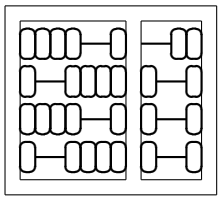
UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO



Proposta de Tese de Doutorado

Algoritmos para o Problema da Ordenação por Rearranjo

Aluno: Gustavo Rodrigues Galvão

Orientador: Prof. Dr. Zanoni Dias

CAMPINAS

2014

Resumo

Ao longo da evolução, mutações globais podem alterar a ordem dos genes de um genoma. Tais mutações são chamadas de eventos de rearranjo. Em Rearranjo de Genomas, estimamos a distância evolutiva entre dois genomas calculando-se a distância de rearranjo entre eles, que é o tamanho da menor sequência de eventos de rearranjo que transforma um genoma no outro. Representando genomas como permutações, nas quais os genes aparecem como elementos, a distância de rearranjo pode ser obtida resolvendo-se o problema combinatório de ordenar uma permutação utilizando o menor número de eventos de rearranjo. Este problema, que é referido como Problema da Ordenação por Rearranjo, varia de acordo com os tipos de eventos de rearranjo considerados. Neste doutorado, focaremos nosso estudo em dois tipos de eventos: reversões e transposições. Variações do Problema da Ordenação por Rearranjo que consideram esses eventos têm se mostrado difíceis de serem resolvidas otimamente, por isso a maior parte dos algoritmos propostos são aproximados. Nossa proposta é estudar algumas dessas variações, buscando desenvolver algoritmos aproximados ou heurísticos que superem os existentes.

Sumário

1 Introdução

Um dos desafios modernos da Ciência é tentar desvendar como ocorreu a evolução das espécies. Dado que a evolução pode ser vista como um processo de ramificação, em que novas espécies surgem a partir de modificações ocorridas nos organismos vivos, o estudo da história evolutiva de um grupo de espécies é comumente realizado por meio da construção de árvores, nas quais os nós representam as espécies e as arestas representam relações evolutivas. As relações evolutivas entre um determinado grupo de espécies são chamadas de filogenia, enquanto que as árvores são chamadas de árvores filogenéticas.

Devido a impossibilidade de descobrir a história evolutiva verdadeira, os cientistas propõem métodos de inferí-la. Tal inferência pode ser feita baseando-se em diferentes tipos de dados, desde geográficos, ecológicos e morfológicos até moleculares, como o DNA. Dados moleculares possuem a grande vantagem de serem exatos e reprodutíveis, pelo menos dentro de erros experimentais, além de serem fáceis de obter [27, Capítulo 12]. Cada nucleotídeo de uma sequência de DNA é, por si próprio, uma característica bem definida, enquanto que um dado morfológico, por exemplo, precisa ser codificado em uma característica, o que acarreta problemas de interpretação, discretização e etc. [27, Capítulo 12].

Dentre os métodos de inferência filogenética existentes baseados em dados moleculares, iremos focar naqueles que se baseiam na noção de distância evolutiva entre as espécies. Tais métodos constroem a árvore filogenética relativa a um grupo de espécies da seguinte maneira: primeiramente, a distância evolutiva entre as espécies é determinada de tal maneira a gerar uma matriz de distâncias M tal que a entrada $M_{i,j}$ contém a distância evolutiva entre as espécies i e j; depois, a árvore filogenética é computada a partir dessa matriz utilizando-se um algoritmo específico, como o *Neighbor-Joining* [42]. Para mais detalhes a respeito de outros métodos de inferência filogenética, incluindo uma análise detalhada de cada um deles, recomendamos a leitura do livro de Gascuel [27, Capítulo 12].

Olhando para a maneira como uma árvore filogenética é construída por esses métodos, podemos notar que um ponto chave está na abordagem utilizada para determinar a distância evolutiva entre as espécies. Uma abordagem bem aceita é a dada pela área de Rearranjo de Genomas. Pesquisadores dessa área propõem estimar a distância evolutiva entre duas espécies utilizando a distância de rearranjo entre seus genomas, que é o tamanho da menor sequência de mutações globais, chamadas de eventos de rearranjo, que transforma um genoma no outro.

Representando genomas como permutações, nas quais os genes aparecem como elementos, a distância de rearranjo pode ser obtida resolvendo-se o problema combinatório de ordenar uma permutação utilizando o menor número de eventos de rearranjo. Este problema, que é

referido como Problema da Ordenação por Rearranjo, varia de acordo com os tipos de eventos de rearranjo considerados. Neste doutorado, focamos nosso estudo em dois tipos de eventos: reversões e transposições. Variações do Problema da Ordenação por Rearranjo que consideram esses eventos têm se mostrado difíceis de serem resolvidas otimamente, por isso a maior parte dos algoritmos propostos são aproximados. Nossa proposta é estudar algumas dessas variações, buscando desenvolver algoritmos aproximados ou heurísticos que superem os existentes.

O restante desta proposta está organizado da seguinte forma. A Seção 1.1 apresenta os conceitos básicos, enquanto que a Seção 1.2 revisa as variações do Problema da Ordenação por Rearranjo relevantes a esta proposta. A Seção 2 descreve os objetivos do trabalho a ser realizado. A Seção 3 contém uma revisão mais aprofundada das variações do Problema da Ordenação por Rearranjo que pretendemos atacar. Por fim, a Seção 4 apresenta o plano de trabalho e o cronograma para a realização desta proposta.

1.1 Conceitos Básicos

Esta seção é destinada à formalização dos conceitos básicos concernentes a esta proposta. A maior parte do conteúdo apresentado é baseado no livro de Fertin *et al.* [21], que consideramos ser a principal referência na área de Rearranjo de Genomas.

1.1.1 Genomas como Permutações

Na literatura de Rearranjo do Genomas, podemos encontrar muitas abordagens para modelar genomas. Basicamente, o que difere uma abordagem de outra são as suposições feitas sobre os genomas. Assumindo que os genomas consistem-se de um único cromossomo linear, partilham o mesmo conjunto de genes e não contêm genes duplicados, podemos modelá-los como permutações de números inteiros (com ou sem sinal), nas quais cada número representa um gene do genoma.

Uma permutação sem sinal π é uma função bijetora sobre o conjunto $\{1, 2, \ldots, n\}$. Uma notação clássica para permutações sem sinal é a chamada notação em duas colunas

$$\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ \pi_1 & \pi_2 & \ldots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \ldots, n\}$ para $1 \leq i \leq n$. Esta notação indica que a imagem do elemento $i \in \{1, 2, \ldots, n\}$ é π_i ou, em outras palavras, $\pi(1) = \pi_1$, $\pi(2) = \pi_2$, $\ldots$, $\pi(n) = \pi_n$. Em Rearranjo de Genomas, geralmente é utilizada a notação chamada de notação em uma coluna $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, a qual possui o mesmo significado. Nós dizemos que a permutação π possui tamanho n.

Uma permutação com sinal π é uma função bijetora sobre o conjunto $\{-n, \ldots, -2, -1, 1, 2, \ldots, n\}$. Sua notação em duas colunas é dada por

$$\pi = \begin{pmatrix} -n & \ldots & -2 & -1 & 1 & 2 & \ldots & n \\ -\pi_n & \ldots & -\pi_2 & -\pi_1 & \pi_1 & \pi_2 & \ldots & \pi_n \end{pmatrix},$$

$\pi_i \in \{1, 2, \ldots, n\}$ para $1 \leq i \leq n$. Utilizando a notação em uma coluna, ocultamos o mapeamento dos elementos negativos uma vez que $\pi(-i) = -\pi(i)$ para todo $i \in \{1, 2, \ldots, n\}$ e também escrevemos $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$. Por abuso de notação, também dizemos que a permutação π possui tamanho n. Desse modo, podemos tratar permutações com ou sem sinal de maneira unificada, fazendo diferenciações quando necessário.

A composição entre duas permutações π e σ, representada por $\pi \circ \sigma$, resulta na permutação γ tal que $\gamma(i) = \pi(\sigma(i))$, ou seja, $\gamma = (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \ldots\ \pi_{\sigma_n})$. Essa operação permite-nos modelar não somente genomas como permutações, mas também eventos de rearranjo, de tal forma que um evento de rearranjo ρ transforma um genoma π no genoma $\pi \circ \rho$. Na Seção 1.2, definiremos formalmente os eventos de rearranjo considerados nesta proposta.

