Estudo de heurísticas para distância por transposição considerando regiões intergênicas e repetição de genes

TCC – Trabalho de Conclusão
Engenharia da Computação
Faculdade de Computação – Facom
Universidade Federal de Mato Grosso do Sul – UFMS

Candidato: Luiz Henrique Pio Freire
Orientador: Francisco Eloi Soares de Araujo

6 de dezembro de 2024

Resumo

Este trabalho aborda o cálculo de Distância de Transposição Intergênica (ITD) em genomas lineares com genes repetidos e regiões intergênicas, uma ferramenta relevante para a análise da evolução biológica. Estudamos uma solução aproximada dada por [Siqueira et al. 2021] que usa uma solução do Problema de Mínima Partição de String Intergênica Comum (MCISP) como subrotina. Tanto ITD quanto MCISP são problemas NP-difíceis. O trabalho tem como objetivo avaliar a relevância prática de usar MCISP como subrotina, substituindo-o por heurísticas conceitualmente mais simples. Os resultados preliminares sugerem que investir em heurísticas para esse problema nessa etapa pode resultar em soluções melhores.

1 Introdução

Comparar organismos é fundamental para compreender a diversidade da vida, incluindo a evolução das espécies, suas interações e adaptações [Futuyma 2013]. Além disso, esses conhecimentos têm aplicações importantes em várias áreas como medicina, ecologia, conservação e biotecnologia. Essas comparações revelam as semelhanças e diferenças fundamentais que influenciam a biologia das diversas formas de vida no planeta.

Um *genoma* é composto por *genes* e *regiões intergênicas*. Os *genes* são essenciais para a produção de proteínas e para a regulação dos processos celulares enquanto as *regiões intergênicas* exercem funções regulatórias importantes, determinando quando e como os genes são ativados, além de contribuir para a estrutura do *genoma*. Com os avanços na genômica, ficou evidente que essas regiões intergênicas são fundamentais para a regulação genética e a evolução do genoma, mostrando que, longe de serem apenas "DNA lixo" [Doolittle 2013], elas são elementos ativos e cruciais para o funcionamento das células.

Uma maneira de comparar organismos é comparando seus genomas e isso é feito por uma área da Biologia Computacional conhecida como Genômica Comparativa [Haubold e Wiehe 2004]. A *genômica comparativa* utiliza mutações no DNA como ferramentas para analisar a diversidade genética entre diferentes organismos. Essas mutações podem ser pontuais, que envolvem pequenas alterações em bases nucleicas isoladas, ou grandes mudanças representadas por mudanças na ordem dos genes, e que podem resultar em reestruturações significativas do genoma. Esses estudos ajudam a entender não apenas a evolução das espécies, mas também a funcionalidade dos genes e suas interações.

Considerando a comparação de genomas levando em conta a ordem dos genes, as medidas de distância entre os genomas podem ser baseadas em diferentes aspectos das mudanças estruturais causadas por eventos de rearranjo: as *mudanças estruturais* propriamente ditas que consideram como os eventos afetam a organização do genoma usando como medida a quantidade das mudanças observadas, tais como *distância por breakpoint* [Blin, Fertin e Chauve 2004] e a *distância considerando a partição mínima comum* [Kolman e Waleń 2007]; ou no número de eventos de rearranjo que quantifica a distância entre dois genomas com base no número mínimo de eventos de rearranjo necessários para transformar um genoma no outro. A "*distância de rearranjo*", é útil para entender a relação evolutiva entre espécies e como o ancestral comum divergiu em duas novas espécies. A distância de rearranjo depende diretamente do evento considerado, como, por exemplo, a *distância de transposição*. Essas medidas são complementares e oferecem *insights* sobre a evolução e a diversidade genética, permitindo que os pesquisadores compreendam melhor como as alterações estruturais impactam a funcionalidade e a adaptação dos genomas ao longo do tempo.

Duas medidas em particular interessam nesse trabalho: *distância por transposição* e uma versão considerando regiões intergênicas de *Minimum Common String Partition (MCSP)*. Uma transposição é uma operação que troca a posição de um bloco contínuo de elementos dentro de um genoma, e o problema da distância por transposição entre dois genomas é encontrar a menor sequência de operações de transposição que, aplicadas em um deles, transforma-o no outro. A versão de decisão do problema é NP-completa [Bulteau, Fertin e Rusu 2012] e possui um algoritmo com fator de aproximação de 1.375 [Elias e Hartman 2006]. Se considerarmos que a entrada é dada com regiões intergênicas, o problema resultante é claramente também NP-completo e possui um algoritmo com fator de aproximação de 3.5 [Oliveira et al. 2020].

O *Minimum Common String Partition Problem (MCSP)* é um problema de otimização aplicado a um par de genomas. O objetivo é particionar os genomas em subgenomas (ou blocos menores) de forma que os subgenomas resultantes possam ser rearranjados para transformar um genoma no outro com o menor número possível de blocos. Esse problema é classificado como *NP-difícil*, conforme demonstrado por [Goldstein, Kolman e Zheng 2004]. Além disso, o MCSP permanece difícil quando são consideradas as *regiões intergênicas*.

Nosso trabalho de conclusão de curso aborda o problema da distância de transposição em genomas, considerando genomas lineares com o mesmo conjunto de genes, o mesmo número de cópias de cada gene e regiões intergênicas. O artigo fundamental estudado em nossa pesquisa é *Approximation Algorithm for Rearrangement Distances Considering Repeated Genes and Intergenic Regions* [Siqueira et al. 2021], que apresenta um algoritmo com fator de aproximação $\Theta(k)$, onde k representa o número

máximo de cópias. Com base nos aspectos teóricos propostos nesse artigo, projetamos e implementamos duas heurísticas mais simples, denominadas *Randomized Genes* e *Naive Partition*, com o objetivo de avaliar a viabilidade de substituir o problema MCISP, utilizado no cálculo da distância por transposição (ITD), por abordagens mais acessíveis. Realizamos comparações entre os resultados obtidos com essas heurísticas e o algoritmo de aproximação original, utilizando o algoritmo fornecido pelos autores para gerar a base de dados e o próprio algoritmo de aproximação. Dessa forma, buscamos analisar a qualidade das soluções obtidas e a eficácia das heurísticas propostas.

Os conceitos abordados neste artigo envolvem a análise de genomas lineares com genes repetidos e regiões intergênicas e a aplicação da transposição como uma medida para calcular a distância entre dois genomas. A distância de transposição no contexto deste estudo, é aplicada para resolver o problema da distância de transposição intergênica (ITD). A partição intergênica, outro conceito fundamental, refere-se à divisão dos genomas em subgenomas, o que facilita o cálculo da distância de transposição. O artigo explora ainda o MCISP (Mínima Partição de String Intergênica Comum), um problema NP-difícil relacionado à partição intergênica, e discute como a correspondência entre o MCISP e o ITD pode ser utilizada para resolver o problema ITD. A pesquisa também revisita o algoritmo de aproximação proposto por [Siqueira et al. 2021], que possui um fator de aproximação $2k$ para o MCISP, e propõe duas heurísticas para melhorar a solução do problema ITD, buscando aumentar a eficiência no cálculo da distância de transposição com heurísticas mais simples.

2 Conceitos Fundamentais

2.1 Estrutura

Sequências ordenadas são representadas basicamente por um vetor com i elementos, onde uma lista de *caracteres* é chamada *string*. Tem-se que em uma lista X que contém um número i de elementos, $|X|$ é o número total de elementos de X e um elemento na i-ésima posição pode ser escrito como X_i.

Dada uma string S, define-se como o *alfabeto* de S o conjunto Σ_S contendo diferentes elementos. Cada elemento de Σ_S é chamado de *rótulo* de S. A *ocorrência* de um rótulo α em uma string S é o número de *caracteres* de S com rótulo α denotado por $occ(\alpha, S)$. É chamado de *único* o rótulo cuja ocorrência é igual a um e de *réplica* o rótulo que ocorre pelo menos duas vezes.

Duas *strings* S e P são *balanceadas* se $\sum_S = \sum_P$ e $occ(\alpha, S) = occ(\alpha, P), \forall \alpha \in \sum_S$, ou seja, *strings balanceadas* são formadas pelos mesmos *rótulos* em ordens possivelmente diferentes.

2.2 Genomas

Ao modelar os genomas, considera-se, neste trabalho, as regiões intergênicas entre os genes, as quais são representadas por seus comprimentos (números inteiros). As *regiões intergênicas* são as áreas do genoma localizadas entre os genes. Normalmente, um genoma real começa e termina com essas regiões, entretanto, para fins de representação,

serão adicionados dois *genes artificiais*, um no início e outro no final, processo denominado *capping*, o qual utiliza o mesmo par de genes para qualquer genoma.

Formalmente, um *genoma* $\mathcal{G} = (S_1, \breve{S}_1, S_2, \ldots, S_{n-1}, \breve{S}_{n-1}, g_n)$ de *tamanho* n é uma sequência intercalada de n genes $S = (S_1, \ldots, S_n)$ e $n-1$ regiões intergênicas $\breve{S} = (\breve{S}_1, \ldots, \breve{S}_{n-1})$. Quando conveniente representamos $\mathcal{G} = (S, \breve{S})$. A Figura 1 mostra um exemplo de um genoma.

Um *genoma* $\mathcal{G} = (S, \breve{S})$ é composto por uma string S e uma lista de inteiros $\breve{S}$ de forma que:

- O gene g_i é representado pelo rótulo S_i de S, para $1 \leq i \leq n$.
- A região intergênica $\breve{g}_i$ é dada pelo inteiro $\breve{S}_i$ de $\breve{S}$, para $1 \leq i \leq n-1$.

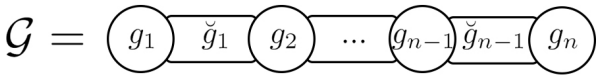

Figura 1: O *genoma* $\mathcal{G} = (g_1, \breve{g}_1, g_2, \ldots, gn-1, \breve{g}n-1, g_n)$ ilustra a representação de um genoma genérico

Dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ são chamados *co-caudais* caso tenham o mesmo gene inicial e final ($S_1 = P_1$ e $S_n = P_n$). Dois genomas $\mathcal{G}$ e $\mathcal{H}$ são considerados iguais ($\mathcal{G} = \mathcal{H}$) se as listas de rótulos de $\mathcal{G}$ e $\mathcal{H}$ forem idênticas, bem como as listas de inteiros $\breve{S}$ e $\breve{P}$.

Um *subgenoma* $\mathcal{G}^{i,j} = (\breve{S}^{i,j}, \breve{S}^{i,j})$ é uma porção do genoma $\mathcal{G} = (S, \breve{S})$ entre os genes g_i e g_j. Ou seja, um subgenoma $\mathcal{G}^{i,j}$ é representado pelas listas $S^{i,j}$ e $\breve{S}^{i,j}$, onde:

- $S_k^{i,j} = S_{i+k-1}, \forall k(1 \leq k \leq j-i+1)$
- $\breve{S}_k^{i,j} = \breve{S}_{i+k-1}, \forall k(1 \leq k \leq j-i)$.

Um genoma $\mathcal{G}$ contém outro genoma $\mathcal{H}$ se $\mathcal{H}$ for igual a um subgenoma de $\mathcal{G}$, o que é indicado pela notação $\mathcal{H} \subset \mathcal{G}$. Caso contrário, se $\mathcal{H}$ não for um subgenoma de $\mathcal{G}$, temos $\mathcal{H} \not\subset \mathcal{G}$.

Um genoma $\mathcal{K} = (Q, \breve{Q})$ é a combinação de dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ se a string Q for a *concatenação* das strings S e P, e se a lista $\breve{Q}$ for formada pela sequência $\breve{S}$ seguida de um inteiro (representando o tamanho da região intergênica entre os dois genomas), e depois pela sequência $\breve{P}$. A Figura 2 mostra um exemplo de um genoma formado pela concatenação de dois outros.

Dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ de tamanho n são chamados de *balanceados* se as strings S e P forem balanceadas e se a soma dos inteiros que representam as regiões intergênicas for igual em ambos os genomas ($\sum_{i=1}^{n-1} \breve{S}i = \sum^{n-1} i = 1\breve{P}_i$). A Figura 3 mostra um exemplo de dois genomas balanceados.

Sejam $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ dois genomas balanceados. Uma *atribuição ortóloga* ξ é uma correspondência entre os genes de S e os de P, tal que cada gene S_i de S é mapeado para um gene correspondente $\xi(S_i)$ de mesmo rótulo em P. A região intergênica que segue o gene $\xi(S_i)$ é representada por $\xi(\breve{S}_i)$. Dessa forma, cada *réplica* de S é associada a uma réplica de mesmo rótulo em P, considerando que podem existir múltiplas formas de associar os genes e suas regiões intergênicas.

