thesis, Universidade Federal Rural de Pernambuco. Departamento de Estatística e Informática, Recife, PE, 2007. 15

[21] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In *International Computer Games Association*, pages 177–183, 1993. 86

[22] M. M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'95, pages 160–169, 1995. 33

[23] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71(1):137–151, 1996. 7, 19, 20, 30, 34, 35, 64

[24] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 581–592. IEEE, 1995. 20

[25] R. Hilker, C. Sickinger, C. N. Pedersen, and J. Stoye. UniMoG—a unifying framework for genomic distance calculation and sorting based on DCJ. *Bioinformatics*, 28(19):2509–2511, 2012. 104, 112

[26] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* University Michigan Press, Michigan, USA, 1975. 12

[27] I. Holyer. The NP-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981. 21

[28] H. Jiang, L. Wang, B. Zhu, and D. Zhu. A (1.408+ ε)-approximation algorithm for sorting unsigned genomes by reciprocal translocations. In *Frontiers in Algorithmics*, pages 128–140. Springer, 2014. 30, 32, 33

[29] R. M. Karp. *Reducibility among combinatorial problems.* Springer, 1972. 32

[30] J. D. Kececioglu and R. Ravi. Of mice and men: algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 95 of *SODA'95*, pages 604–613, 1995. 1, 19, 31

[31] L. G. Latre. *Contribuições à Solução de Problemas de Escalonamento pela Aplicação Conjunta de Computação Evolutiva e Otimização com Restrições.* PhD thesis, Universidade Estadual de Campinas, 2000. 72

[32] R. Linden. *Algoritmos genéticos (2a edição).* Brasport, Rio de Janeiro, Brasil, 2008. 12

[33] D. Miles, B. Leback, and D. Norton. Optimizing application performance on cray systems with PGI compilers and tools. *Cray User Group (CUG) 2006 Proceedings*, pages 1–7, 2006. 81

[34] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989. 71

[35] P. Moscato. *New Ideas in Optimization*, chapter Memetic algorithms: A short introduction, pages 219–234. McGraw-Hill Ltd., UK, Maidenhead, England, 1999. 72

[36] D. M. Muñoz, C. H. Llanos, L. S. Coelho, and M. Ayala-Rincón. Opposition-based shuffled PSO with passive congregation applied to FM matching synthesis. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2775–2781. IEEE, 2011. 108

[37] M. Ozery-Flato and R. Shamir. An $O(n^{\frac{3}{2}}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations. *J. Discrete Algorithms*, 9(4):344–357, 2011. 20

[38] P. S. Pacheco. *An introduction to parallel programming.* Elsevier, Amsterdam, Netherlands, 2011. 79, 83

[39] C. D. Polychronopoulos, M. B. Gikar, M. R. Haghighat, C. L. Lee, B. P. Leung, and D. A. Schouten. The structure of parafrase-2: An advanced parallelizing compiler for C and FORTRAN. In *Selected Papers of the Second Workshop on Languages and Compilers for Parallel Computing*, pages 423–453, London, England, 1990. Pitman Publishing. 81

[40] C. Shannon. A mathematical theory of communication. *The Bell System Technical*, 27:379–423, 1948. 72, 76

[41] L. A. Silveira, J. L. Soncco-Álvarez, and M. Ayala-Rincón. Parallel memetic genetic algorithms for sorting unsigned genomes by translocations. Aceito para apresentação e publicação (IEEE Xplore) em CEC 2016: The IEEE annual Congress on Evolutionary Computation., 2016. 3, 5

[42] L. A. Silveira, J. L. Soncco-Álvarez, T. A. de Lima, and M. Ayala-Rincón. Computing translocation distance by a genetic algorithm. In *2015 Latin American Computing Conference, CLEI 2015, Arequipa*, pages 1–12. IEEE, 2015. 2, 5, 59, 106, 107

[43] L. A. Silveira, J. L. Soncco-Álvarez, T. A. de Lima, and M. Ayala-Rincón. Memetic and Opposition-Based Learning Genetic Algorithms for Sorting Unsigned Genomes by Translocations. In *Proc. 7th World Congress on Nature and Biologically Inspired Computing, NaBIC 2015, Pietermaritzburg, December, 2015*, volume 419 of *Advances in Intelligent Systems and Computing*, pages 73–85. Springer, 2016. 3, 5, 71, 97, 104, 106, 108, 113

[44] S. N. Sivanandam and S. N. Deepa. *Introduction to genetic algorithms*. Springer, New York, USA, 2007. 12, 17

[45] C. B. Tey, M. R. Leuze, and J. J. Grefenstette. Parallel genetic algorithm. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*, 1987. 86

[46] H. R. Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. *Computational Intelligence for Modelling, Control and Automation, International Conference on*, 1:695–701, 2005. 75

[47] H. R. Tizhoosh, M. Ventresca, and S. Rahnamayan. Opposition-based computing. In *Oppositional Concepts in Computational Intelligence*, pages 11–28. Springer, 2008. 75

[48] L. Wang, D. Zhu, X. Liu, and S. Ma. An $O(n^2)$ algorithm for signed translocation. *Journal of Computer and System Sciences*, 70(3):284–299, 2005. 2, 19

[49] R. P. Wilson, R. S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. Tjiang, S. Liao, Chau-Wen Tseng, M. W. Hall, M. S. Lam, et al. SUIF: An infrastructure for research on parallelizing and optimizing compilers. *ACM Sigplan Notices*, 29(12):31–37, 1994. 81

[50] D. Zhu and L. Wang. On the complexity of unsigned translocation distance. *Theoretical Computer Science*, 352(1):322–328, 2006. 2, 20, 21, 25, 27, 29, 30