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

Seja G um subconjunto de S_n ($S_n^{\pm}$) tal que qualquer permutação de S_n ($S_n^{\pm}$) pode ser obtida pela composição das permutações de G. As permutações pertencentes a G são chamadas de geradores de S_n ($S_n^{\pm}$). Ademais, se $\pi^{-1} \in G$ para toda permutação $\pi \in G$, então nós dizemos que G é simétrico. Um modelo de rearranjo M é um conjunto formado por geradores de S_n ($S_n^{\pm}$) tal que todos os geradores pertencentes a M modelam eventos de rearranjo e M é simétrico.

O Problema da Distância de Rearranjo pode ser definido formalmente da seguinte forma: dadas duas permutações π e σ, determine o número mínimo de geradores $\rho_1, \rho_2, \ldots, \rho_t$ pertencentes a um modelo de rearranjo M tal que $\pi \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_t = \sigma$. Este número corresponde à distância de rearranjo entre as permutações π e σ com respeito a M, denotada por $d_M(\pi, \sigma)$. Como M é simétrico, nós temos que $d_M(\pi, \sigma) = d_M(\sigma, \pi)$ pois

$$\pi \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_t = \sigma \iff \sigma \circ \rho_t^{-1} \circ \rho_{t-1}^{-1} \circ \cdots \circ \rho_1^{-1} = \pi.$$

A distância de rearranjo entre as permutações π e σ com respeito a M é igual à distância de rearranjo entre as permutações $\sigma^{-1} \circ \pi$ e ι com respeito a M uma vez que

$$\pi \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_t = \sigma \iff (\sigma^{-1} \circ \pi) \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_t = \iota.$$

Isso significa que podemos restringir a análise do Problema da Distância de Rearranjo ao caso em que uma das permutações é a permutação identidade. Sendo assim, definimos a distância de rearranjo de uma permutação π com respeito a M, denotada por $d_M(\pi)$, como sendo a distância de rearranjo entre π e ι com respeito a M. Ou seja, $d_M(\pi) = d_M(\pi, \iota)$.

Ordenar uma permutação significa transformá-la na permutação identidade por meio da composição de geradores. Desse modo, o Problema da Ordenação por Rearranjo é formalmente definido da seguinte maneira: dada uma permutação γ, determine a menor sequência de geradores $\rho_1, \rho_2, \ldots, \rho_k$ pertencentes a um modelo de rearranjo M tal que $\gamma \circ \rho_1 \circ \rho_2 \circ \cdots \circ \rho_k = \iota$.

Agora podemos visualizar porque o Problema da Distância de Rearranjo é fortemente vinculado ao Problema da Ordenação por Rearranjo: claramente, ao resolvermos este, teremos também resolvido aquele. Note que o inverso não é verdadeiro, pois a solução para o Problema da Distância de Rearranjo nos diz apenas qual é o tamanho da menor sequência de geradores. Apesar disso, a literatura costuma considerar esses problemas como equivalentes, dando maior ênfase ao Problema da Ordenação por Rearranjo. Nós iremos adotar essa abordagem por razões de simplicidade.

Ao definirmos o Problema da Ordenação por Rearranjo, fixamos um modelo de rearranjo. Isso significa que, dependendo do modelo de rearranjo considerado, temos uma variação do Problema da Ordenação por Rearranjo. Se, ao contrário, tivéssemos definido o modelo de rearranjo como um parâmetro de entrada, então teríamos formulado uma espécie de Problema da Ordenação por Rearranjo genérico, o qual aceitaria como caso específico qualquer variação do Problema da Ordenação por Rearranjo. Todavia, graças ao resultado de Even e Goldreich [20], sabemos que o problema genérico é NP-Difícil.

1.1.2 Algoritmos Aproximados e Algoritmos Heurísticos

Algoritmos aproximados ou aproximativos são algoritmos que não necessariamente produzem uma solução ótima para um determinado problema de otimização, mas produzem uma solução com uma certa garantia de aproximação em relação à solução ótima. Algoritmos heurísticos, por outro lado, produzem soluções sem nenhuma garantia formal de qualidade e, por essa razão, são tipicamente avaliados por meio de experimentos. A motivação para estudar-se algoritmos aproximados e algoritmos heurísticos advém principalmente da provável impossibilidade de encontrarmos algoritmos polinomiais ótimos para problemas de otimização NP-Difíceis.

De acordo com Cormen *et al.* [14, Capítulo 35], um algoritmo aproximado para um problema de otimização possui um fator de aproximação $\alpha(n)$ se, para qualquer instância de entrada de tamanho n, o custo C da solução produzida pelo algoritmo está dentro de um fator $\alpha(n)$ da solução ótima C^*. Isto é, se o problema é de maximização, então $\frac{C^*}{C} \leq \alpha(n)$. Ao contrário, se o problema é de minimização, então $\frac{C}{C^*} \leq \alpha(n)$.

Se um algoritmo aproximado possui fator de aproximação $\alpha(n)$, nós dizemos que ele é um algoritmo $\alpha(n)$-aproximado. No nosso caso, seja A um algoritmo aproximado para uma determinada variação do Problema da Ordenação por Rearranjo tal que, para cada permutação $\pi \in S_n$ ($S_n^{\pm}$), ele retorna uma sequência de geradores (pertencentes a um modelo de rearranjo M) de tamanho $A(\pi)$. Então, A é dito ser um algoritmo $\alpha(n)$-aproximado se $\frac{A(\pi)}{d_M(\pi)} \leq \alpha(n)$ para toda permutação $\pi \in S_n$ ($S_n^{\pm}$).

1.2 Variações do Problema da Ordenação por Rearranjo

Nesta seção, faremos uma breve revisão bibliográfica das principais variações do Problema da Ordenação por Rearranjo que consideram reversões ou transposições. Embora algumas dessas variações não tenham sido originalmente motivadas pela área de Rearranjo de Genomas, tendo inclusive pouca relevância do ponto de vista biológico, elas foram absorvidas pela área e atualmente são apresentadas como problemas de Rearranjo de Genomas [21, Prefácio].

1.2.1 Variações que Consideram Apenas Reversões

Uma reversão $r(i,j)$, $1 \leq i < j \leq n$, transforma a permutação sem sinal $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}\ \pi_j}\ \pi_{j+1}\ \ldots\ \pi_n)$ na permutação sem sinal $\pi \circ r(i,\ j) = (\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \ldots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$, ou seja, $r(i,j)$ é a permutação sem sinal $(1\ 2\ \ldots\ (i-1)\ j\ (j-1)\ \ldots\ (i+1)\ i\ (j+1)\ \ldots\ n)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões é chamada de Problema da Ordenação por Reversões.

Caprara [10] demonstrou que o Problema da Ordenação por Reversões é NP-Difícil. Watterson *et al.* [48] foram os primeiros a apresentar um algoritmo para este problema, sendo este um algoritmo $\frac{n-1}{2}$-aproximado. Kececioglu e Sankoff [37] foram os primeiros a apresentar um algoritmo de aproximação com fator constante. Eles apresentaram um algoritmo 2-aproximado. Posteriormente, Bafna e Pevzner [2] construíram um algoritmo de aproximação com fator 1.75. A próxima evolução foi dada por Christie [12] ao desenvolver um algoritmo com fator de aproximação 1.5. Por fim, o melhor resultado conhecido até o presente momento foi apresentado por

Berman, Hannenhalli e Karpinski [7]. Eles desenvolveram um algoritmo 1.375-aproximado.

Uma reversão com sinal $rs(i,j)$, $1 \leq i \leq j \leq n$, transforma a permutação com sinal $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}\ \pi_j}\ \pi_{j+1}\ \ldots\ \pi_n)$ na permutação com sinal $\pi \circ rs(i,\ j) = (\pi_1\ \pi_2\ \ldots\ \pi_{i-1}\ \underline{-\pi_j\ -\pi_{j-1}\ \ldots\ -\pi_{i+1}\ -\pi_i}\ \pi_{j+1}\ \ldots\ \pi_n)$, ou seja, $rs(i,j)$ é a permutação com sinal $(1\ 2\ \ldots\ (i-1)\ -j\ -(j-1)\ \ldots\ -(i+1)\ -i\ (j+1)\ \ldots\ n)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões com sinal é chamada de Problema da Ordenação por Reversões com Sinal.

Hannanhalli e Pevzner [30] foram os primeiros a resolver o Problema da Ordenação por Reversões com Sinal, apresentando um algoritmo ótimo que executa em tempo $O(n^4)$. Alguns refinamentos foram feitos neste algoritmo ao longo dos anos até que Tannier, Bergeron e Sagot [44] apresentaram um algoritmo $O(n^{\frac{3}{2}}\sqrt{\log n})$, sendo este o melhor resultado conhecido. Bader, Moret e Yan [1] mostraram como calcular a distância de reversão com sinal em tempo linear.