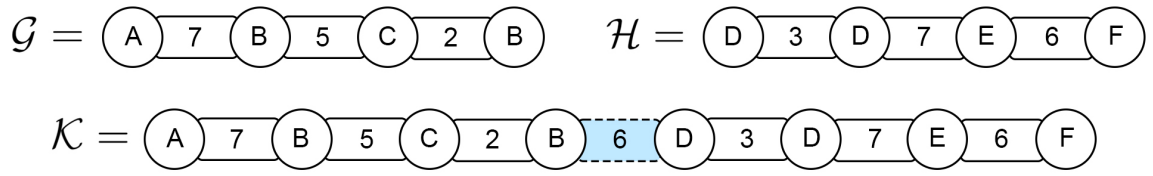

Figura 2: Os genomas $\mathcal{G} = ([A\ B\ C\ B], [7\ 5\ 2])$ e $\mathcal{H} = ([D\ D\ E\ F], [3\ 7\ 6])$ combinados para formar o genoma $\mathcal{K} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$. Note que, uma região intergênica com tamanho 6 foi encontrada durante a combinação. Além disso, o genoma $\mathcal{K}$ contém os genomas $\mathcal{G}(\mathcal{G} = \mathcal{K}^{1,4})$ e $\mathcal{H}(\mathcal{H} = \mathcal{K}^{5,8})$

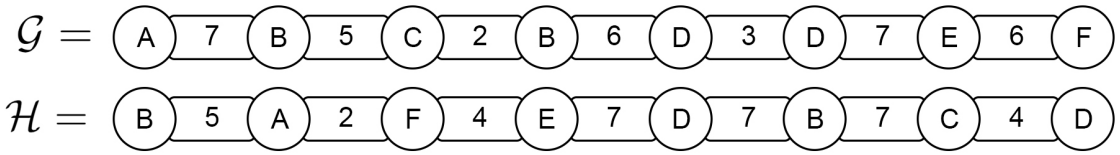

Figura 3: Os *genomas* $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ são dados pelos seguintes exemplos: $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([B\ A\ F\ E\ D\ B\ C\ D], [5\ 2\ 4\ 7\ 7\ 7\ 4])$. Ambos os genomas são balanceados, pois as strings S e P são balanceadas e a soma das regiões intergênicas é igual a 36 para ambos os genomas

O número total de atribuições ortólogas possíveis é dado por 2^m, onde m é a soma de $occ(\alpha_i, S) - 1$ para cada α_i, ou seja, a quantidade total de "escolhas"que podem ser feitas para atribuir valores ortológicos aos rótulos em S, levando em consideração o número de suas ocorrências.

Formalmente, temos:

$$m = \sum_{i=1}^{n} (occ(\alpha_i, S) - 1)$$

E o número total de atribuições ortólogas possíveis é:

$$2^m = 2^{\sum_{i=1}^{n} (occ(\alpha_i, S) - 1)}$$

Explicação:

- $occ(\alpha_i, S)$: Número de vezes que o rótulo α_i aparece na sequência S.
- $occ(\alpha_i, S) - 1$: Refere-se ao número de escolhas adicionais que podem ser feitas para as ocorrências de α_i, excluindo a ocorrência inicial.
- m: Soma de todas as escolhas possíveis para os diferentes rótulos $\alpha_1, \alpha_2, \ldots, \alpha_n$.
- 2^m: O número total de combinações possíveis de atribuições ortólogas, onde cada escolha tem duas opções.

A Figura 4 mostra uma *atribuição ortóloga* entre dois genomas $\mathcal{G}$ e $\mathcal{H}$.

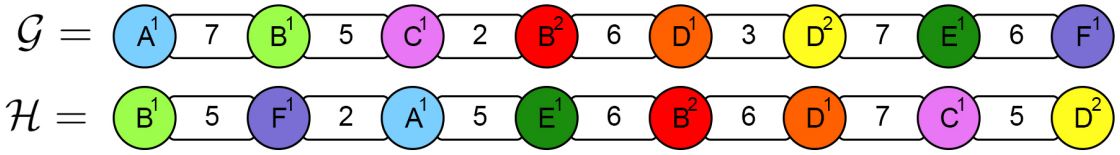

Figura 4: Uma possível atribuição ortóloga entre dois genomas balanceados $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([B\ F\ A\ E\ B\ D\ C\ D], [5\ 2\ 5\ 6\ 6\ 7\ 5])$. Observe que esta é apenas uma das possíveis atribuições ortólogas entre os genes dos dois genomas

2.3 Distância de transposição intergênica

Considerando um genoma $\mathcal{G} = (S, \breve{S})$ de tamanho n e os números i, j, k para os genes, e x, y, z para as regiões intergênicas, com $2 \leq i < j < k \leq n$, $0 \leq x \leq \breve{S}_{i-1}$, $0 \leq y \leq \breve{S}_{j-1}$ e $0 \leq z \leq \breve{S}_{k-1}$, temos a operação de *transposição intergênica* representada por $\tau^{(i,j,k)}_{(x,y,z)}$. Esta operação transforma o genoma $\mathcal{G}$ no novo genoma $\mathcal{G}.\tau^{(i,j,k)}_{(x,y,z)} = (S', \breve{S}')$, sendo que as sequências S' e $\breve{S}'$ são obtidas conforme as seguintes regras:

A sequência S' é gerada a partir de S com as posições das subsequências rearranjadas. O novo vetor $S' = (S'_1, S'_2, \ldots, S'_n)$ é definido por:

$$S'_h = \begin{cases} S_h & \text{se } h \leq i-1, \\ S_{h+k-j} & \text{se } i \leq h \leq j-1, \\ S_{h+i-j} & \text{se } j \leq h \leq k-1, \\ S_h & \text{se } h \geq k. \end{cases}$$

Resultando em:

$$S' = [S_1 \ldots S_{i-1}\ \underline{S_j \ldots S_{k-1}}\ \underline{S_i \ldots S_{j-1}}\ S_k \ldots S_n]$$

A sequência $\breve{S}'$ resulta em uma transformação da sequência $\breve{S}$, com modificações nos valores de algumas de suas posições, utilizando os valores x, y, z. O novo vetor $\breve{S}' = (\breve{S}'_1, \breve{S}'_2, \ldots, \breve{S}'_n)$ é definido por:

$$\breve{S}'_h = \begin{cases} \breve{S}_h & \text{se } h \leq i-2, \\ x + y' & \text{se } h = i-1, \\ \breve{S}_{h+k-j} & \text{se } i \leq h \leq j-2, \\ z + x' & \text{se } h = j-1, \\ \breve{S}_{h+i-j} & \text{se } j \leq h \leq k-2, \\ y + z' & \text{se } h = k-1, \\ \breve{S}_h & \text{se } h \geq k. \end{cases}$$

Resultando em:

$$\breve{S}' = [\breve{S}_1 \ldots \breve{S}_{i-2}\ \underset{\sim}{x + y'}\ \underline{\breve{S}_j \ldots \breve{S}_{k-2}}\ \underset{\sim}{z + x'}\ \underline{\breve{S}_i \ldots \breve{S}_{j-2}}\ \underset{\sim}{y + z'}\ \breve{S}_k \ldots \breve{S}_{n-1}]$$

As variáveis x', y', e z' são definidas em função dos valores em $\breve{S}$ nas posições $i-1$, $j-1$, e $k-1$, respectivamente, conforme descrito a seguir. Especificamente:

$$x' = \breve{S}_{i-1} - x, \quad y' = \breve{S}_{j-1} - y, \quad z' = \breve{S}_{k-1} - z.$$

Com isso, o genoma $\mathcal{G}$ será transformado na sequência $\mathcal{G}.\tau^{(i,j,k)}_{(x,y,z)}$, que reflete a nova configuração das subsequências S' e $\breve{S}'$.

A Figura 5 mostra um exemplo de uma *transposição intergênica* aplicada em um *genoma* $\mathcal{G}$.

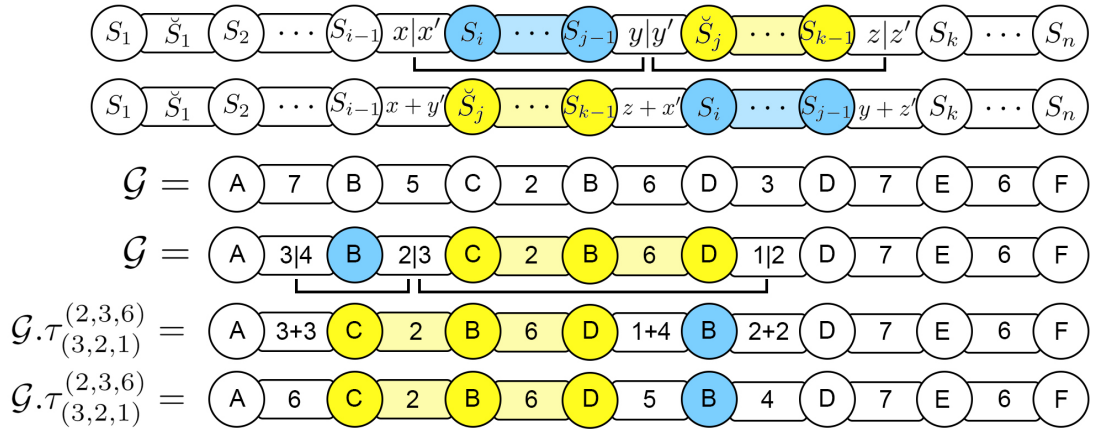

Figura 5: Representação genérica de uma transposição intergênica seguida de uma aplicação de transposição intergênica $\tau^{(2,3,6)}_{(3,2,1)}$ no *genoma* $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ resultando no genoma $\mathcal{G}.\tau^{(2,3,6)}_{(3,2,1)} = ([A\ C\ B\ D\ B\ D\ E\ F], [6\ 2\ 6\ 5\ 4\ 7\ 6])$

Problema 1 (ITD - distância de transposição intergênica) .

Instância: *Genomas* $\mathcal{G}, \mathcal{H}$.

Objetivo: *determinar o número mínimo de transposições intergênicas que transforma* $\mathcal{G}$ *em* $\mathcal{H}$.

ITD é uma generalização do problema *distância por transposição* que é conhecido ser NP-completo [Bulteau, Fertin e Rusu 2012]. Logo, vale

Teorema 1 *ITD é NP-completo.*

A *distância de transposição intergênica* (ITD) é uma generalização do problema clássico de *distância por transposição*, onde o objetivo é determinar o número mínimo de transposições necessárias para transformar um genoma em outro. O problema ITD envolve transformações mais complexas entre dois genomas, permitindo manipulações flexíveis em subsequências e rearranjos de segmentos.

A dificuldade de resolver o problema ITD é evidente, uma vez que ele pode envolver um número maior de operações quando se lida com genomas mais complexos. De fato, o problema ITD é NP-completo, o que significa que, em genomas grandes, encontrar uma solução exata pode ser computacionalmente inviável dentro de um tempo razoável, a menos que P = NP. Isso torna o problema ITD um desafio importante na computação

e na biologia computacional, pois soluções exatas podem ser inviáveis para genomas grandes.

Uma estratégia comum é utilizar algoritmos de aproximação, que fornecem soluções próximas da ótima com um fator de aproximação garantido. Esses algoritmos permitem encontrar soluções razoáveis para o número de transposições, mesmo quando o cálculo exato seria computacionalmente inviável.

2.4 Partição intergênica

Uma *partição intergênica* é o processo de dividir um genoma em subgenomas menores, chamados de *blocos*, onde cada bloco representa uma parte do genoma que pode ser reorganizada de acordo com regras específicas. Em termos simples, uma partição intergênica divide o genoma em segmentos, e cada subgenoma resultante pode ser analisado separadamente, mas sempre levando em conta a interação e a sequência dos segmentos no genoma completo.

Formalmente, dado dois *genomas balanceados* $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$, uma *partição intergênica* entre eles é um par de sequências de genomas $(\mathbb{S}, \mathbb{P})$, tal que:

1. Os *genomas* de $\mathbb{S}$, quando reorganizados, correspondem ao *genoma* $\mathcal{G}$.
2. Os *genomas* de $\mathbb{P}$, quando reorganizados, correspondem ao *genoma* $\mathcal{H}$.
3. Existe uma *permutação* ϕ dos índices $1, 2, \ldots, |\mathbb{S}|$ tal que os elementos de $\mathbb{P}$ são obtidos permutando os elementos de $\mathbb{S}$, ou seja, $\mathbb{P}_i = \mathbb{S}_{\phi_i}$, $\forall i (1 \leq i \leq |\mathbb{S}|)$.

A Figura 6 mostra um exemplo de uma partição intergênica $(\mathbb{S}, \mathbb{P})$ aplicada entre um *genoma* $\mathcal{G}$ e $\mathcal{H}$.