Uma reversão de prefixo $rp(i)$, $2 \leq i \leq n$, é equivalente à reversão $r(1,i)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões de prefixo é chamada de Problema da Ordenação por Reversões de Prefixo. Este problema também é conhecido como Problema da Ordenação de Panquecas e foi introduzido por Dweighter [18]. Bulteau, Fertin e Rusu [8] provaram que ele é NP-Difícil e o melhor algoritmo aproximado conhecido para resolvê-lo foi apresentado por Fischer e Ginzinger [22]. Tal algoritmo é uma 2-aproximação.

Uma reversão de prefixo com sinal $rps(i)$, $1 \leq i \leq n$, é equivalente à reversão com sinal $rs(1,i)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões de prefixo com sinal é chamada de Problema da Ordenação por Reversões de Prefixo com Sinal. Este problema também é conhecido como Problema da Ordenação de Panquecas Queimadas e foi introduzido por Cohen e Blum [13]. O melhor algoritmo aproximado conhecido para resolvê-lo é uma 2-aproximação apresentada por eles [13]. A complexidade deste problema não é conhecida.

Uma reversão $r(i,j)$ é dita uma k-reversão se $j - i + 1 \leq k$. Uma reversão curta é uma k-reversão tal que $k = 3$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões curtas é chamada de Problema da Ordenação por Reversões Curtas. A complexidade deste problema é desconhecida e o melhor algoritmo aproximado conhecido para resolvê-lo é uma 2-aproximação apresentada por Heath e Vergara [34, 45]. Cabe destacar que a variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por 2-reversões, isto é, apenas por reversões de elementos adjacentes, pode ser resolvida em tempo polinomial utilizando o algoritmo *Bubble*

Sort [35].

1.2.2 Variações que Consideram Apenas Transposições

Uma transposição $t(i,\ j,\ k)$, $1 \leq i < j < k \leq n+1$, transforma a permutação $\pi = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_i\ \ldots\ \pi_{j-1}}\ \underline{\pi_j\ \ldots\ \pi_{k-1}}\ \pi_k\ \ldots\ \pi_n)$ na permutação $\pi \circ t(i,\ j,\ k) = (\pi_1\ \ldots\ \pi_{i-1}\ \underline{\pi_j\ \ldots\ \pi_{k-1}}\ \underline{\pi_i\ \ldots\ \pi_{j-1}}\ \pi_k\ \ldots\ \pi_n)$, ou seja, $t(i,\ j,\ k)$ é a permutação $(1\ 2\ \ldots\ (i-1)\ j\ (j+1)\ \ldots\ (k-1)\ i\ \ldots\ (j-1)\ k\ \ldots\ n)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas transposições é chamada de Problema da Ordenação por Transposições.

O primeiro resultado relevante obtido para o Problema da Ordenação por Transposições é devido a Bafna e Pevzner [3]. Eles apresentaram um algoritmo aproximativo com fator de aproximação 1.5. Elias e Hartman [19] produziram um algoritmo de aproximação com fator de aproximação 1.375. Recentemente, Bulteau, Fertin e Rusu [9] demonstraram que este problema é NP-Difícil.

Uma permutação pode ser dividida em blocos de elementos contíguos tais que cada bloco é formado por uma sequência crescente maximal de inteiros consecutivos. Em outras palavras, a sequência de elementos contíguos π_i, π_{i+1}, ..., π_j, $i \leq j$, de uma permutação π é um bloco caso $\pi_{k+1} = \pi_k + 1$ para $k = i, i+1, \ldots, j-1$ e caso não existam π_{i-1} e π_{j+1} tais que $\pi_{i-1} = \pi_i - 1$ e $\pi_{j+1} = \pi_j + 1$. Por exemplo, a permutação (6 3 4 5 2 1) é formada por 4 blocos: 6, 3 4 5, 2 e 1. Uma transposição $t(i,\ j,\ k)$ aplicada em uma permutação π é dita ser uma transposição de bloco caso $\pi_i\ \pi_{i+1}\ \ldots\ \pi_{j-1}$ ou $\pi_j\ \pi_{j+1}\ \ldots\ \pi_{k-1}$ seja um bloco. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por transposições de blocos é chamada de Problema da Ordenação por Transposições de Blocos ou simplesmente Problema da Ordenação por Blocos. Este problema foi introduzido por Latifi [38] e demonstrado ser NP-Difícil por Bein *et al.* [4]. Os melhores algoritmos conhecidos para resolvê-lo são 2-aproximados, um deles proposto por Mahajan *et al.* [40] e o outro proposto por Bein *et al.* [5].

Uma transposição de prefixo $tp(i,\ j)$, $2 \leq i < j \leq n+1$, é equivalente à transposição $t(1, i, j)$. A variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por transposições de prefixo é chamada de Problema da Ordenação por Transposições de Prefixo. Este problema foi introduzido por Dias e Meidanis [17] e o melhor algoritmo aproximado conhecido para resolvê-lo é uma 2-aproximação apresentada por eles [17]. A complexidade deste problema é desconhecida.

Uma transposição $t(i,\ j,\ k)$ é dita uma m-transposição se $k - i \leq m$. Uma transposição curta é uma m-transposição tal que $m = 3$. A variação do Problema da Ordenação por Rearranjo

que considera um modelo de rearranjo composto apenas por transposições curtas é chamada de Problema da Ordenação por Transposições Curtas. Heath e Vergara foram os primeiros a estudarem essa variação [32] e apresentaram um algoritmo $\frac{4}{3}$-aproximado para resolvê-la [33]. Recentemente, Jiang, Zhu e Zhu [36] apresentaram uma $(1+\epsilon)$-aproximação para este problema, sendo ϵ o número de elementos dividido pelo número de inversões da permutação (dois elementos π_i e π_j de uma permutação π são ditos ser uma inversão caso $i < j$ e $\pi_i > \pi_j$). A complexidade desta variação é desconhecida.

1.2.3 Variações que Consideram Reversões e Transposições

Walter, Dias e Meidanis [46] foram os primeiros a considerar tanto a variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões com sinal e transposições quanto a variação que considera um modelo de rearranjo composto apenas por reversões e transposições. A primeira é chamada de Problema da Ordenação por Reversões com Sinal e Transposições e a segunda é chamada de Problema da Ordenação por Reversões e Transposições. Para este problema, eles apresentaram um algoritmo 3-aproximado, enquanto que, para aquele, um algoritmo 2-aproximado. Rahman, Shatabda e Hasan [41] apresentaram um algoritmo $2k$-aproximado para o Problema da Ordenação por Reversões e Transposições, sendo k o fator de aproximação do algoritmo de decomposição de ciclos. Para o melhor valor de k conhecido [39], que é igual a $1.4193 + \epsilon$, $\epsilon > 0$, nós temos que o fator de aproximação desse algoritmo é igual a $2.8386 + \gamma$, para qualquer $\gamma > 0$. A complexidade de ambos problemas é desconhecida.

Sharmina *et al.* [43] estudaram a variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto apenas por reversões de prefixo e transposições de prefixo, chamada de Problema da Ordenação por Reversões de Prefixo e Transposições de Prefixo ou de Problema da Ordenação de Panquecas com Duas Espátulas. Eles [43] desenvolveram um algoritmo 3-aproximado para resolvê-la. Recentemente, Dias e Dias [15] desenvolveram um algoritmo 2-aproximado, sendo este o melhor algoritmo aproximado conhecido. A complexidade do problema não é conhecida.

Alguns pesquisadores também consideraram um outro tipo de operação, chamada transreversão. Ela é similar a uma transposição, exceto que um dos segmentos transpostos da permutação é invertido. Um transreversão com sinal do tipo A $trs_a(i,\ j,\ k)$, $1 \leq i < j < k \leq n+1$, transforma a permutação com sinal π na permutação com sinal $\pi \circ trs_a(i,\ j,\ k) = \pi \circ rs(i,\ j-1) \circ t(i,\ j,\ k)$. Uma transreversão com sinal do tipo B $trs_b(i,\ j,\ k)$, $1 \leq i < j < k \leq n+1$, transforma a permutação com sinal π na permutação com sinal $\pi \circ trs_b(i,\ j,\ k) = \pi \circ$

$rs(j,\, k-1) \circ t(i,\, j,\, k)$.

Gu *et al.* [28] desenvolveram um algoritmo 2-aproximado para a variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto por reversões com sinal, transposições e transreversões com sinal do tipo A, a qual chamamos de Problema da Ordenação por Reversões com Sinal, Transposições e Transreversões com Sinal do Tipo A. Hartman e Sharan [31] desenvolveram um algoritmo 1.5-aproximado para a variação do Problema da Ordenação por Rearranjo que considera um modelo de rearranjo composto por transposições e transreversões com sinal dos tipos A e B, a qual chamamos de Problema da Ordenação por Transposições e Transreversões com Sinal do Tipo A e do Tipo B. A complexidade de ambos os problemas é desconhecida.