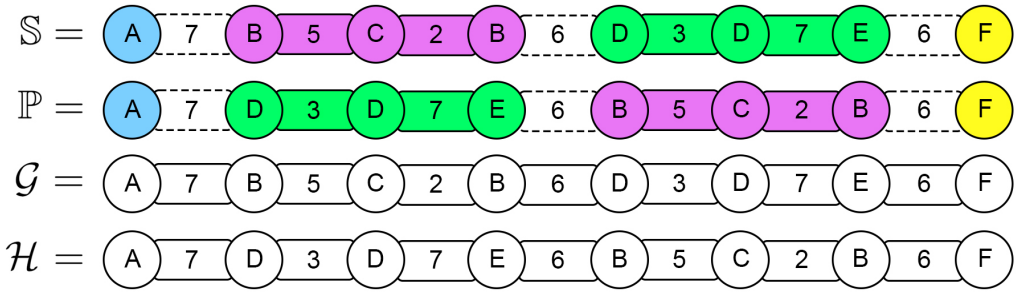

Figura 6: Representação de uma partição intergênica $(\mathbb{S}, \mathbb{P})$, sendo $\mathbb{S} = [([A], [\ ])([B\ \ C\ \ B], [5\ \ 2])([D\ \ D\ \ E], [3\ \ 7])([F], [\ ])]$ e $\mathbb{P} = [([A], [\ ])([D\ \ D\ \ E], [3\ \ 7])([B\ \ C\ \ B], [5\ \ 2])([F], [\ ])]$, entre dois genomas balanceados $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ sendo esses $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 3\ 5\ 3\ 7\ 6])$. A permutação de blocos de $\mathbb{S}$ resulta na formação de $\mathbb{P}$, conforme as regras de reorganização

Na partição intergênica descrita, os genomas correspondentes aos elementos de $\mathbb{S}$ e $\mathbb{P}$ são denominados *blocos*, os quais representam subgenomas dos genomas $\mathcal{G}$ e $\mathcal{H}$, respectivamente. Os blocos de $\mathbb{S}$ podem ser representados por $b(\mathbb{S})$. Como os blocos de $\mathbb{S}$ devem ser reorganizados para reconstruir $\mathcal{G}$, é necessário que a ordem dos blocos

siga a sequência em que esses blocos aparecem em $\mathcal{G}$, de forma que cada gene esteja presente em algum bloco.

Os blocos de $\mathbb{S}$, representados por $b(\mathbb{S})$ são as unidades menores que compõem o genoma $\mathcal{G}$.A ordem dos blocos é crucial para a reconstrução do genoma $\mathcal{G}$, pois a sequência dos blocos deve ser mantida para refletir a estrutura original do genoma.

Por outro lado, algumas regiões intergênicas não estão presentes em $\mathbb{S}$. Essas regiões precisam ser inseridas durante a recombinação dos blocos e são responsáveis por marcar os pontos de divisão do genoma $\mathcal{G}$ nos blocos. Esses pontos de divisão são chamados de *breakpoints* de $\mathbb{S}$, e a definição de breakpoints para $\mathbb{P}$ segue uma lógica análoga.

Dois breakpoints $\breve{X}_i$ e $\breve{Y}_j$ são *correspondentes* se os elementos ao redor desses breakpoints nas duas sequências X e Y forem os mesmos, ou seja, se $X_i = Y_i$ e $X_{i+1} = Y_{i+1}$, ou se a troca de posição entre os elementos X_i, X_{i+1} e Y_i, Y_{i+1} ocorrer de forma correspondente nas duas sequências.

O *custo* de uma partição intergênica $(\mathbb{S}, \mathbb{P})$ é definido como o número de breakpoints presentes na sequência $\mathbb{S}$. Esse custo também pode ser calculado como $custo(\mathbb{S}, \mathbb{P}) = b(\mathbb{S}) - 1$. Vale ressaltar que as sequências $\mathbb{S}$ e $\mathbb{P}$ devem conter o mesmo número de blocos, o que garante que o custo seja o mesmo, independentemente da sequência considerada.

Uma partição intergênica é considerada *mínimal* quando não é possível unir dois blocos consecutivos de forma a criar uma nova partição intergênica com custo menor.

Uma atribuição ortóloga entre dois genomas $\mathcal{G}$ e $\mathcal{H}$ mapeia os genes de $\mathcal{G}$ para os genes de $\mathcal{H}$ de forma que resulta em uma partição intergênica *mínima*, embora possa não ser única, dependendo das relações de *semelhança* e estrutura dos genomas.

Considerando uma atribuição ortóloga ξ entre dois genomas balanceados $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$, e a *partição intergênica mínima* $(\mathbb{S}, \mathbb{P})$ induzida por ξ, podemos classificar os breakpoints de $\mathbb{S}$ em dois tipos. Um breakpoint $\breve{S}i$ é chamado de *hard* se os genes $\xi(S_i)$ e $\xi(S_{i+1})$ forem *adjacentes* em P, ou seja, se $\xi(S_i)$ e $\xi(S_{i+1})$ ocorrerem consecutivamente na sequência P, sem qualquer outro gene entre eles. Se o breakpoint não for hard, ele é classificado como *soft*. Um breakpoint hard é considerado *overcharged* se $\breve{S}_i > \xi(\breve{S}_i)$, ou *undercharged* se $\breve{S}_i < \xi(\breve{S}_i)$. Um exemplo de uma partição intergênica entre os genomas $\mathcal{G}$ e $\mathcal{H}$ com breakpoints hard e soft é mostrado na Figura 7.

Além disso, uma transposição intergênica $\tau^{(i,j,k)}_{(x,y,z)}$ aplicada a $\mathcal{G}$ remove b breakpoints de $\mathbb{S}$ se, após a transposição, o custo da nova partição $(\mathbb{R}, \mathbb{Q})$ entre $\mathcal{G}.\tau^{(i,j,k)}_{(x,y,z)}$ e $\mathcal{H}$, induzida pela atribuição ξ, for tal que a relação a seguir seja satisfeita: $custo(\mathbb{R}, \mathbb{Q}) = custo(\mathbb{S}, \mathbb{P}) - b$.

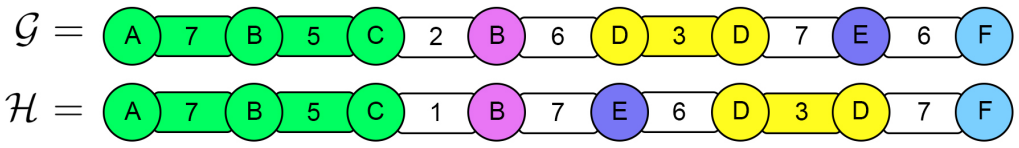

Figura 7: Representação de uma partição intergênica entre dois *genomas* $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ sendo esses $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([A\ B\ C\ B\ E\ D\ D\ F], [7\ 5\ 1\ 7\ 6\ 3\ 7])$. As regiões intergênicas em branco são os breakpoints e cada bloco é mostrado em uma cor diferente. O breakpoint entre os genes C e D é hard pois os genes $\xi(S_3)$ e $\xi(S_{3+1})$ são adjacentes em P. Além de hard é overcharged pois $\breve{S}_3 > \xi(\breve{S}_3)$. Os outros breakpoints são softs

Problema 2 (MCISP - Mínima Partição de String Intergênica Comum) .
Instância: *Genomas* $\mathcal{G}$ *e* $\mathcal{H}$.
Objetivo: *Encontrar a partição intergênica de* custo mínimo *entre os genomas* $\mathcal{G}$ *e* $\mathcal{H}$.

MCISP é uma generalização do problema de *partição mínima de strings*, que é conhecido ser NP-difícil [Siqueira et al. 2021]. Logo, vale

Teorema 2 *MCISP é NP-difícil.*

O problema MCISP está intimamente relacionado aos problemas de partição e distância entre genomas. Em particular, ele busca uma partição que minimize as distâncias entre genes em dois genomas, levando em consideração suas relações de correspondência. A solução para o problema MCISP pode ser aplicada como uma aproximação para o problema *ITD*, uma vez que a minimização do custo da partição intergênica está diretamente relacionada à otimização das transformações entre os genomas.

Por ser um problema NP-difícil, a solução proposta recorre, novamente, ao uso de algoritmos de aproximação como uma estratégia eficaz para encontrar soluções subótimas em tempo viável.

2.5 Correspondências entre MCISP partição e ITD

As transposições intergênicas são transformações que alteram a ordem dos genes em um genoma. A relação entre a partição e a transposição é central para entender como uma boa partição pode auxiliar um algoritmo que procura uma sequência pequena de transposições que transforma um genoma em outro. A partição, ao dividir os genomas em subconjuntos com base em correspondências ortológicas, oferece uma base para sugerir quais transposições seriam convenientes para transformar um genoma no outro.

Os lemas 1, 2, 3, 4 e 5 estabelecem uma conexão entre os problemas de partição e de distância. Essa relação possibilita a aplicação de uma abordagem aproximada para resolver o problema MCISP, com o intuito de usá-lo como subrotina para resolver o problema ITD. Adicionalmente, os lemas apresentados, cujas demonstrações omitimos, fornecem limites inferiores para as distâncias, baseando-se no custo das partições.

Lema 1 *[Siqueira et al. 2021] Seja* $(\mathbb{S}, \mathbb{P})$ *uma partição intergênica mínima induzida por uma atribuição ortóloga entre dois genomas balanceados* $\mathcal{G} = (S, \breve{S})$ *e* $\mathcal{H} = (P, \breve{P})$. *Para qualquer transposição intergênica* $\tau_{(x,y,z)}^{(i,j,k)}$, *a partição intergênica mínima* $(\mathbb{R}, \mathbb{Q})$ *entre os genomas* $\mathcal{G}.\tau_{(x,y,z)}^{(i,j,k)}$ *e* $\mathcal{H}$, *induzida pela mesma atribuição ortóloga, respeita a restrição* $custo(\mathbb{R}, \mathbb{Q}) \geq custo(\mathbb{S}, \mathbb{P}) - 3$.

A Figura 8 mostra um exemplo gráfico com genomas do Lema 1.

Lema 2 *[Siqueira et al. 2021] Seja* $(\mathbb{S}, \mathbb{P})$ *uma partição intergênica de custo mínimo entre dois genomas balanceados* $\mathcal{G} = (S, \breve{S})$ *e* $\mathcal{H} = (P, \breve{P})$. *Qualquer sequência de transposições intergênicas que transforma* S *em* P *deve ter tamanho pelo menos* $custo(\mathbb{S}, \mathbb{P})/3$.

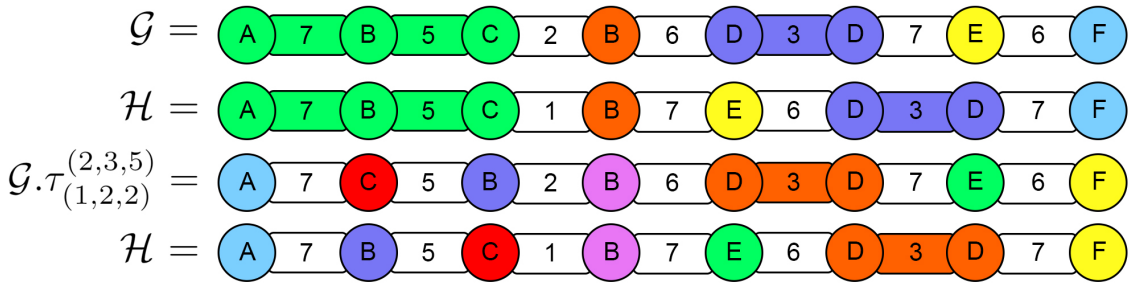

Figura 8: Dados dois genomas $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([A\ B\ C\ B\ E\ D\ D\ F], [7\ 5\ 1\ 7\ 6\ 3\ 7])$. A partição intergênica mínima é $(\mathbb{S}, \mathbb{P})$, sendo $\mathbb{S} = [([A\ B\ C], [7\ 5])([B], [\ ])([D\ D], [3])([E], [\ ])([F], [\ ])]$ e $\mathbb{P} = [([A\ B\ C], [7\ 5])([B], [\ ])([E], [\ ])([D\ D], [3])([F], [\ ])]$ onde $(\mathbb{S}, \mathbb{P})$ tem $custo(\mathbb{S}, \mathbb{P}) = 4$. Após fazer uma transposição intergênica $\tau^{(2,3,5)}_{(1,2,2)}$, a partição mínima $(\mathbb{R}, \mathbb{Q})$ entre os genomas $\mathcal{G}.\tau^{(2,3,5)}_{(1,2,2)}$ e $\mathcal{H}$, o custo segue o Lema 1, e é obtido $custo(\mathbb{R}, \mathbb{Q}) = 6$

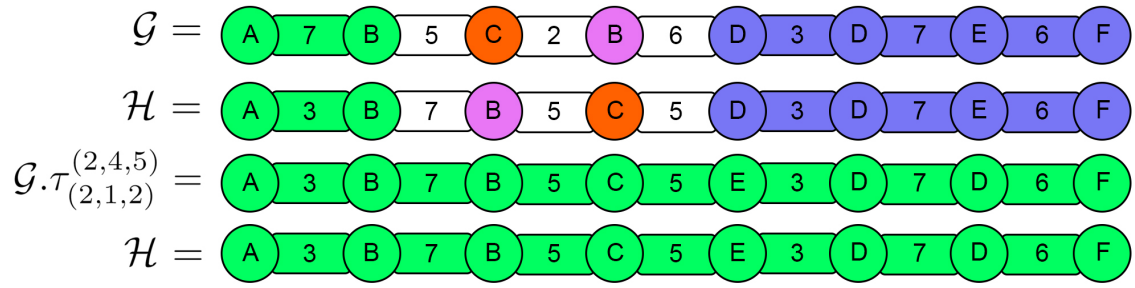

Figura 9: Dados dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ sendo esses $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([A\ B\ B\ C\ D\ D\ E\ F], [3\ 7\ 5\ 5\ 3\ 7\ 6])$. A partição intergênica mínima é $(\mathbb{S}, \mathbb{P})$, sendo $\mathbb{S} = [([A\ B], [7])([C], [\ ])([B], [\ ])([D\ D\ E\ F], [3\ 7\ 6])]$ e $\mathbb{P} = [([A\ B], [3])([B], [\ ])([C], [\ ])([D\ D\ E\ F], [3\ 7\ 6])]$ onde $(\mathbb{S}, \mathbb{P})$ tem $custo(\mathbb{S}, \mathbb{P}) = 3$. Nesse caso, basta uma única transposição intergênica para transformar $\mathcal{G}$ em $\mathcal{H}$. Após fazer a transposição $\tau^{(2,4,5)}_{(2,1,2)}$, $\mathcal{G}.\tau^{(2,4,5)}_{(2,1,2)}$ fica igual a $\mathcal{H}$, seguindo o Lema 2 pois foi realizada apenas uma transposição intergênica

A Figura 9 mostra um exemplo gráfico com genomas do Lema 2.