1.2.4 Consolidação do Estado da Arte

A Tabela 1 resume o estado da arte das variações do Problema da Ordenação por Rearranjo revisadas nesta seção.

Modelo de Rearranjo	**Complexidade**	**Melhor Solução Teórica**
Reversões com Sinal	Polinomal	Algoritmo exato $O(n^{\frac{3}{2}}\sqrt{\log n})$
Reversões	NP-Difícil	Algoritmo 1.375-aproximado
Transposições	NP-Difícil	Algoritmo 1.375-aproximado
Transposições Curtas	?	Algoritmo $(1+\epsilon)$-aproximado
Transposições e Transreversões do Tipo A e do Tipo B	?	Algoritmo 1.5-aproximado
Reversões de Prefixo	NP-Difícil	Algoritmo 2-aproximado
Reversões de Prefixo com Sinal	?	Algoritmo 2-aproximado
Reversões Curtas	?	Algoritmo 2-aproximado
Transposições de Blocos	NP-Difícil	Algoritmo 2-aproximado
Transposições de Prefixo	?	Algoritmo 2-aproximado
Reversões com Sinal e Transposições	?	Algoritmo 2-aproximado
Reversões com Sinal, Transposições e Transreversões do Tipo A	?	Algoritmo 2-aproximado
Reversões de Prefixo e Transposições de Prefixo	?	Algoritmo 2-aproximado
Reversões e Transposições	?	Algoritmo $2k$-aproximado

Tabela 1: Estado da arte das variações do Problema da Ordenação por Rearranjo revisadas na Seção 1.2.

2 Objetivos

Esta proposta de doutorado corresponde a uma continuação dos trabalhos desenvolvidos durante o mestrado do aluno [23]. Tal continuação pode ser dividida em três frentes distintas, tal como descrito nas seções seguintes.

2.1 Algoritmos Alternativos para o Problema da Ordenação por Transposições

Os melhores algoritmos aproximados conhecidos para o Problema da Ordenação por Transposições (alguns deles citados na Seção 1.2.2) são baseados em uma estrutura muito utilizada para resolver variações do Problema da Ordenação por Rearranjo, chamada grafo de ciclos. Alguns pesquisadores alegam que tal estrutura é muito complexa, por isso desenvolveram algoritmos aproximados e algoritmos heurísticos baseados em estruturas alternativas, supostamente mais simples. Mais especificamente, os algoritmos aos quais estamos nos referindo são: o algoritmo 2.25-aproximado proposto por Walter, Dias e Meidanis [47]; o algoritmo 3-aproximado proposto por Benoît-Gagné e Hamel [6]; e a heurística proposta por Guyer, Heath e Vergara [29].

No mestrado, nós apresentamos [26] alguns resultados teóricos a respeito do fator de aproximação desses algoritmos e alguns resultados experimentais obtidos a partir da comparação das respostas produzidas por eles para permutações pequenas. Um de nossos objetivos foi continuarmos a investigar esses algoritmos e tentar obter resultados tanto no sentido de melhorar a teoria subjacente a eles quanto no sentido de mostrar experimentalmente como esses algoritmos se comparam aos melhores algoritmos conhecidos.

Nós dedicamos o primeiro e parte do segundo semestre do Doutorado nesta frente. Como resultado, nós conseguimos melhorar a complexidade do algoritmo proposto por Benoît-Gagné e Hamel [6] de $O(n^2)$ para $O(n \log n)$. Além disso, nós realizamos a comparação experimental entre os três algoritmos citados anteriormente com os melhores algoritmos conhecidos: os algoritmos aproximados propostos Bafna e Pevzner [3] e Elias e Hartman [19] e as heurísticas propostas por Dias e Dias [16]. Os resultados mostraram que o algoritmo proposto por Walter, Dias e Meidanis [47] é o único que se compara, em termos de qualidade das respostas, aos melhores algoritmos conhecidos. Todos esses resultados deram origem a um artigo entitulado "On Alternative Approaches for Approximating the Transposition Distance". Este artigo foi submetido ao periódico *Journal of Universal Computer Science*.

2.2 Desenvolvimento de Heurísticas

A principal contribuição do mestrado concluído pelo aluno foi uma ferramenta [25] para avaliar algoritmos aproximados e heurísticos para o Problema da Ordenação por Rearranjo. Para construir essa ferramenta, foi necessário calcular a distância de rearranjo de todas as permutações de S_n, $1 \leq n \leq 13$, e de $S_n^{\pm}$, $1 \leq n \leq 10$, com respeito a diversos modelos de rearranjo abordados na literatura, tais como os revisados na Seção 1.2. As distâncias de rearranjo dessas permutações, juntamente com as sequências mínimas de eventos de rearranjo que as ordenam, estão armazenadas em uma base de dados [24] que pode ser acessada pela internet. Um de nossos objetivos foi utilizar as informações contidas nessa base de dados para desenvolver heurísticas para variações do Problema da Ordenação por Rearranjo.

Nós dedicamos parte do segundo e o terceiro semestre do Doutorado nesta frente. Como resultado, nós desenvolvemos uma heurística de melhoria, isto é, uma heurística que atua de modo a melhorar uma dada solução não ótima para uma variação do Problema da Ordenação de Rearranjo, que pode ser obtida a partir da execução de um algoritmo aproximado ou heurístico. Para testar a qualidade dessa heurística, nós a aplicamos nas respostas produzidas por 23 algoritmos aproximados ou heurísticos relativos a 9 variações distintas do Problema da Ordenação por Rearranjo. Os resultados obtidos, assim como uma descrição mais detalhada da heurística, deram origem a um artigo entitulado "A General Heuristic for Genome Rearrangement Problems". Este artigo foi aceito para publicação no periódico *Journal of Bioinformatics and Computational Biology*.

2.3 Desenvolvimento de Algoritmos Aproximados

Grande parte dos estudos realizados no mestrado diz respeito às variações do Problema da Ordenação por Rearranjo revisadas na Seção 1.2. Esses estudos levaram à percepção de que algumas dessas variações possuem uma lacuna a ser explorada, que se refere ao fato dos melhores algoritmos aproximados conhecidos para resolvê-las possuírem fatores de aproximação relativamente altos se comparados aos algoritmos aproximados conhecidos para o Problema da Ordenação por Reversões e Problema da Ordenação por Transposições (ver Tabela 1). Desse modo, o nosso objetivo é desenvolver algoritmos aproximados cujos fatores de aproximação superem os dos algoritmos aproximados existentes. As variações para as quais pretendemos alcançar tal objetivo são o Problema da Ordenação por Reversões Curtas, o Problema da Ordenação por Blocos e o Problema da Ordenação por Reversões e Transposições. Os melhores algoritmos aproximados para essas variações serão revisados na próxima seção.

3 Revisão Bibliográfica Aprofundada

Esta seção apresenta uma revisão mais detalhada das variações do Problema da Ordenação por Rearranjo que constituirão o foco de nossa pesquisa.

3.1 Problema da Ordenação por Reversões Curtas

O Problema da Ordenação por Reversões Curtas consiste em determinar a menor sequência de reversões curtas que ordena uma determinada permutação sem sinal π. O tamanho dessa sequência é denominado como a distância de reversão curta de π, denotada por $d_{rc}(\pi)$. Nesta seção, iremos apresentar três algoritmos aproximados para o Problema da Ordenação por Reversões Curtas: um algoritmo 3-aproximado e um algoritmo 2-aproximado propostos por Vergara [45] e um algoritmo 2-aproximado proposto por Heath e Vergara [34, 45].

Uma inversão em uma permutação sem sinal π é um par de elementos (π_i, π_j) que não estão na ordem relativa correta, isto é, $i < j$ e $\pi_i > \pi_j$. O número de inversões de π é denotado por $inv(\pi)$. Note que a permutação identidade é a única permutação que não possui inversões.

Exemplo 1. *Seja* $\pi = (3\ 2\ 4\ 1)$. *Os pares de elementos* $(3, 2)$, $(3, 1)$, $(2, 1)$ *e* $(4, 1)$ *são inversões em* π.

Uma reversão curta $rc(i, j)$ é dita ser corretiva se ela inverte a ordem de dois elementos π_i e π_j que são uma inversão em π. Heath e Vergara [34, 45] demonstraram que, para qualquer permutação sem sinal π, existe uma sequência de reversões curtas que ordena π otimamente tal que todas as reversões curtas são corretivas. Isso garante que podemos restringir nossa análise apenas às reversões curtas corretivas. Como uma reversão curta corretiva pode remover 1 ou 3 inversões de uma permutação sem sinal (Lema 1), o Lema 1 segue trivialmente.