Lema 3 *[Siqueira et al. 2021] Seja $\mathcal{G} = (S, \breve{S})$ um genoma. Dada uma sequência de duas transposições intergênicas $\tau^{(i+1,j+1,k+1)}_{(\phi_i,\phi_j,\phi_k)}$, $\tau^{(i+1,i+k-j+1,k+1)}_{(\phi'_i,\phi'_{i+k-j},\phi'_k)}$ aplicadas nessa ordem, é possível encontrar valores para $\phi_i, \phi_j, \phi_k, \phi'_i, \phi'_{i+k-j}, \phi'_k$ para realizar qualquer redistribuição de nucleotídeos dentro das regiões $\breve{S}_i, \breve{S}_j$ e $\breve{S}_k$.*

Note que, após as duas transposições intergênicas descritas no Lema 3, a sequência S permanece a mesma. A Figura 10 mostra um exemplo gráfico com genomas do Lema 3.

Lema 4 *[Siqueira et al. 2021] Dados dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$, e uma atribuição ortóloga ξ entre eles. Seja $(\mathbb{S}, \mathbb{P})$ a partição mínima derivada da atribuição*

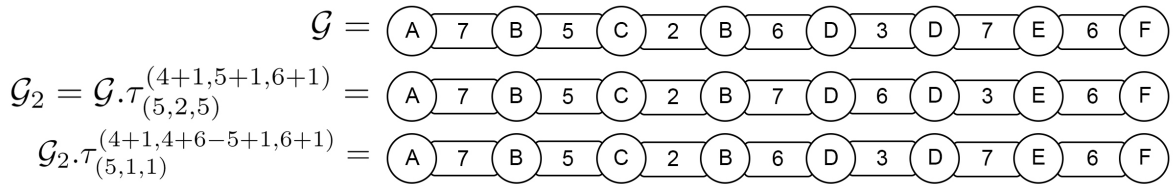

Figura 10: Representação de duas transposições intergênicas $\mathcal{G}.\tau^{(4+1,5+1,6+1)}_{(5,2,5)}$ e $\mathcal{G}_2.\tau^{(4+1,4+6-5+1,6+1)}_{(5,1,1)}$ realizadas no genoma $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ resultando no mesmo genoma $\mathcal{G}$ inicial

ortóloga ξ. Se $\mathbb{S}$ tem um breakpoint soft $\breve{S}_i$ tal que $\breve{S}_i \geq \xi(\breve{S}_i)$, então podemos aplicar uma transposição intergênica em $\mathcal{G}$ que remove pelo menos um breakpoint de $\mathbb{S}$. Além disso, se $\mathbb{S}$ tem pelo menos 4 breakpoints softs e nenhum breakpoint $\breve{S}_r$, $r \neq i$, tal que $\breve{S}_r \geq \xi(\breve{S}_r)$, podemos escolher uma transposição intergênica que não cria breakpoints overcharged.

A Figura 11 mostra um exemplo gráfico com genomas do Lema 4.

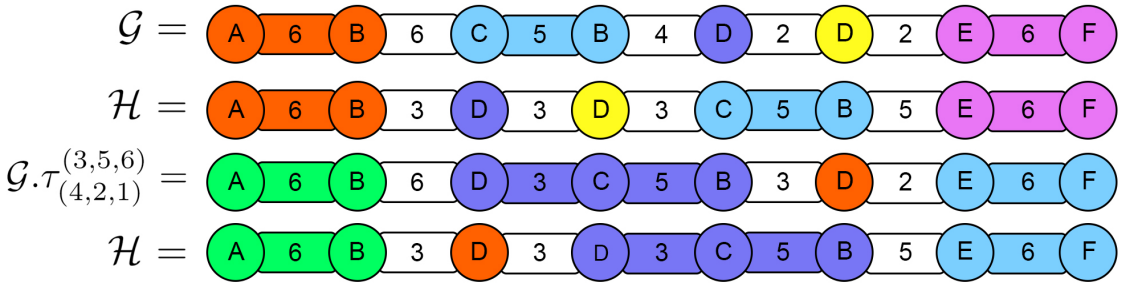

Figura 11: Dados dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ sendo esses $\mathcal{G} = ([A\ B\ C\ D\ E\ F\ G], [6\ 6\ 5\ 4\ 2\ 2])$ e $\mathcal{H} = ([A\ B\ D\ D\ C\ B\ E\ F], [6\ 3\ 3\ 3\ 5\ 5\ 6])$. A partição intergênica mínima é $(\mathbb{S}, \mathbb{P})$, sendo $\mathbb{S} = [([A\ B], [6])([C\ B], [5])([D], [\ ])([D], [\ ])([E\ F], [6])]$ e $\mathbb{P} = [([A\ B], [6])([D], [\ ])([D], [\ ])([C\ B], [5])([E\ F], [6])]$. Nesse caso é necessário uma transposição intergênica $\tau^{(3,5,6)}_{(4,2,1)}$ realizada no genoma $\mathcal{G}$ para remover um breakpoint e não criar breakpoint overchargeds, assim resultando no genoma $\mathcal{G}.\tau^{(3,5,6)}_{(4,2,1)} = ([A\ B\ D\ D\ C\ B\ E\ F], [6\ 3\ 3\ 3\ 5\ 5\ 6])$

Lema 5 *[Siqueira et al. 2021] Dados dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ e uma atribuição ortóloga ξ entre eles, é possível transformar $\mathcal{G}$ em $\mathcal{H}$ usando no máximo custo$(\mathbb{S}, \mathbb{P}) + 1$ transposições intergênicas, onde $(\mathbb{S}, \mathbb{P})$ é a partição minimal derivada da atribuição ortóloga ξ.*

A Figura 12 mostra um exemplo gráfico com genomas do Lema 5.

Como demonstrado por [Siqueira et al. 2021], os lemas 1, 3 e 4 foram usados na demonstração dos lemas 2 e 5.

Essas propriedades, discutidas nos lemas anteriores, revelam informações importantes sobre a relação entre a partição intergênica e o número de transposições necessárias

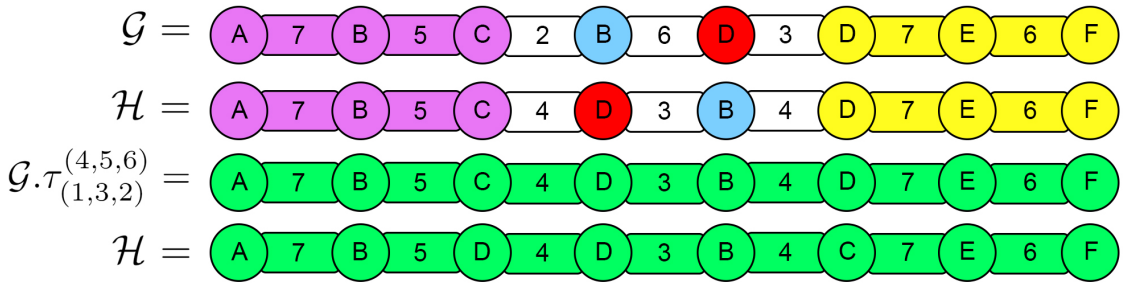

Figura 12: Dados dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$ sendo esses $\mathcal{G} = ([A\ B\ C\ B\ D\ D\ E\ F], [7\ 5\ 2\ 6\ 3\ 7\ 6])$ e $\mathcal{H} = ([A\ B\ C\ D\ B\ D\ E\ F], [7\ 5\ 4\ 3\ 4\ 7\ 6])$. A partição intergênica mínima é $(\mathbb{S}, \mathbb{P})$, sendo $\mathbb{S} = [([A\ B\ C], [7\ 5])([B], [\ ])([D], [\ ])([D\ E\ F], [7\ 6])]$ e $\mathbb{P} = [([A\ B\ C], [7\ 5])([D], [\ ])([B], [\ ])([D\ E\ F], [7\ 6])]$ onde $(\mathbb{S}, \mathbb{P})$ tem $custo(\mathbb{S}, \mathbb{P}) = 3$. Nesse caso é necessário apenas uma transposição intergênica $\tau^{(4,5,6)}_{(1,2,3)}$ realizada no genoma $\mathcal{G}$ para remover os 3 breakpoints e transformar $\mathcal{G}$ em $\mathcal{H}$, assim dentro do limitante máximo de transposições do Lema 5 sendo esse $custo(\mathbb{S}, \mathbb{P}) + 1$ transposições intergênicas para transformar $\mathcal{G}$ em $\mathcal{H}$. O genoma resultante é $\mathcal{G}.\tau^{(4,5,6)}_{(1,2,3)} = ([A\ B\ C\ D\ B\ D\ E\ F], [7\ 5\ 4\ 3\ 4\ 7\ 6])$

para transformar um genoma em outro. Em particular, [Siqueira et al. 2021] mostrou que elas fornecem limites sobre como a partição pode ser usada para controlar o custo das transposições. O lema 1 mostra que a partição mínima pode ser alterada por transposições, mas o custo não aumenta de forma descontrolada. O lema 2 estabelece uma relação entre o custo da partição e o número mínimo de transposições, enquanto o lema 5 garante que a transformação de um genoma em outro pode ser realizada com um número limitado de transposições, baseando-se na partição mínima.

Esses lemas descrevem como a partição, ao dividir os genomas em subconjuntos de genes, influencia diretamente a quantidade e o tipo de transposições necessárias para transformar um genoma em outro. Quanto mais eficiente for a partição, menor será o número de transposições necessárias para alcançar a transformação, o que se reflete em um custo menor para a distância de transposição intergênica.

MCISP é uma generalização do problema de *partição mínima de strings*, que é conhecido ser NP-difícil [Siqueira et al. 2021]. Logo, o teorema seguinte é válido

Teorema 3 *Uma aproximação l para o problema* MCISP *garante uma aproximação assintótica de* $3l$ *para o problema ITD.*

Com base nas propriedades dos lemas 2 e 5, [Siqueira et al. 2021] demonstrou que o problema da distância de transposição intergênica, considerando que não há genes repetidos, pode ser resolvido com um algoritmo que possui um fator de aproximação assintótico de 3. Ou seja, no pior cenário, a solução fornecida pelos autores será, no máximo, três vezes maior que a solução ótima. Vale ressaltar que o melhor fator de aproximação conhecido na literatura para esse problema é 3,5 [Oliveira et al. 2020].

Portanto, a partição intergênica desempenha um papel importante na análise de distâncias de transposição intergênica. O algoritmo de partição, ao construir uma divisão ótima dos genomas, facilita o cálculo do número de transposições necessárias para

transformar um genoma em outro, fornecendo uma estimativa eficiente da distância de transposição. Os lemas apresentados no artigo ajudam a estabelecer os limites inferiores para o custo da transposição, mostrando como a estrutura de partição influencia diretamente a complexidade do problema de distâncias entre genomas.

2.6 Algoritmo com fator de aproximação $2k$ para MCISP

Será apresentado um algoritmo para o problema *MCISP* entre dois genomas $\mathcal{G} = (S, \breve{S})$ e $\mathcal{H} = (P, \breve{P})$, com um fator de aproximação de $2k$, onde $k = \text{occ}(S)$, como mostrado por [Siqueira et al. 2021].

Para descrever o algoritmo, são necessárias duas funções:

- `subgen` $(\mathcal{G}, \mathcal{X})$: Retorna o número de subgenomas em $\mathcal{G}$ que são iguais a $\mathcal{X}$ (cada um desses subgenomas corresponde a uma ocorrência de $\mathcal{X}$).
- `weight` $(\mathcal{G}, \mathcal{H}, \mathcal{X}) =$ `subgen`$(\mathcal{G}, \mathcal{X}) -$ `subgen`$(\mathcal{H}, \mathcal{X})$: Um valor que indica o número de ocorrências de $\mathcal{X}$ em excesso em $\mathcal{G}$ ou em $\mathcal{H}$. Se o valor for positivo, $\mathcal{G}$ contém mais ocorrências de $\mathcal{X}$ do que $\mathcal{H}$. Se o valor for negativo, $\mathcal{H}$ tem mais ocorrências de $\mathcal{X}$ do que $\mathcal{G}$.