Lema 1. *Uma reversão curta corretiva pode remover 1 ou 3 inversões de uma permutação sem sinal.*

Demonstração. Uma reversão curta corretiva $rc(i, i+1)$ remove somente 1 inversão de uma permutação π, aquela caracterizada pelos par de elementos π_i e π_{i+1}. Já uma reversão curta corretiva $rc(i, i+2)$ remove 1 inversão caso $\pi_{i+1} > \pi_i > \pi_{i+2}$ ou $\pi_i > \pi_{i+2} > \pi_{i+1}$ e remove 3 inversões caso $\pi_i > \pi_{i+1} > \pi_{i+2}$. □

Corolário 1. *Para toda permutação sem sinal* π, $d_{rc}(\pi) \geq \frac{inv(\pi)}{3}$.

Três elementos contíguos $(\pi_i, \pi_{i+1}, \pi_{i+2})$, $0 \leq i \leq n-2$, de uma permutação $\pi \in S_n$ formam uma tripla se $\pi_i > \pi_{i+1} > \pi_{i+2}$. O conjunto formado por todas as triplas de π é denotado por

$triplas(\pi)$. Não é difícil notar que a reversão curta corretiva $rc(i,\, i+2)$ remove 3 inversões de π.

Exemplo 2. *Seja* $\pi = (4\ 3\ 2\ 1\ 5)$. *Nós temos que* $triplas(\pi) = \{(4,\, 3,\, 2),\, (3,\, 2,\, 1)\}$.

Vergara [45] desenvolveu um algoritmo guloso (Algoritmo 1) baseado em inversões. A cada iteração, esse algoritmo seleciona a reversão curta corretiva que remove a maior quantidade de inversões possível. Caso exista mais do que uma reversão curta corretiva que remove a maior quantidade de inversões, o algoritmo seleciona aquela que produz uma permutação com o maior número de triplas.

Algorithm 1: Algoritmo 3-aproximado proposto por Vergara [45]

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de reversões curtas aplicadas para ordenar π.

```
1  d ← 0;
2  enquanto π ≠ ι faça
3  |  se |triplas(π)| > 0 então
4  |  |  Seja (π_i, π_{i+1}, π_{i+2}) uma tripla tal que |triplas(π ∘ rc(i, i+2))| possui valor
   |  |  máximo;
5  |  |  π ← π ∘ rc(i, i+2);
6  |  |  d ← d + 1;
7  |  senão
8  |  |  Seja rc(i, j) uma reversão curta corretiva tal que |triplas(π ∘ rc(i, j))| possui
   |  |  valor máximo;
9  |  |  π ← π ∘ rc(i, j);
10 |  |  d ← d + 1;
11 |  fim
12 fim
13 retorna d;
```

Como o Algoritmo 1 remove pelo menos uma inversão a cada reversão curta aplicada e $d_{rc}(\pi) \geq \lceil \frac{inv(\pi)}{3} \rceil$ (Lema 1), nós concluímos que ele é uma 3-aproximação. Quanto à complexidade de tempo, $triplas(\pi)$ pode ser computada em tempo $O(n)$. Como existem no máximo $n-2$ triplas e aplicar uma reversão curta leva tempo constante, podemos computar tanto a linha 4 quanto a linha 8 também em tempo $O(n)$. Ou seja, as linhas 3-11 são executadas em tempo $O(n)$. Dado que podem existir no máximo $\binom{n}{2}$ inversões, o laço **enquanto** executa $O(n^2)$ vezes, portanto o Algoritmo 1 roda em tempo $O(n^3)$.

Para desenvolver um algoritmo com um fator de aproximação melhor, Heath e Vergara [34,45] utilizaram um estrutura chamada diagrama vetorial. Para cada elemento π_i de uma permutação

sem sinal π, definimos um vetor $v(\pi_i)$ cujo tamanho é dado por $|v(\pi_i)| = |\pi_i - i|$. Se $|v(\pi_i)| > 0$, a direção de $v(\pi_i)$ é dada pelo sinal de $\pi_i - i$. O diagrama vetorial $V_{rc}(\pi)$ de π é o conjunto formado pelos vetores dos elementos de π. A soma dos tamanhos dos vetores pertencentes a $V_{rc}(\pi)$ é denotada por $|V_{rc}(\pi)|$. Note que $|V_{rc}(\pi)| = 0$ se, e somente se, $\pi = \iota$. Como uma reversão curta corretiva pode diminuir $|V_{rc}(\pi)|$ de 4 unidades no máximo, o Lema 2 segue trivialmente.

Lema 2. *Para toda permutação sem sinal π, $d_{rc}(\pi) \geq \frac{|V_{rc}(\pi)|}{4}$.*

Dois elementos π_i e π_j, $i < j$, de uma permutação sem sinal π são ditos serem vetorialmente opostos se os vetores $v(\pi_i)$ e $v(\pi_j)$ possuem direções opostas e tanto $|v(\pi_i)| \geq j - i$ quanto $|v(\pi_j)| \geq j - i$. Ademais, eles são dito serem m-vetorialmente opostos se $j - i = m$. Note que m especifica a distância entre elementos vetorialmente opostos.

Exemplo 3. *Seja π a permutação do Exemplo 2. Nós temos que os elementos π_2 e π_3 são 1-vetorialmente opostos e os elementos π_1 e π_4 são 3-vetorialmente opostos.*

Heath e Vergara [34,45] provaram que toda permutação sem sinal π, $\pi \neq \iota$, possui pelo menos dois elementos vetorialmente opostos. Sejam π_i e π_j elementos m-vetorialmente opostos de $\pi \in S_n$ e seja $\pi' \in S_n$ a permutação que possui tais elementos em posições invertidas, isto é, $\pi'_k = \pi_k$ para $k \in \{1, 2, \ldots, n\} \setminus \{i, j\}$, $\pi'_i = \pi_j$ e $\pi'_j = \pi_i$. Eles também provaram que $|V_{rc}(\pi)| - |V_{rc}(\pi')| = 2m$, ou seja, ao trocarmos dois elementos m-vetorialmente opostos de uma permutação sem sinal π, diminuímos $|V_{rc}(\pi)|$ de $2m$ unidades.

Sejam π_i e π_j elementos m-vetorialmente opostos em $\pi \in S_n$. Vergara [45] mostrou que é possível trocar π_i e π_j de posição por meio da aplicação de reversões curtas, tal como descrito pelo Algoritmo *TrocaComReversoesCurtas*. Além disso, ele também mostrou que o Algoritmo *TrocaComReversoesCurtas* aplica $m - 1$ reversões curtas se m é par e aplica m reversões curtas se m é ímpar. Isso implica que cada reversão curta aplicada pelo Algoritmo *TrocaComReversoesCurtas* diminui $|V_{rc}(\pi)|$ de $\frac{2m}{m-1}$ unidades em média se m é par. Por outro lado, se m é ímpar, então cada reversão curta aplicada diminui $|V_{rc}(\pi)|$ de 2 unidades em média.

Heath e Vergara [34, 45] apresentam um algoritmo para uma ordenar uma permutação sem sinal por reversões curtas baseado em elementos vetorialmente opostos (Algoritmo 4). A cada iteração, o algoritmo seleciona dois elementos vetorialmente opostos e chama o Algoritmo *TrocaComReversoesCurtas* para realizar a troca desses elementos. Como as reversões curtas aplicadas pelo Algoritmo *TrocaComReversoesCurtas* diminui a soma dos tamanhos dos vetores, em algum momento esta soma irá ser nula, precisamente quando a permutação estiver ordenada. No pior caso, cada reversão curta aplicada pelo algoritmo diminui a soma dos tamanhos dos vetores de 2

unidades em média. Dado que $d_{rc}(\pi) \geq \frac{|V_{rc}(\pi)|}{4}$ (Lema 2), concluímos que esse algoritmo é uma 2-aproximação.

Algorithm 2: TrocaComReversoesCurtas

Entrada: Uma permutação $\pi \in S_n$ e par de posições (i, j) tal que $i < j$.
Saída: O número de reversões curtas aplicadas para trocar os elementos π_i e π_j de posição.

```
1  k ← i;
2  d ← 0;
3  enquanto k < j − 2 faça
4  |  π ← π ∘ rc(k, k + 2);
5  |  k ← k + 2;
6  |  d ← d + 1;
7  fim
8  se j − i é ímpar então
9  |  π ← π ∘ rc(k, k + 1);
10 |  k ← k − 2;
11 |  d ← d + 1;
12 fim
13 enquanto k ≥ i faça
14 |  π ← π ∘ rc(k, k + 2);
15 |  k ← k − 2;
16 |  d ← d + 1;
17 fim
18 retorna d;
```

Algorithm 3: EncontraOpostos

Entrada: Uma permutação $\pi \in S_n$.
Saída: Par de posições (i, j) tal que π_i e π_j são vetorialmente opostos.

```
1  i ← n;
2  enquanto π_i ≤ i faça
3  |  i ← i − 1;
4  fim
5  j ← i + 1;
6  enquanto π_j = j faça
7  |  j ← j + 1;
8  fim
9  retorna (i, j);
```

Algorithm 4: Algoritmo 2-aproximado proposto por Heath e Vergara [34, 45]

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de reversões curtas aplicadas para ordenar π.

```
1 d ← 0;
2 enquanto π ≠ ι faça
3 |   (i, j) ← EncontraOpostos(π);
4 |   d ← d + TrocaComReversoesCurtas(π, i, j);
5 fim
6 retorna d;
```

Quanto à complexidade de tempo, nós temos que podem existir no máximo $\binom{n}{2}$ elementos vetorialmente opostos, portanto o laço **enquanto** executa $O(n^2)$ vezes. Cada chamada ao Algoritmo *EncontraOpostos* toma tempo $O(n)$, então as chamadas a esse algoritmo tomam tempo $O(n^3)$ no total. O tempo total gasto pelas chamadas ao Algoritmo *TrocaComReversoesCurtas* não depende do número de vezes que o laço *enquanto* itera, mas sim da soma dos tamanhos dos vetores. Cada chamada do Algoritmo *TrocaComReversoesCurtas* toma tempo $O(m)$ para diminuir a soma dos tamanhos dos vetores de $2m$. Como cada vetor pode ter tamanho igual a n no máximo, a soma dos tamanhos dos vetores é igual a n^2 no máximo. Logo, no total, as chamadas ao Algoritmo *TrocaComReversoesCurtas* tomam tempo $O(n^2)$. Sendo assim, podemos concluir que o Algoritmo 4 executa em tempo $O(n^3)$.

Vergara [45] desenvolveu uma variante gulosa do Algoritmo 4 (Algoritmo 6) baseada na distância entre elementos vetorialmente opostos. A cada iteração, o algoritmo tenta encontrar dois elementos m-vetorialmente opostos tal que m seja par e possua o menor valor possível. Caso ele encontre, o algoritmo seleciona tais elementos. Caso ele não encontre, o algoritmo seleciona dois elementos m-vetorialmente opostos tal que m possua o menor valor possível. Após ter selecionado dois elementos m-vetorialmente opostos, o algoritmo chama o Algoritmo *TrocaComReversoesCurtas* para realizar a troca desses elementos. No pior caso, o algoritmo seleciona apenas elementos m-vetorialmente opostos tal que m é ímpar e aplica, portanto, $\frac{|V_{rc}(\pi)|}{2}$ reversões curtas. Desse modo, esse algoritmo também é uma 2-aproximação.

Quanto à complexidade de tempo, a única diferença em relação ao Algoritmo 4 é o tempo gasto pelas chamadas ao Algoritmo *EncontraOpostosGuloso*. A cada iteração, ele tem que verificar todos os elementos vetorialmente oposto, o que toma tempo $O(n^2)$. Sendo assim, podemos concluir que o Algoritmo 6 executa em tempo $O(n^4)$.

Algorithm 5: EncontraOpostosGuloso

Entrada: Uma permutação $\pi \in S_n$.
Saída: Par de posições (i, j) tal que π_i e π_j são vetorialmente opostos.

se *existem dois elementos m-vetorialmente opostos tal que m é par* **então**
  $(i, j) \leftarrow$ posições de dois elementos m-vetorialmente opostos tal que m é par e m possui o menor valor possível;
  retorna (i, j);
senão
  $(i, j) \leftarrow$ posições de dois elementos m-vetorialmente opostos tal que m possui o menor valor possível;
  retorna (i, j);
fim

Algorithm 6: Variante gulosa do Algoritmo 4 proposta por Vergara [45]

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de reversões curtas aplicadas para ordenar π.

$d \leftarrow 0$;
enquanto $\pi \neq \iota$ **faça**
  $(i, j) \leftarrow$ `EncontraOpostosGuloso(`π`)`;
  $d \leftarrow d +$ `TrocaComReversoesCurtas(`π, i, j`)`;
fim
retorna d;

3.2 Problema da Ordenação por Blocos

O Problema da Ordenação por Blocos consiste em determinar a menor a sequência de transposições de blocos que ordena uma determinada permutação sem sinal π. O tamanho dessa sequência é denominado como a distância de transposição de bloco de π, denotada por $d_{tb}(\pi)$. Nesta seção, iremos apresentar dois algoritmos aproximados para o Problema da Ordenação por Blocos: um algoritmo 2-aproximado proposto por Mahajan *et al.* [40] e o outro algoritmo 2-aproximado proposto por Bein *et al.* [5]. Ambos os algoritmos baseiam-se na mesma estratégia de reduzir o Problema da Ordenação por Blocos a um outro problema que pode ser resolvido em tempo polinomial. Um desses problemas é chamado de Problema da União de Blocos e o outro, Problema da Deleção de Blocos. Por razões de simplicidade, apresentaremos apenas de que forma é possível utilizar a solução desses problemas para aproximar a distância de transposição de bloco de uma permutação sem sinal π. Ressaltamos, contudo, que os artigos originais [5, 40] também mostram como derivar a sequência de transposições de bloco que ordena π a partir da

solução desses problemas.

Dizemos que uma sequência de números inteiros é uma subsequência de outra sequência se ela é obtida após (possivelmente) removermos alguns elementos desta. Uma *substring* de uma sequência de números inteiros é uma subsequência formada por elementos consecutivos da sequência original. Para qualquer $1 \leq i \leq j \leq n$, denotamos por $\pi_{i \ldots j}$ a *substring* de uma permutação $\pi \in S_n$ que começa em π_i e termina em π_j. Dizemos que uma sequência de números inteiros é uma subsequência de $\pi \in S_n$ se ela é uma subsequência de $\pi_1\ \pi_2\ \ldots\ \pi_n$.

Exemplo 4. *A sequência de números inteiros* 2 3 *é uma subsequência da permutação* (2 4 3 1)*, mas não é uma substring. Por outro lado,* 4 3 1 *é tanto uma subsequência quanto uma substring de* π*.*

Uma sequência de números inteiros é dita ser crescente caso seus elementos estejam em ordem crescente (da esquerda para direita). Uma *substring* crescente de uma sequência de números inteiros é dita ser uma *substring* crescente maximal caso ela não seja *substring* de nenhuma outra *substring* crescente da sequência original. Um bloco de uma sequência de números inteiros é definido como uma *substring* maximal de inteiros consecutivos.

Exemplo 5. *Dada a sequência de números inteiros* 3 5 4 1 2*, nós temos que* 3 5*,* 4 *e* 1 2 *são substrings crescentes maximais. Além disso, esta sequência possui 4 blocos:* 3*,* 5*,* 4 *e* 1 2*.*

Qualquer permutação $\pi \in S_n$ pode ser unicamente decomposta em *substrings* crescentes maximais. Considerando cada uma dessas *substrings* como uma sequência crescente, nós podemos construir um multiconjunto $\mathbb{S}_\pi = \{S_1, S_2, \ldots, S_k\}$ de sequências crescentes tal que cada S_i, $1 \leq i \leq k$, é uma *substring* crescente maximal de π. O Problema da União de Blocos consiste em transformar $\mathbb{S}_\pi$ em $\mathbb{M} = \{\zeta, \epsilon, \ldots, \epsilon\}$, sendo ζ a sequência $1\ 2\ \ldots\ n$ e ϵ a sequência vazia, utilizando o menor número de movimentos possível. Os movimentos permitidos são: pegue um bloco de qualquer sequência S_i e insira-o em alguma outra sequência S_j de tal forma que o bloco movido se junte a outro bloco formando um novo bloco (note que a nova sequência S'_j continua sendo uma sequência crescente). O menor número de movimentos possível é denotado por $ub(\mathbb{S}_\pi)$.