A função `weight` pode ser generalizada para operar com duas sequências de genomas $\mathbb{S}$ e $\mathbb{P}$ da seguinte forma:

$$\texttt{weight}(\mathbb{S}, \mathbb{P}, \mathcal{X}) = \sum_{i=1}^{|\mathbb{S}|} \text{subgen}(\mathbb{S}_i, \mathcal{X}) - \sum_{i=1}^{|\mathbb{P}|} \text{subgen}(\mathbb{P}_i, \mathcal{X})$$

Lema 6 *Dados dois genomas $\mathcal{G} = (S, \breve{S}), \mathcal{H} = (P, \breve{P})$, e um par $(\mathbb{S}, \mathbb{P})$ de sequências de genomas, tal que ele satisfaça as condições 1 e 2 de partição intergênica, temos que $(\mathbb{S}, \mathbb{P})$ satisfaz a condição 3 se e somente se* `weight` $(\mathbb{S}, \mathbb{P}, \mathcal{X}) = 0$ *para todos os genomas $\mathcal{X}$ contidos em $\mathcal{G}$ ou em $\mathcal{H}$ [Siqueira et al. 2021].*

Dado dois genomas $\mathcal{G}$ e $\mathcal{H}$, podemos facilmente construir um par de sequências de genomas $(\mathbb{S}, \mathbb{P})$ que satisfaça as condições 1 e 2 da partição intergênica. Para isso, basta escolher as regiões intergênicas de $\mathcal{G}$ e $\mathcal{H}$ que serão os breakpoints em $\mathbb{S}$ e $\mathbb{P}$, respectivamente. Segundo o Lema 6, para garantir que $(\mathbb{S}, \mathbb{P})$ seja uma partição intergênica de $\mathcal{G}$ e $\mathcal{H}$, devemos escolher os breakpoints de forma que $\texttt{weight}(\mathbb{S}, \mathbb{P}, \mathcal{X}) = 0$ para todos os genomas $\mathcal{X}$ de $\mathcal{G}$ ou $\mathcal{H}$.

Seja $\mathbf{T}_{\mathcal{G},\mathcal{H}}$ o conjunto de todos os subgenomas $\mathcal{X}$, tal que $\mathcal{X} \subset \mathcal{G}$ ou $\mathcal{X} \subset \mathcal{H}$, e $\texttt{weight}(\mathcal{G}, \mathcal{H}, \mathcal{X}) \neq 0$. Considere também o subconjunto

$$\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}} = \{\mathcal{X} \in \mathbf{T}_{\mathcal{G},\mathcal{H}} \mid \mathcal{Y} \not\subset \mathcal{X}, \forall \mathcal{Y} \in \mathbf{T}_{\mathcal{G},\mathcal{H}}, \mathcal{Y} \neq \mathcal{X}\}$$

.

Note que, para incluir um breakpoint em alguma ocorrência de um genoma $\mathcal{Y} \in \mathbf{T}_{\mathcal{G},\mathcal{H}} \setminus \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$, basta incluir um breakpoint na ocorrência correspondente de um genoma $\mathcal{X} \in \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$, tal que $\mathcal{X} \subset \mathcal{Y}$.

Por essa razão, começamos incluindo breakpoints nos elementos de $\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$. De fato, o seguinte lema garante que devemos incluir pelo menos um breakpoint para cada elemento de $\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$.

Lema 7 *Para construir uma partição intergênica* $(\mathbb{S}, \mathbb{P})$ *entre dois genomas* $\mathcal{G}$ *e* $\mathcal{H}$*, devemos incluir um breakpoint em pelo menos uma cópia de cada elemento* $\mathcal{X} \in \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$ *[Siqueira et al. 2021].*

Em alguns casos, pode ser necessário inserir breakpoints em múltiplas ocorrências de um genoma $\mathcal{X} \in \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$. Define-se $\texttt{break}(\mathcal{X})$ como o breakpoint associado ao genoma $\mathcal{X}$. Quando um breakpoint é inserido em uma ocorrência de $\mathcal{X}$, seleciona-se sempre um breakpoint correspondente a $\texttt{break}(\mathcal{X})$.

Para a inclusão dos breakpoints, não basta apenas identificar os genomas que possuem um peso inicialmente diferente de zero em $\mathcal{G}$ ou $\mathcal{H}$, mas também é necessário monitorar as mudanças nos pesos dos genomas após a inserção de cada breakpoint. Com esse objetivo, os conjuntos $\mathbf{T}_{\mathcal{G},\mathcal{H}}$ e $\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$ são generalizados para sequências de genomas. Seja $\mathbb{S}$ e $\mathbb{P}$ duas sequências de genomas, o conjunto $\mathbf{T}_{\mathbb{S},\mathbb{P}}$ é composto pelos genomas $\mathcal{X}$ que satisfazem a condição $\mathcal{X} \subset \mathbb{S}_i$ para $1 \leq i \leq |\mathbb{S}|$, ou $\mathcal{X} \subset \mathbb{P}_j$ para $1 \leq j \leq |\mathbb{P}|$, e para os quais $\texttt{weight}(\mathbb{S}, \mathbb{P}, \mathcal{X}) \neq 0$. Além disso, o conjunto $\mathbf{T}^{\min}_{\mathbb{S},\mathbb{P}}$ é definido como $\mathbf{T}^{\min}_{\mathbb{S},\mathbb{P}} = \{\mathcal{X} \in \mathbf{T}_{\mathbb{S},\mathbb{P}} \mid \mathcal{Y} \not\subset \mathcal{X}, \forall \mathcal{Y} \in \mathbf{T}_{\mathbb{S},\mathbb{P}}, \mathcal{Y} \neq \mathcal{X}\}$.

Seja $\texttt{break}(\mathcal{X})$ o breakpoint associado ao genoma $\mathcal{X} \in \mathbf{T}^{\min}_{\mathbb{S},\mathbb{P}}$. Caso $\mathcal{X} \in \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$, o breakpoint $\texttt{break}(\mathcal{X})$ já está definido. Caso contrário, deve ser inserido pelo menos um breakpoint em alguma ocorrência de $\mathcal{X}$ em $\mathcal{G}$ ou $\mathcal{H}$, e, nesse caso, $\texttt{break}(\mathcal{X})$ será correspondente ao primeiro breakpoint inserido em alguma ocorrência de $\mathcal{X}$.

O algoritmo para a seleção dos breakpoints, descrito no Algoritmo 1, segue a seguinte abordagem. Inicialmente, são consideradas duas sequências $\mathbb{S}^0 = [\mathcal{G}]$ e $\mathbb{P}^0 = [\mathcal{H}]$, cada uma contendo um único bloco. No i-ésimo passo, as sequências $\mathbb{S}^i$ e $\mathbb{P}^i$ são geradas, incluindo-se um breakpoint nas sequências $\mathbb{S}^{i-1}$ e $\mathbb{P}^{i-1}$, conforme as regras abaixo:

- O breakpoint é inserido em uma ocorrência de um genoma $\mathcal{X} \in \mathbf{T}^{\min}_{\mathbb{S}^{i-1},\mathbb{P}^{i-1}}$.
- Se $\texttt{weight}(\mathbb{S}^{i-1}, \mathbb{P}^{i-1}, \mathcal{X}) > 0$, a ocorrência selecionada de $\mathcal{X}$ deve ser proveniente de $\mathcal{G}$.
- Se $\texttt{weight}(\mathbb{S}^{i-1}, \mathbb{P}^{i-1}, \mathcal{X}) < 0$, a ocorrência selecionada de $\mathcal{X}$ deve ser proveniente de $\mathcal{H}$.
- O breakpoint selecionado deve ser correspondente a $\texttt{break}(\mathcal{X})$.

O algoritmo prossegue até que $\texttt{weight}(\mathbb{S}^i, \mathbb{P}^i, \mathcal{X}) = 0$ para todos os genomas $\mathcal{X}$ em $\mathcal{G}$ ou $\mathcal{H}$, ou seja, até que a sequência de genomas $(\mathbb{S}^i, \mathbb{P}^i)$ seja uma partição intergênica.

A seguir, é apresentada uma análise da complexidade de tempo do Algoritmo 1, conforme discutido por [Siqueira et al. 2021]. Seja n o tamanho das sequências de entrada. Inicialmente, considera-se o tempo necessário para construir o conjunto $\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$. Usando a estrutura de dados da árvore de sufixos [Crochemore e Lecroq 2009], que pode ser construída em tempo $O(n)$, a função $\texttt{subgen}(\mathcal{G}, \mathcal{X})$ é computada em tempo $O(n)$, o que implica que a função $\texttt{weight}(\mathcal{G}, \mathcal{H}, \mathcal{X})$ também pode ser executada em tempo $O(n)$. Da mesma forma, para um genoma $\mathcal{Y}$ específico, é possível recuperar todos os genomas $\mathcal{X}$ que contêm $\mathcal{Y}$, ou seja, $\mathcal{Y} \subset \mathcal{X}$, em tempo $O(n)$.

Considerando que existem até $2n^2$ subgenomas em $\mathcal{G}$ e $\mathcal{H}$, o conjunto $\mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$ pode ser construído em tempo $O(n^3)$. Além disso, é possível armazenar os subgenomas pertencentes ao conjunto $\mathbf{T}^{\min}_{\mathbb{S}^i,\mathbb{P}^i}$ em uma árvore de sufixos, permitindo que os breakpoints

sejam atualizados em uma árvore binária de busca. Isso possibilita a recuperação de `break`($\mathcal{X}$) em tempo $O(n \log n)$.

A inicialização do Algoritmo 1 (linhas 1 a 5) leva um tempo de $O(n^3)$. O loop que vai da linha 5 até a linha 16 é repetido no máximo $O(n)$ vezes, pois existem no máximo $2n$ breakpoints, e cada iteração do loop leva no máximo $O(n \log n)$ de tempo, dado que a busca pelo breakpoint e a atualização de $\mathbf{T}^{\min}_{\mathbb{S}^i,\mathbb{P}^i}$ exigem tempo $O(n \log n)$ e linear, respectivamente.

Portanto, [Siqueira et al. 2021] diz que a complexidade total do Algoritmo 1 é $O(n^3)$.

Algorithm 1: 2k-Aproximação para MCISP

Input: Dois genomas balanceados $\mathcal{G}$ e $\mathcal{H}$
Output: Uma partição intergênica entre $\mathcal{G}$ e $\mathcal{H}$

1 **Função** `aproximacao_MCISP(`$\mathcal{G}$, $\mathcal{H}$`):`
2 $\quad \mathbb{S}^0 \leftarrow [\mathcal{G}]$
3 $\quad \mathbb{P}^0 \leftarrow [\mathcal{H}]$
4 $\quad i \leftarrow 0$
5 $\quad \mathbf{T}^{\min}_{\mathbb{S}^0,\mathbb{P}^0} \leftarrow \mathbf{T}^{\min}_{\mathcal{G},\mathcal{H}}$
6 $\quad$ **while** $(\mathbb{S}^i, \mathbb{P}^i)$ *não é uma partição intergênica* **do**
7 $\quad\quad i \leftarrow i + 1$
8 $\quad\quad \mathcal{X} \leftarrow$ um genoma de $\mathbf{T}^{\min}_{\mathcal{S}^{i-1},\mathbb{P}^{i-1}}$
9 $\quad\quad$ **if** *weight*$(\mathbb{S}^{i-1}, \mathbb{P}^{i-1}, \mathcal{X}) > 0$ **then**
10 $\quad\quad\quad \mathbb{S}^{i-1}_r \leftarrow$ um bloco de $\mathbb{S}$ contendo uma ocorrência de $\mathcal{X}$
11 $\quad\quad\quad \mathbb{S}^i \leftarrow$ sequência $\mathbb{S}^{i-1}$ com a inclusão de um breakpoint correspondente a break$(\mathcal{X})$ em $\mathbb{S}^{i-1}_r$
12 $\quad\quad\quad \mathbb{P}^i \leftarrow \mathbb{P}^{i-1}$
13 $\quad\quad$ **else**
14 $\quad\quad\quad \mathbb{P}^{i-1}_r \leftarrow$ um bloco de $\mathbb{P}$ contendo uma ocorrência de $\mathcal{X}$
15 $\quad\quad\quad \mathbb{S}^i \leftarrow \mathbb{S}^{i-1}$
16 $\quad\quad\quad \mathbb{P}^i \leftarrow$ sequência $\mathbb{P}^{i-1}$ com a inclusão de um breakpoint correspondente a break$(\mathcal{X})$ em $\mathbb{P}^{i-1}_r$
17 $\quad\quad \mathbf{T}^{\min}_{\mathbb{S}^i,\mathbb{P}^i} \leftarrow$ atualiza $\mathbf{T}^{\min}_{\mathbb{S}^{i-1},\mathbb{P}^{i-1}}$ para considerar o novo breakpoint
18 $\quad$ **return** $(\mathbb{S}^i, \mathbb{P}^i)$

3 Metodologia

3.1 Abordagem

A abordagem proposta por [Siqueira et al. 2021] para calcular a distância intergênica por transposição entre os genomas $\mathcal{G}$ e $\mathcal{H}$ baseia-se fortemente em um pré-processamento robusto, utilizando um algoritmo com fator de aproximação $2k$ para o problema de partição. Nessa metodologia, os elementos das partições não são separados por operações de transposição. Nesta seção, apresentamos duas novas heurísticas: a primeira busca uma atribuição ortóloga, estabelecendo a correspondência entre os genes repetidos e,

em seguida, aplica um algoritmo para o problema de transposição intergênica, sem duplicar qualquer gene; a segunda heurística resolve o problema de partição por meio de um método de força bruta e, em seguida, aplica o algoritmo para o cálculo da distância de transposição intergênica. Ambas as heurísticas foram avaliadas em comparação com o algoritmo de aproximação de [Siqueira et al. 2021], utilizando um banco de dados gerado a partir de um algoritmo desenvolvido pelos autores do referido artigo, com o objetivo de realizar uma avaliação preliminar da qualidade prática das soluções obtidas por sua metodologia. O principal objetivo dessa comparação foi identificar soluções mais simples que pudessem gerar resultados relevantes e eficazes.

3.2 Randomized genes

Uma das ideias iniciais consistiu em aleatorizar os genes repetidos entre os dois genomas, criando assim uma representação única para cada gene e mantendo a sua estrutura base. Essa heurística foi denominada *Randomized Partition*. A premissa por trás dessa abordagem era executar múltiplas iterações do algoritmo com a mesma entrada, permitindo que a aleatorização gerasse diferentes configurações, ou seja, diferentes atribuições ortólogas da instância inicial. Ao longo dessas execuções, o objetivo era identificar a aleatorização que resultasse no menor valor para a distância de transposição. Essa estratégia visava não apenas aumentar a eficiência do algoritmo, mas também explorar uma ampla gama de possibilidades, a fim de encontrar uma solução otimizada.

3.3 Naive Partition

Outra abordagem implementada envolveu a renomeação dos subgenomas comuns entre os dois genomas analisados. Essa heurística foi denominada *Naive Partition*. O objetivo da renomeação era reduzir o tamanho dos genomas, preservando o mesmo conjunto de dados e criando uma partição, como se cada subgenoma renomeado fosse imediatamente associado a um bloco específico de uma partição. Ao diminuir a redundância nos dados, espera-se facilitar o processamento e, consequentemente, gerar uma distância de transposição menor. Essa etapa é fundamental, pois a simplificação da estrutura dos genomas pode resultar em uma melhoria significativa na distância, especialmente em análises de grandes conjuntos de dados.

3.4 Implementação

Esta subseção mostra como foi feita a implementação dos algoritmos. Ambos os algoritmos foram feitos utilizando a linguagem Python. Os dados de entrada dos algoritmos foram gerados a partir do código que foi disponibilizado pelos autores no Github. Os genomas utilizados inicialmente não são co-caudais. No entanto, antes de realizar o cálculo da distância de transposição, o algoritmo os transforma em genomas co-caudais por meio do processo denominado capping. Para a execução dos algoritmos responsáveis pela geração do banco de dados, cálculo das partições e das distâncias de transposição (implementados em c++ pelos autores), conforme disponibilizados por [Siqueira et al. 2021], utilizou-se uma máquina virtual *Oracle VirtualBox*. A máquina foi configurada com um processador *AMD Ryzen™ 5 5600*, alocado com 4 núcleos de

processamento e 8 GB de memória RAM, utilizando o sistema operacional Ubuntu 24.04.1 LTS.

O primeiro algoritmo implementado, denominado Randomized genes, propõe um método simples para substituir genes repetidos por valores aleatórios exclusivos, de forma a preservar a integridade do conjunto genômico, enquanto realiza o mapeamento dessas repetições. O objetivo principal é estabelecer uma atribuição ortóloga específica após um número determinado de execuções, visando otimizar a distância por transposição e garantindo que a correspondência entre os genes repetidos contribua para uma redução efetiva da distância de transposição.

O algoritmo 2 identifica os genes repetidos e os modifica por meio de um processo de substituição. Para isso, é desenvolvida uma função que conta a frequência de cada gene e, em seguida, gera um vetor aleatório para cada rótulo que aparece mais de uma vez, com o objetivo de substituir todos os valores repetidos desse rótulo.

Uma vez que os vetores de substituição são gerados, a substituição dos genes repetidos é feita percorrendo as listas S e P e substituindo cada gene repetido pelo valor aleatório correspondente, garantindo que a substituição seja realizada de forma controlada e sem duplicação. O resultado dessa substituição é uma nova lista de genes, com os valores repetidos mapeados para valores aleatórios. Por fim, o algoritmo 2 retorna os genomas iniciais porém com rótulos únicos, preservando as regiões intergênicas.

A seguir, realizamos uma análise da complexidade do algoritmo 2, levando em conta as operações principais.

O algoritmo consiste nas seguintes etapas:

1. **Divisão dos Genomas** (Linha 3): O algoritmo começa dividindo os genomas $\mathcal{G}$ e $\mathcal{H}$ nas listas $S, \breve{S}, P, \breve{P}$. Essa operação de divisão é linear em relação ao tamanho dos genomas, ou seja, tem complexidade $O(n)$, onde n é o tamanho de $\mathcal{G}$ e $\mathcal{H}$.

2. **Geração de Vetores Aleatórios** (Linha 4): Para cada gene repetido em S, o algoritmo cria um vetor de genes aleatórios exclusivos. A complexidade dessa etapa depende do número de genes repetidos em S, que pode ser no máximo $O(n)$ no pior caso, onde n é o número de genes em S. Para cada gene repetido, o algoritmo gera um vetor aleatório, o que tem complexidade $O(k)$, onde k é a quantidade de genes repetidos.

3. **Substituição dos Genes Repetidos** (Linha 5): Assim que os vetores aleatórios foram gerados, os genes repetidos nas listas S e P são substituídos pelos genes dos vetores aleatórios, enquanto as listas $\breve{S}$ e $\breve{P}$ permanecem inalteradas. A operação de substituição é linear em relação ao tamanho de S e P, ou seja, tem complexidade $O(n)$, onde n é o número de genes em S e P.

4. **Retorno do Resultado** (Linha 6): Finalmente, o algoritmo retorna os genomas com rótulos únicos. Essa operação é constante, ou seja, tem complexidade $O(1)$, pois apenas realiza o retorno dos dados processados.

A complexidade total do algoritmo é determinada pela soma das complexidades das etapas descritas. A operação mais cara do algoritmo é a geração dos vetores aleatórios para os genes repetidos. No pior caso, se houver n genes repetidos, o algoritmo terá uma complexidade de $O(n)$ para gerar vetores para cada gene repetido e $O(n)$ para

substituir os genes repetidos nas listas. Assim, a complexidade total do algoritmo é dada por:

$$O(n) + O(n) + O(n) = O(n)$$

Portanto, a complexidade do algoritmo é $O(n)$, onde n é o número de genes nos genomas $\mathcal{G}$ e $\mathcal{H}$. Isso significa que o algoritmo possui uma complexidade linear no número de genes dos genomas de entrada.

Algorithm 2: randomized_genes

Input: Dois genomas balanceados $\mathcal{G}$ e $\mathcal{H}$
Output: Dois genomas balanceados $\mathcal{G}$ e $\mathcal{H}$ com rótulos únicos

1 **Função** `randomized_genes(`$\mathcal{G}$, $\mathcal{H}$`):`
2 Genomas_rótulos_únicos ← [ ]
3 $S, \breve{S}, P, \breve{P}$ ← dividir os genomas $\mathcal{G}$ e $\mathcal{H}$ em listas
4 *vetores_aleatorios* ← para cada gene repetido de S, cria um vetor de genes aleatórios exclusivos
5 *Genomas_rótulos_únicos* ← as listas S e P tem seus genes repetidos substituídos pelos genes dos vetores_aleatorios, e as listas $\breve{S}$ e $\breve{P}$ são inalteradas
6 **return** *Genomas_rótulos_únicos*

Já o algoritmo 3 chamado de Naive Partition tem como objetivo focar na identificação e renomeação de subgenomas comuns entre dois genomas. A principal estratégia adotada é realizar o particionamento dos genomas, intercalando genes e regiões intergênicas de maneira a facilitar a comparação e a busca por subgenomas semelhantes.

A principal diferença entre o algoritmo Partition, proposto por [Siqueira et al. 2021], e o Naive Partition reside no método de *força bruta* utilizado para o particionamento dos genomas por meio de subgenomas comuns. Enquanto o algoritmo com fator de aproximação $2k$ para a partição, proposto pelos autores, se baseia em estratégias de aproximação, o algoritmo de força bruta implementado neste trabalho realiza uma busca exaustiva e direta por todas as possíveis configurações de tamanho 3, a fim de identificar subgenomas comuns entre os dois genomas, simplificando assim a estrutura dos dados.

Em resumo, a principal inovação do nosso algoritmo reside na combinação da abordagem de força bruta com a identificação de subgenomas comuns, o que permite uma redução no cálculo da distância por transposição, embora implique em um aumento no tempo de execução.

O algoritmo inicia com a intercalação dos genes e regiões intergênicas de cada genoma. Após essa etapa, o objetivo é buscar por subgenomas comuns entre os dois genomas, considerando subgenomas de tamanho fixo (3 elementos). Para isso, armazena subgenomas válidos do primeiro genoma em um dicionário, onde a chave é o subgenoma e o valor são suas posições iniciais e finais. Em seguida, percorre o segundo genoma à procura de correspondências. Quando encontra um subgenoma comum, o algoritmo o renomeia nos dois genomas, substituindo-o por um número único, maior do que os números já presentes nos genes. Essa renomeação ocorre sem mover o subgenoma, preservando sua identidade.

O processo de busca e renomeação é repetido até que não sejam mais encontrados subgenomas válidos para renomear. Cada subgenoma identificado é substituído por um rótulo único. Ao final da execução, o algoritmo retorna os genomas com os subgenomas iguais entre eles renomeados.

A seguir, realizamos uma análise detalhada da complexidade do algoritmo 3 com base em suas operações principais.

O algoritmo consiste em diversas etapas principais:

1. **Intercalação dos Genomas** (Linhas 2-3): O algoritmo começa intercalando os genomas S e $\breve{S}$, e P e $\breve{P}$, para formar os genomas $\mathcal{G}$ e $\mathcal{H}$. Esta operação é linear, ou seja, tem complexidade $O(n)$, onde n é o tamanho dos genomas.

2. **Busca por Subgenomas Comuns** (Linhas 4-5): O laço `while` (linha 8) é responsável por buscar subgenomas comuns de tamanho 3 entre os genomas $\mathcal{G}$ e $\mathcal{H}$. Para cada subgenoma de tamanho 3 em $\mathcal{G}$, o algoritmo verifica se ele aparece em $\mathcal{H}$, o que exige uma comparação de $O(n)$ tempo, resultando em uma complexidade de $O(n^2)$ para esta etapa.

3. **Renomeação de Subgenomas** (Linhas 6-7): Quando um subgenoma comum é encontrado, ele é renomeado com um rótulo único. Este processo tem complexidade $O(1)$, pois é uma simples substituição do rótulo do subgenoma.

4. **Interrupção do Laço** (Linhas 9): O laço `while` continua até que não haja mais subgenomas comuns. A quantidade de iterações do laço depende do número de subgenomas comuns encontrados. No pior caso, o número de iterações pode ser $O(n)$, onde n é o número de subgenomas a serem encontrados.

A complexidade total do algoritmo é determinada pela combinação das operações descritas. A parte mais cara do algoritmo é a busca por subgenomas comuns, que tem complexidade $O(n^2)$, e o laço `while` que pode iterar até $O(n)$ vezes no pior caso. Assim, a complexidade total do algoritmo é dada por:

$$O(n^2) \times O(n) = O(n^3)$$

Portanto, a complexidade do algoritmo é $O(n^3)$ no pior caso, onde n é o tamanho dos genomas $\mathcal{G}$ e $\mathcal{H}$. Este crescimento cúbico se deve à combinação da busca quadrática por subgenomas comuns e à iteração do laço `while`.

3.5 Resultados Esperados

As duas heurísticas propostas, **Randomized Genes** e **Naive Partition**, visam não apenas otimizar o desempenho do algoritmo, mas também estabelecer uma base robusta para melhorias subsequentes. O algoritmo **Randomized Genes** busca gerar genomas distintos em cada execução. A fundamentação dessa abordagem reside na ideia de que, ao executar o algoritmo múltiplas vezes, há uma probabilidade de que, em algumas execuções, a configuração dos genes converja para uma solução mais otimizada, resultando em uma redução na distância de rearranjo por transposição.

Por outro lado, o algoritmo Naive Partition atua de maneira complementar. Ao agrupar subgenomas comuns, o tamanho do genoma é reduzido, o que, por sua vez,

Algorithm 3: Naive Partition

Input: Dois genomas balanceados $\mathcal{G}$ e $\mathcal{H}$
Output: Dois genomas balanceados $\mathcal{G}$ e $\mathcal{H}$ menores e com rótulos únicos

```
Função naive_partition(S, Š, P, Ṕ):
    G ← S e Š intercalados
    H ← P e Ṕ intercalados
    while existir subgenomas comuns entre G e H do
        G^{i,j}, i, k ← o primeiro subgenoma comum G^{i,j} de tamanho 3 que
          encontrar entre G e H e sua posição em ambos os genomas
        if G^{i,j} ≠ None then
            renomear o subgenoma G^{i,j} nas posições i e k respectivamente nos
              genomas G e H, onde é inserido um rótulo único
        else
            print "Nenhum novo subgenoma comum encontrado. Encerrando."
            break
    return (G, H)
```

leva a uma diminuição na complexidade do problema. Com a redução do tamanho do genoma, espera-se que a distância de rearranjo por transposição também diminua, tornando o problema menos complexo e facilitando a identificação de padrões ou soluções que antes poderiam ser difíceis de detectar.