Exemplo 6. *Seja* $\pi = (1\ 6\ 2\ 5\ 3\ 4)$*. A instância do Problema da União dos Blocos é dada por* $\mathbb{S}_\pi = \{1\ 6,\ 2\ 5,\ 3\ 4\}$*. Nós temos que os movimentos mínimos necessários para transformar* $\mathbb{S}_\pi$ *em* $\mathbb{M} = \{1\ 2\ 3\ 4\ 5\ 6,\ \epsilon,\ \epsilon\}$ *são: pegar o bloco* 3 4 *da última sequência de* $\mathbb{S}_\pi$ *e inserí-lo na segunda sequência de* $\mathbb{S}_\pi$*, obtendo a instância* $\mathbb{S}'_\pi = \{1\ 6,\ 2\ 3\ 4\ 5,\ \epsilon\}$*; e depois, pegar o bloco* 2 3 4 5 *da segunda sequência de* $\mathbb{S}'_\pi$ *e inserí-lo na primeira sequência de* $\mathbb{S}'_\pi$*, obtendo* $\mathbb{M}$*. Logo, nós temos que* $ub(\mathbb{S}_\pi) = 2$*.*

Dada uma instância de entrada $\mathbb{S}_\pi = \{S_1, S_2, \ldots, S_k\}$ do Problema da União de Blocos, construímos um grafo direcionado $G_{\mathbb{S}_\pi} = (V, E)$ tal que:

1. $V = \{1, 2, \ldots, n\}$ e $E = \{(u, v) : u < v$ e $u, v \in S_p$ para algum $p = 1, 2, \ldots, k\}$;
2. Para $1 \leq i \leq j \leq n$, $G_{\mathbb{S}_\pi}([i, j])$ denota o subgrafo induzido pelo conjunto de vértices $\{i, i+1, \ldots, j\}$.
3. Um par de arestas (i, j) e (k, l) é dito se cruzar caso $i \leq k < j \leq l$ ou $k \leq i < l \leq j$. Um conjunto $E' \subseteq E$ é dito ser um conjunto livre de cruzamentos caso nenhum par de arestas de E' se cruze.
4. O tamanho do maior conjunto livre de cruzamentos de $G_{\mathbb{S}_\pi}([i, j])$ é denotado por $c(i, j)$. Além disso, denotamos $c(1, n)$ por $c(\mathbb{S}_\pi)$.

Exemplo 7. *A Figura 1 ilustra o grafo direcionado associado à instância* $\mathbb{S}_\pi = \{8, 2\ 5\ 6, 3\ 9, 1\ 4\ 7\}$. *Note que o par de arestas* $(2, 5)$ *e* $(1, 4)$ *se cruza, assim como o par de arestas* $(2, 5)$ *e* $(2, 6)$. *Por outro lado, o par de arestas* $(5, 6)$ *e* $(3, 9)$ *não se cruza, nem o par* $(1, 4)$ *e* $(4, 7)$. *O conjunto* $\{(3, 9), (4, 7), (5, 6)\}$ *é um dos maiores conjuntos livres de cruzamentos de* $G_{\mathbb{S}_\pi}([1, 9])$ (*o conjunto* $\{(2, 5), (5, 6), (1, 7)\}$ *e o conjunto* $\{(1, 4), (4, 7), (5, 6)\}$ *também são*), *portanto* $c(\mathbb{S}_\pi) = 3$.

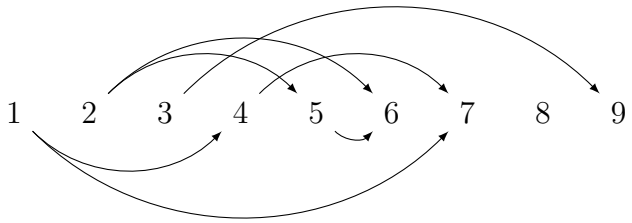


Figura 1: Grafo direcionado associado à instância $\mathbb{S}_\pi = \{8, 2\ 5\ 6, 3\ 9, 1\ 4\ 7\}$.

Seja $t(i, j)$ o tamanho do maior conjunto livre de cruzamentos de $G_{\mathbb{S}_\pi}([i, j])$ que contém a aresta (i, j) e seja $q(i, j)$ o tamanho do maior conjunto livre de cruzamentos de $G_{\mathbb{S}_\pi}([i, j])$ que não contém a aresta (i, j). Mahajan *et al.* [40] demonstraram que as seguintes relações de recorrência valem para $c(i, j)$, $t(i, j)$ e $q(i, j)$:

$$\text{Para } 1 \leq i \leq n, \quad c(i,\, i) = 0$$

$$\text{Para } 1 \leq i \leq n-1, \quad c(i,\, i+1) = \begin{cases} 1 & \text{se } (i,\, i+1) \in E \\ 0 & \text{caso contrário} \end{cases}$$

$$\text{Para } 1 \leq i \leq j-2 \leq n-2, \quad c(i,\, j) = \max\{t(i,\, j),\, q(i,\, j)\}$$

$$t(i,\, j) = \begin{cases} 1 + c(i+1, j-1) & \text{se } (i,\, j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

$$q(i,\, j) = \max_{i<k<j}\{c(i,\, k),\, c(k,\, j)\}$$

Essas relações de recorrência dão origem a um algoritmo de programação dinâmica $O(n^3)$ para calcular $c(1,\, n)$. Como Mahajan *et al.* [40] provaram que $ub(\mathbb{S}) = n - 1 - c(\mathbb{S})$ (Lema 3) e que $ub(\mathbb{S}_\pi) \leq 2d_{tb}(\pi)$ para qualquer permutação sem sinal π (Lema 4), podemos construir um algoritmo 2-aproximado para calcular a distância de transposição de bloco de uma permutação sem sinal π (Algoritmo 7).

Lema 3. $ub(\mathbb{S}_\pi) = n - 1 - c(\mathbb{S}_\pi)$.

Lema 4. $ub(\mathbb{S}_\pi) \leq 2d_{tb}(\pi)$.

Algorithm 7: Algoritmo 2-aproximado adaptado de Mahajan *et al.* [40].

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de transposições de bloco aplicadas para ordenar π.

1 Construa $\mathbb{S}_\pi$;
2 Construa o grafo $G_{\mathbb{S}_\pi}$;
3 Calcule $c(\mathbb{S}_\pi)$ usando o algoritmo de programação dinâmica;
4 **retorna** $n - 1 - c(\mathbb{S}_\pi)$;

Construir a $\mathbb{S}_\pi$ toma tempo $O(n)$, enquanto que construir $G_{\mathbb{S}_\pi}$ toma tempo $O(|V| + |E|)$. Dado que $|E| \in O(|V|^2)$ e que $|V| \in O(n)$, nós temos que construir $G_{\mathbb{S}_\pi}$ toma tempo $O(n^2)$. Por fim, calcular $c(\mathbb{S}_\pi)$ toma tempo $O(n^3)$, logo o Algoritmo 7 executa em tempo $O(n^3)$.

Bein *et al.* [5] desenvolveram em algoritmo 2-aproximado baseando-se em outro problema, chamado Problema da Deleção de Blocos. Dada uma subsequência y de π, dizemos que uma sequência $A_1, \ldots, A_m$ de subsequências disjuntas de y é uma sequência de deleção de blocos para y se:

1. A_i é um bloco em $y - \bigcup_{u=1}^{i-1} A_u$ para todo $i = 2, \ldots, m$;

2. e $y - \bigcup_{u=1}^{m} A_u$ é uma sequência crescente.

O Problema da Deleção de Blocos consiste em encontrar a menor sequência de deleção de blocos para a subsequência $y = \pi_1\ \pi_2\ \ldots\ \pi_n$ de π. O tamanho dessa sequência é denotado por $db(\pi)$.

Exemplo 8. *Seja* $\pi = (1\ 4\ 2\ 5\ 3)$. *O Problema da Deleção de Blocos consiste em encontrar a menor sequência de deleção de blocos para a subsequência* $y = 1\ 4\ 2\ 5\ 3$. *Uma possível solução neste caso seria deletar o bloco* 2, *obtendo a subsequência* $1\ 4\ 5\ 3$, *e depois deletar o bloco* $4\ 5$, *obtendo a subsequência crescente* $1\ 3$. *Ou seja, tal solução é dada pela sequência de deleção de blocos* A_1, A_2 *tal que* $A_1 = 1$ *e* $A_2 = 4\ 5$, *portanto* $db(\pi) = 2$.

Dada uma subsequência y de π, dizemos que uma sequência $A_1, \ldots, A_m$ de subsequências disjuntas de y é uma sequência de deleção completa de blocos para y se:

1. A_i é um bloco em $y - \bigcup_{u=1}^{i-1} A_u$ para todo $i = 2, \ldots, m$;

2. e $y - \bigcup_{u=1}^{m} A_u$ é a sequência vazia.

O Problema da Deleção Completa de Blocos consiste em encontrar a menor sequência de deleção completa de blocos para a subsequência $y = \pi_1\ \pi_2\ \ldots\ \pi_n$ de π. O tamanho dessa sequência é denotado por $dcb(\pi)$.