3.6 Materiais Disponíveis

Os algoritmos desenvolvidos e os conjuntos de dados gerados ao longo deste estudo estão disponíveis para consulta e download através do repositório público a seguir: https://github.com/LuizHenique1308/Estudo-de-heuristicas-para-distancia-por-transposicao-considerando-regioes-intergenicas-e-repeticao.

4 Resultados experimentais

O conjunto de instâncias utilizado neste estudo foi gerado a partir de uma única instância inicial. Os parâmetros foram definidos de forma aleatória. Os valores escolhidos para essa instância inicial foram: 50 operações, um alfabeto de 10 rótulos e 500 genes, sendo essa quantidade de genes determinada sem um padrão específico, para simular uma situação genérica. Embora 500 genes possam não refletir diretamente a realidade de alguns cenários biológicos mais complexos, essa escolha foi feita para fornecer um ponto de partida controlado e simplificado, permitindo a avaliação do desempenho do algoritmo em diferentes configurações. Assim, os resultados podem ser extrapolados para outros cenários com diferentes tamanhos e complexidades. A partir dessa instância, o algoritmo Randomized Genes gerou outras 50 instâncias, correspondendo a 50 atribuições ortólogas aleatórias para a instância inicial. Após isso, cada uma das 50 atribuições ortólogas foi utilizada para calcular a distância por transposição.

Além disso, foi testada também a aplicação do algoritmo Partition logo após a

execução do Randomized genes para as 50 atribuições ortólogas geradas, com o intuito de explorar o efeito da aleatorização dos genes repetidos. A ideia é que, a cada nova aleatorização, uma nova atribuição ortológica seria gerada, o que poderia resultar em genes posicionados de maneira estratégica para formar blocos mais facilmente. Isso, por sua vez, poderia diminuir o tamanho dos genomas através do processo de Partition e também diminuir a distância de transposição.

A Figura 13 apresenta um diagrama de caixas das distâncias dos 50 testes realizados com o algoritmo Randomized genes, utilizando sua saída como instância para calcular a distância por transposição e também a combinação de usar o Partition logo em seguida.

A média das distâncias de transposição obtidas apenas com o algoritmo Randomized Genes foi de 500,4, enquanto a combinação do algoritmo Randomized Genes com o método Partition resultou em uma distância média de transposição também de 500,4. Esses resultados são particularmente interessantes, pois os valores obtidos ficaram notavelmente próximos do número total de genes da instância inicial, que era 500. Isso indica que, em ambas as abordagens, os algoritmos não conseguiram transformar de maneira significativa a estrutura dos genes durante o processo de transposição.

A proximidade desses valores com o número inicial de genes (500) indica que, apesar da aplicação das estratégias propostas, os métodos utilizados não foram capazes de alterar a sequência genética de maneira relevante.

É ainda mais preocupante quando comparamos esses resultados com o desempenho obtido ao se utilizar apenas o método Partition, que obteve uma distância de transposição média de 315. Ou seja, a combinação do Randomized Genes com o Partition resultou em uma distância maior, o que sugere que a aleatorização dos genes não ajudou na transposição, podendo até ter prejudicado o processo devido à falta de diversidade nas soluções geradas. Essa degradação nos resultados pode ser atribuída à falta de diversidade nas soluções geradas pelo Randomized Genes, que, ao ser combinado com o Partition, não conseguiu superar as limitações impostas pela aleatoriedade das escolhas dos genes. O único aspecto positivo foi o tempo do algoritmo, que apresentou uma média de 0,00312 segundos. Dessa forma, fica claro que a estratégia de combinar o Randomized Genes com o Partition não foi eficaz.

Já para o algoritmo Naive Partition, durante os experimentos, foram variados três parâmetros das instâncias: o número de operações, que foi ajustado de 50 a 150; o tamanho do alfabeto, que variou entre 10 e 100; e o número de genes, que oscilou entre 50 e 2000.

Para a realização dos testes, foram utilizados o algoritmo Partition e o algoritmo de cálculo de distância baseado em transposição, conforme disponibilizados pelos autores.

A metodologia de testes foi estruturada da seguinte forma: inicialmente, as instâncias foram geradas e, em seguida, três tipos distintos de testes foram realizados.

- **Teste 1**: Avaliar o tempo de execução e a distância de transposição ao aplicar o algoritmo Partition na instância inicial.
- **Teste 2**: Obter os mesmos dados, mas usando o algoritmo Naive Partition.
- **Teste 3**: Investigar se a combinação dos algoritmos Naive Partition e Partition traz vantagens em termos de tempo de execução e distância de transposição.

Para o **Teste 3**, a instância foi primeiramente processada pelo Naive Partition, e o

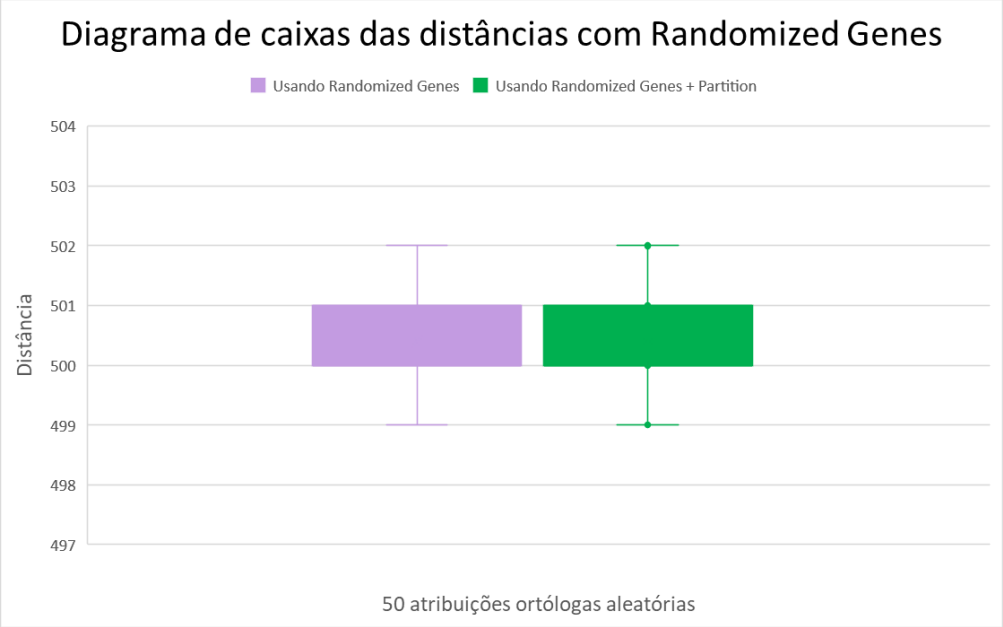

Figura 13: Diagrama de caixas das distâncias para o problema ITD com o uso do Randomized Genes

resultado foi então passado para o algoritmo Partition, no qual, após o processamento, foi medida a distância de transposição.

A Tabela 1 apresenta os resultados dos testes realizados com os algoritmos Partition e Naive Partition, os quais realizam um pré-processamento nos dados.

A Figura 14 ilustra um diagrama de caixas, que revela que as distâncias de transposição obtidas nos três testes realizados são bastante semelhantes, indicando consistência nos resultados.

Ao analisar os dados da Tabela 1, observa-se que o algoritmo Naive Partition apresentou um desempenho notavelmente bom em termos de distância de transposição, especialmente considerando sua simplicidade como heurística. De maneira geral, os resultados obtidos com o Naive Partition foram bastante próximos daqueles alcançados pelo algoritmo Partition proposto pelos autores. Em situações específicas, especialmente quando o alfabeto utilizado é pequeno (como no caso de um alfabeto de 10 rótulos), o Naive Partition demonstrou uma melhoria interessante na distância de transposição em comparação com o Partition. Esse comportamento sugere que, para instâncias menores, a utilização do Naive Partition pode ser uma escolha vantajosa.

As Figuras 15, 16 e 17 evidenciam uma vantagem em termos de distância de transposição tanto para o algoritmo Naive Partition quanto para a combinação de Naive Partition e Partition, mesmo com variações no número de operações para construir a instância inicial. Observa-se que, à medida que a quantidade de genes aumenta, mantendo o alfabeto constante com 10 rótulos, há uma melhoria na distância de transposição em comparação com o algoritmo Partition. Esse comportamento sugere que, para instâncias com um alfabeto pequeno, a combinação dos algoritmos foi mais eficaz na redução das distâncias de transposição, ao contrário do que ocorreu com alfabetos maiores.

À medida que o tamanho do alfabeto cresce, como no caso em que o alfabeto pos-

			Teste 1		Teste 2		Teste 3	
			Usando Partition		Usando Naive Partion		Usando Naive Partion + Partition	
OP	|X|	|∑|	Distância	Tempo	Distância	Tempo	Distância	Tempo
50	50	10	40	0,028	40	0,037	41	0,043
50	50	100	43	0,041	42	0,040	43	0,058
100	50	10	48	0,036	48	0,020	48	0,031
100	50	100	48	0,030	48	0,039	48	0,049
150	50	10	49	0,363	48	0,034	49	0,053
150	50	100	49	0,028	49	0,029	49	0,033
50	200	10	148	0,393	147	0,921	148	1,293
50	200	100	139	0,408	138	0,863	139	1,033
100	200	10	155	0,559	154	0,890	155	1,103
100	200	100	155	0,367	155	0,923	155	1,982
150	200	10	169	0,513	169	0,899	169	1,041
150	200	100	172	0,676	172	0,936	172	1,230
50	500	10	325	3,514	320	4,713	319	6,005
50	500	100	310	3,501	309	9,035	310	11,979
100	500	10	315	3,144	316	5,242	316	6,181
100	500	100	329	3,609	328	7,689	329	9,122
150	500	10	335	3,446	328	4,091	328	5,012
150	500	100	367	3,247	366	12,273	367	14,302
50	1000	10	485	11,359	479	29,321	475	34,950
50	1000	100	485	11,485	484	56,961	485	59,787
100	1000	10	495	12,612	473	28,625	469	35,102
100	1000	100	564	12,124	561	58,349	564	65,720
150	1000	10	603	11,584	588	35,419	586	41,406
150	1000	100	555	11,051	554	60,665	555	64,020
50	2000	10	780	54,466	752	162,590	741	199,209
50	2000	100	914	53,274	914	349,780	914	388,274
100	2000	10	1091	70,069	1041	195,587	1036	237,603
100	2000	100	1191	69,691	1189	412,559	1189	452,967
150	2000	10	1152	69,590	1094	217,578	1082	252,467
150	2000	100	1138	69,554	1138	420,027	1139	448,269

Tabela 1: Tempo de execução dos algoritmos Partition, Naive Partition e a combinação de ambos, juntamente com a distância de transposição calculada para o problema ITD; OP representa o número de operações realizadas para criar a instância inicial; $|X|$ é o número de genes em cada genoma; $|\sum|$ é o tamanho do alfabeto dos genomas

Média dos tempo de execução			
|X|	Usando Partition	Usando Naive Partion	Usando Naive Partion + Partition
50	0,088	0,033	0,044
200	0,486	0,905	1,281
500	3,410	7,174	8,767
1000	11,703	44,890	50,164
2000	64,441	293,020	329,798

Tabela 2: Média dos tempos de execução em segundos dos algoritmos Partition, Naive Partition e sua combinação, variando o tamanho dos genes de 50 a 2000 elementos; $|X|$ é o número de genes em cada genoma

sui 100 rótulos, ilustrado na Figura 15, o desempenho do algoritmo Naive Partition se aproxima do algoritmo Partition. Isso é particularmente notável em relação à distância de rearranjo, na qual ambos os algoritmos apresentam resultados muito semelhantes, ficando praticamente empatados. Esse comportamento sugere que, para alfabetos maiores, a simplicidade do Naive Partition pode se tornar um fator limitante. A principal dificuldade do Naive Partition em instâncias com grandes alfabetos pode ser atribuída à sua abordagem direta, que carece de estratégias mais sofisticadas para lidar com a maior diversidade de rótulos.

Quanto ao tempo de execução, o algoritmo Partition, devido à sua abordagem mais

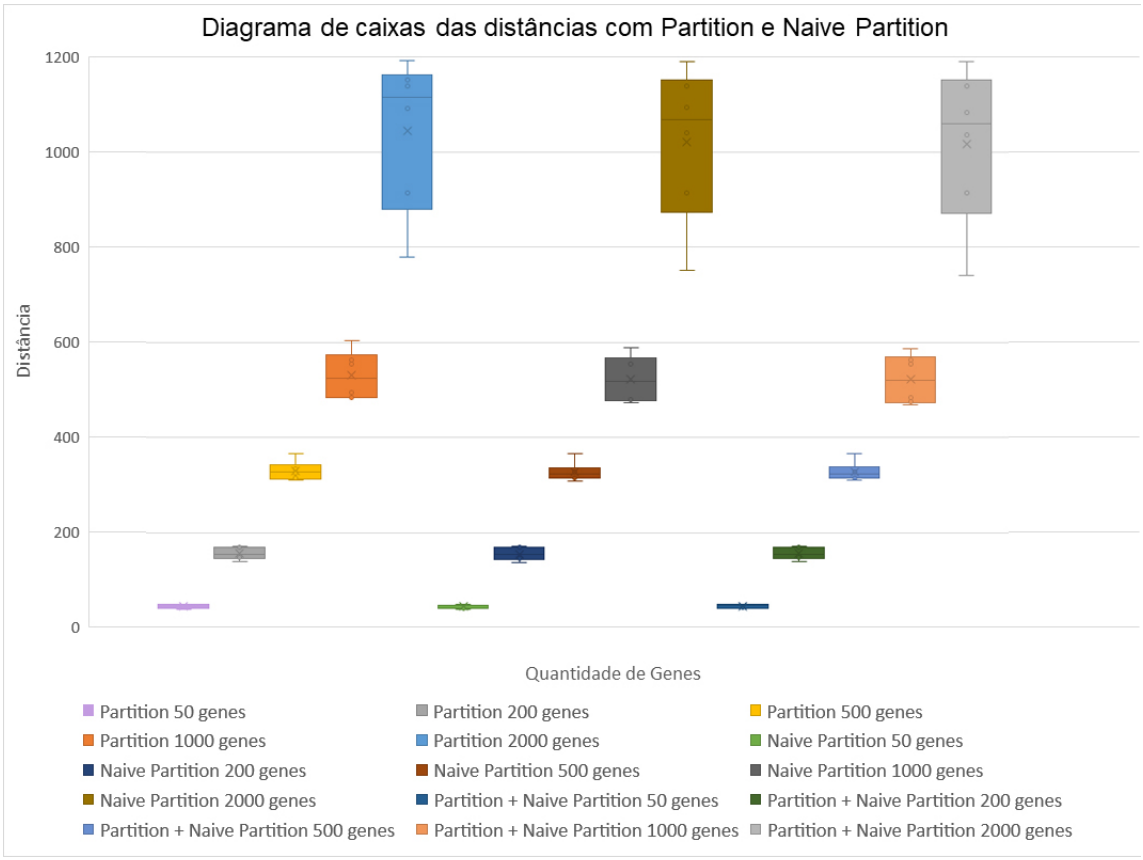

Figura 14: Diagrama de caixas das distâncias para o problema ITD com o uso do Partition, Naive Partition e ambos combinados

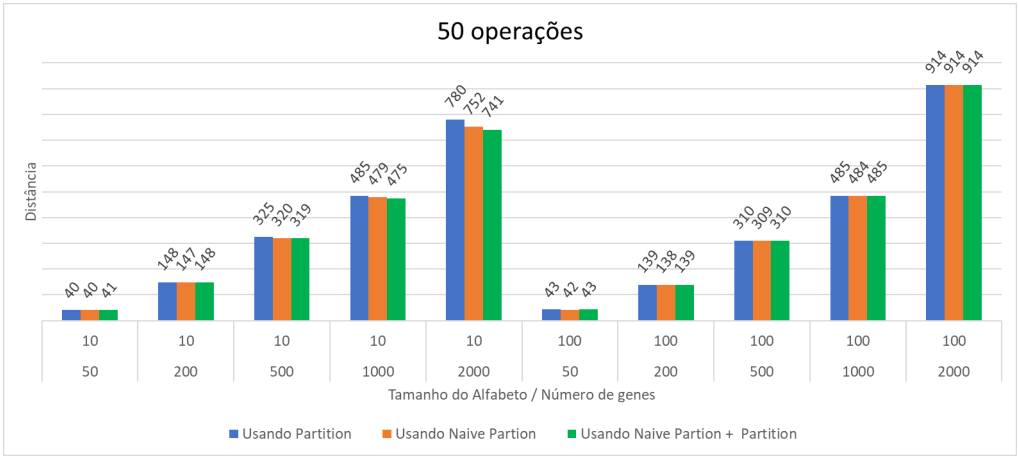

Figura 15: Distâncias para o problema ITD com o uso do Partition, Naive Partition e ambos combinados, utilizando o número fixo de 50 operações

robusta e otimizada, apresentou um desempenho superior na maioria dos casos, conforme ilustrado na Tabela 2. Por outro lado, o algoritmo Naive Partition, apesar de sua simplicidade, demonstrou uma desvantagem significativa em termos de tempo de execução. Isso se deve à sua estrutura baseada em força bruta, que, embora eficaz para instâncias pequenas, não escala bem para problemas mais complexos, resultando

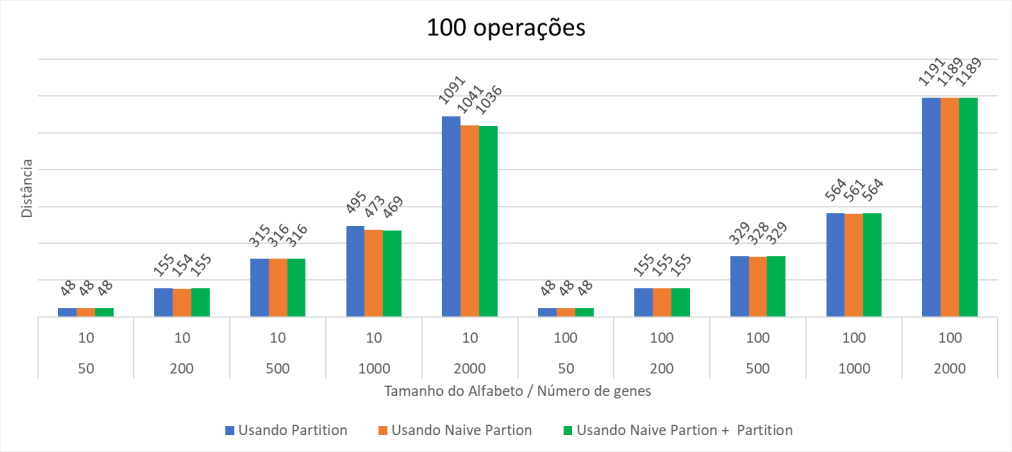

Figura 16: Distâncias para o problema ITD com o uso do Partition, Naive Partition e ambos combinados, utilizando o número fixo de 100 operações

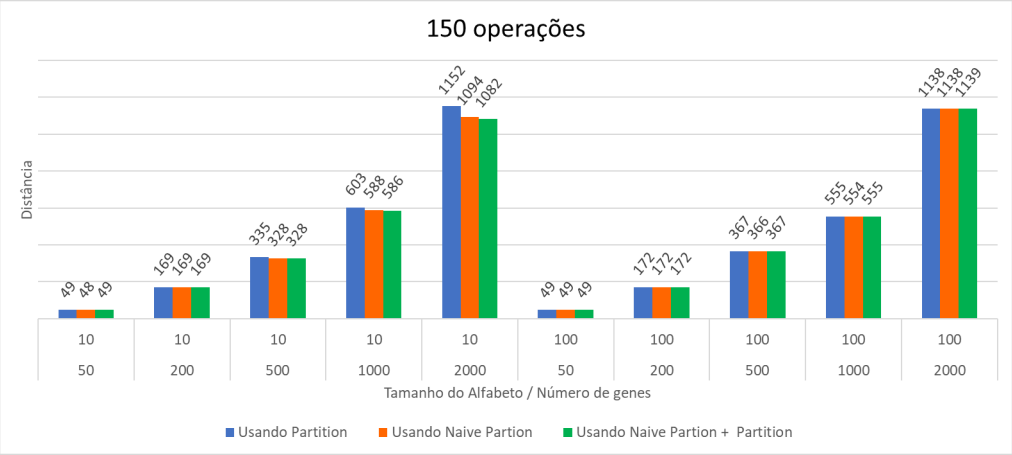

Figura 17: Distâncias para o problema ITD com o uso do Partition, Naive Partition e ambos combinados, utilizando o número fixo de 150 operações

em tempos de execução mais extensos. Esse comportamento é mais evidente quando comparamos o tempo de execução do Naive Partition em execuções distintas, conforme mostrado na Tabela 1, onde, em instâncias com um alfabeto maior (100 rótulos), o tempo de execução foi superior ao tempo registrado para o alfabeto com 10 rótulos.

Uma desvantagem de utilizar a combinação dos dois algoritmos é que o tempo de execução se torna ainda mais elevado, o que sugere novamente que essa abordagem é menos eficiente em instâncias maiores. Esse fator indica que a combinação dos algoritmos é mais vantajosa quando aplicada a conjuntos de dados menores, com um número reduzido de genes, onde o custo computacional adicional é menos impactante.

Além disso, em ambos os casos, a comparação entre os resultados do Naive Partition, do Partition e da combinação dos dois algoritmos mostrou que as distâncias de transposição permanecem bastante semelhantes, mesmo utilizando abordagens distintas para o pré-processamento das instâncias antes do cálculo da distância.

5 Conclusão

Os resultados obtidos a partir da implementação das heurísticas Randomized Genes e Naive Partition demonstraram variações notáveis para solução do problema ITD. Neste estudo, a transposição, essencial para rearranjos genômicos, foi abordada com duas heurísticas distintas: o algoritmo Naive Partition, que obteve resultados consistentes, especialmente em cenários com um grande número de genes e um alfabeto pequeno, e o algoritmo Randomized Genes, que não apresentou os ganhos esperados e, em alguns casos, até prejudicou os resultados quando combinado com o Partition. Isso sugere que, para o problema em questão, a aleatorização não foi capaz de oferecer melhorias significativas, enquanto o Naive Partition, com sua abordagem simples e direta, se mostrou mais eficiente, especialmente quando aplicado a genomas com alfabetos reduzidos.

Quando aplicada à renomeação de subgenomas, a heurística Naive Partition obteve resultados semelhantes aos do algoritmo de aproximação com fator de aproximação $2k$ para o MCISP. Em algumas instâncias, ela até superou o desempenho dos autores em termos de distância de transposição. Embora o algoritmo tenha um tempo de execução relativamente alto, sua capacidade de alcançar bons resultados de distância de transposição demonstra a eficácia da abordagem, apesar de suas limitações. Essa proximidade nos resultados sugere que o algoritmo original possui características robustas, que podem ser refinadas ou até mesmo integradas com as inovações propostas, abrindo caminho para melhorias futuras.

A natureza NP-difícil dos problemas ITD e MCISP torna essencial o desenvolvimento de heurísticas adequadas para lidar com a complexidade dos genomas reais, e os resultados obtidos aqui fornecem insights valiosos para essa direção.

Esses resultados ressaltam a necessidade de continuar explorando novas metodologias e algoritmos no campo da bioinformática. A pesquisa demonstrou que, mesmo com algoritmos simples, é possível alcançar bons resultados em termos de distância de transposição. A continuidade desse trabalho é essencial, pois oferece oportunidades para aprimorar as heurísticas existentes e propor novas abordagens que possam melhorar a eficiência dos algoritmos de análise de genomas.

Por fim, a experiência adquirida neste estudo serve como base para futuras investigações, abrindo caminho para o desenvolvimento de técnicas mais eficientes e inovadoras que podem impactar significativamente a análise e manipulação de genomas, com o objetivo de enfrentar os desafios impostos por problemas complexos e NP-difíceis na bioinformática.

Referências

[Blin, Fertin e Chauve 2004]BLIN, G.; FERTIN, G.; CHAUVE, C. The breakpoint distance for signed sequences. In: KING'S COLLEGE LONDON PUBLICATIONS. *1st conference on algorithms and computational methods for biochemical and evolutionary networks (CompBioNets' 04)*. [S.l.], 2004. v. 3, p. 3–16.

[Bulteau, Fertin e Rusu 2012]BULTEAU, L.; FERTIN, G.; RUSU, I. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, SIAM, v. 26, n. 3, p. 1148–1180, 2012.

[Crochemore e Lecroq 2009]CROCHEMORE, M.; LECROQ, T. Suffix tree. In: *Encyclopedia of Database Systems*. US: Springer, 2009. p. 2876–2880.

[Doolittle 2013]DOOLITTLE, W. F. Is junk dna bunk? a critique of encode. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 110, n. 14, p. 5294–5300, 2013.

[Elias e Hartman 2006]ELIAS, I.; HARTMAN, T. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, IEEE, v. 3, n. 4, p. 369–379, 2006.

[Futuyma 2013]FUTUYMA, D. J. The evolution of evolutionary ecology. *Israel Journal of Ecology & Evolution*, Taylor & Francis, v. 59, n. 4, p. 172–180, 2013.

[Goldstein, Kolman e Zheng 2004]GOLDSTEIN, A.; KOLMAN, P.; ZHENG, J. Minimum common string partition problem: Hardness and approximations. In: SPRINGER. *International Symposium on Algorithms and Computation*. [S.l.], 2004. p. 484–495.

[Haubold e Wiehe 2004]HAUBOLD, B.; WIEHE, T. Comparative genomics: methods and applications. *Naturwissenschaften*, Springer, v. 91, p. 405–421, 2004.

[Kolman e Waleń 2007]KOLMAN, P.; WALEŃ, T. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, Elsevier, v. 155, n. 3, p. 327–336, 2007.

[Oliveira et al. 2020]OLIVEIRA, A. R. et al. A 3.5-approximation algorithm for sorting by intergenic transpositions. In: SPRINGER. *International Conference on Algorithms for Computational Biology*. [S.l.], 2020. p. 16–28.

[Siqueira et al. 2021]SIQUEIRA, G. et al. Approximation algorithm for rearrangement distances considering repeated genes and intergenic regions. *Algorithms for Molecular Biology*, Springer, v. 16, n. 1, p. 21, 2021.