Exemplo 9. *Seja* $\pi = (1\ 4\ 2\ 5\ 3)$. *O Problema da Deleção Completa de Blocos consiste em encontrar a menor sequência de deleção completa de blocos para a subsequência* $y = 1\ 4\ 2\ 5\ 3$. *Uma possível solução neste caso seria deletar o bloco* 4, *obtendo a subsequência* $1\ 2\ 5\ 3$, *depois deletar o bloco* 5, *obtendo a subsequência* $1\ 2\ 3$, *e, finalmente, deletar o bloco* $1\ 2\ 3$, *obtendo a sequência vazia. Ou seja, tal solução é dada pela sequência de deleção completa de blocos* A_1, A_2, A_3 *tal que* $A_1 = 4$, $A_2 = 5$ *e* $A_3 = 1\ 2\ 3$, *portanto* $dcb(\pi) = 3$.

Seja $d_{i,j}$ o tamanho da menor sequência de deleção completa de blocos para a *substring* $\pi_{i\,\ldots\,j}$ de uma permutação $\pi \in S_n$. Não é difícil ver que $d_{1,n} = dcb(\pi)$. Bein *et al.* [5] demonstraram que a seguinte relação de recorrência vale: $d_{i,i} = 1$, $d_{i,j} = 0$ para $i > j$, e

$$d_{i,j} = \begin{cases} \min\{1 + d_{i+1,j}, d_{i+1,k-1} + d_{k,j}\} & \text{se } \exists\, k \in \{i+1, \ldots, j\} \text{ tal que } \pi_k = \pi_i + 1 \\ 1 + d_{i+1,j} & \text{caso contrário} \end{cases}$$

para $i < j$, $1 \leq i, j \leq n$. Essa relação de recorrência dá origem a um algoritmo de programação dinâmica $O(n^2)$ para calcular $d_{1,n}$[1].

Dada uma permutação $\pi \in S_n$, estendemo-na com os elementos $\pi_0 = 0$ e $\pi_{n+1} = n + 1$ e definimos um grafo acíclico direcionado $G' = (V, E)$ com peso nas arestas tal que:

1. $V = \{0, 1, 2, \ldots, n + 1\}$ e $E = \{(i, j) : i < j \text{ e } \pi_i < \pi_j\}$;
2. O peso de uma aresta $(i, j) \in E$ é igual a $d_{i+1,j-1}$.

Observe que existe um caminho $0, i_1, i_2, \ldots, i_k, n + 1$ se, e somente se, $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ é uma subsequência crescente de π. Além disso, o peso da aresta (i_l, i_{l+1}), $1 \leq l \leq k - 1$, é igual ao número mínimo de deleções de blocos necessárias para deletar os elementos entre π_{i_l} e $\pi_{i_{l+1}}$. Desse modo, se $db(\pi) = m$, então deve existir um caminho de 0 a $n + 1$ em G' cuja soma dos pesos das arestas é igual a m, e vice-versa. Como Bein *et al.* [5] mostraram que $db(\pi) \leq 2d_{tb}(\pi)$ para qualquer permutação sem sinal π (Lema 5), podemos construir um algoritmo 2-aproximado para calcular a distância de transposição de bloco de uma permutação sem sinal π (Algoritmo 8).

Algorithm 8: Algoritmo 2-aproximado adaptado de Bein *et al.* [5].

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de transposições de bloco aplicadas para ordenar π.

1 Calcule $d_{i,j}$ para todo $1 \leq i,j \leq n$ usando o algoritmo de programação dinâmica;
2 Construa o grafo G' e encontre um caminho mínimo $0 = i_0 < i_1 < \cdots < i_l = n + 1$ entre 0 e $n + 1$ em G';
3 Seja $p = \sum_{u=1}^{l} d_{i_{u-1}+1,i_{u-1}}$ a soma dos pesos das arestas do caminho mínimo encontrado;
4 **retorna** p;

Lema 5. $db(\pi) \leq 2d_{tb}(\pi) - 1$.

Usando um algoritmo de caminhos mínimos em grafos direcionados acíclicos, podemos encontrar um caminho mínimo entre 0 e $n + 1$ em tempo $O(|V| + |E|)$ [14, Capítulo 24.2]. Dado que $|E| \in O(|V|^2)$ e $|V| \in O(n)$, nós temos que é possível encontrar tal caminho em tempo $O(n^2)$. Sabendo que calcular $d_{i,j}$ para todo $1 \leq i,j \leq n$ toma tempo $O(n^2)$ e que calcular a soma dos pesos das arestas do caminho mínimo encontrado toma tempo $O(n)$, nós podemos concluir que o Algoritmo 8 executa em tempo $O(n^2)$.

[1] A princípio, pode parecer que a complexidade do algoritmo de programação dinâmica é $O(n^3)$ uma vez que encontrar $k \in \{i+1, \ldots, j\}$ tal que $\pi_k = \pi_i + 1$ toma tempo $O(n)$. Entretanto, é possível encontrar k em tempo $O(1)$ usando a permutação inversa de π.

3.3 Problema da Ordenação por Reversões e Transposições

O Problema da Ordenação por Reversões e Transposições consiste em determinar a menor sequência de reversões e transposições que ordena uma determinada permutação sem sinal π. O tamanho dessa sequência é denominado como a distância de reversão e transposição de π, denotada por $d_{rt}(\pi)$. Nesta seção, iremos apresentar dois algoritmos aproximados para este problema, um algoritmo 3-aproximado proposto por Walter, Dias e Meidanis [46] e um algoritmo $2k$-aproximado proposto por Rahman, Shatabda e Hasan [41].

Dada uma permutação π em S_n, nós a estendemos com dois elementos $\pi_0 = 0$ e $\pi_{n+1} = n+1$. A permutação estendida continua sendo denotada por π. Um *breakpoint* de uma permutação π em S_n é um par de elementos adjacentes (π_i, π_{i+1}) tal que $|\pi_{i+1} - \pi_i| \neq 1$, $0 \leq i \leq n$. O número de *breakpoints* de π é denotado por $b_{rt}(\pi)$.

Exemplo 10. *Seja* $\pi = (0\ 4\ 5\ 3\ 2\ 1\ 6)$ *uma permutação estendida. Nós temos que os pares* $(0, 4)$, $(5, 3)$ *e* $(1, 6)$ *são* breakpoints*, portanto* $b_{rt}(\pi) = 3$.

Note que $b_{rt}(\pi) = 0$ se, e somente se, $\pi = \iota$. Pelo fato de uma reversão remover no máximo dois *breakpoints* de uma permutação e de uma transposições remover no máximo três *breakpoints* de uma permutação, o seguinte lema pode ser derivado trivialmente.

Lema 6. *Para toda permutação sem sinal* π, $d_{rt}(\pi) \geq \frac{b_{rt}(\pi)}{3}$.

Uma *strip* de uma permutação π é uma sequência de elementos $\pi_i\ \pi_{i+1}\ \ldots\ \pi_j$, $0 \leq i \leq j \leq n+1$, tal que:

(i) $i = 0$ ou $i > 0$ e (π_{i-1}, π_i) é um *breakpoint*;

(ii) $j = n+1$ ou $j < n+1$ e (π_j, π_{j+1}) é um *breakpoint*;

(iii) e nenhum par (π_k, π_{k+1}), $i \leq k \leq j-1$, é um *breakpoint*.

Uma *strip* com mais de um elemento é dita ser decrescente caso seus elementos formem uma sequência decrescente e é dita ser crescente caso contrário.

Exemplo 11. *Seja* π *a permutação do exemplo 10. Nós temos que* 0, $4\ 5$ *e* 6 *são strips crescentes e que* $3\ 2\ 1$ *é uma strip decrescente.*

Walter, Dias e Meidanis [46] desenvolveram um algoritmo 3-aproximado (Algoritmo 9) cuja ideia básica é, a cada iteração, aumentar a primeira *strip* (da esquerda para direita) da permutação sem introduzir novos *breakpoints*. O algoritmo avança da seguinte forma: dado que a

primeira *strip* da permutação ou possui um único elemento ou é crescente, nós temos que se π_i for o maior (e último) elemento da primeira *strip*, então o elemento $\pi_j = \pi_i + 1$ está a direita de π_i, isto é, $i < j$. Logo, se π_j for o primeiro elemento de uma *strip*, o algoritmo aplica uma transposição. Caso contrário, o algoritmo aplica uma reversão. Como o algoritmo remove pelo menos um *breakpoint* a cada iteração, pelo Lema 6 nós temos que ele é uma 3-aproximação. Além disso, dado que encontrar qual operação aplicar a cada iteração toma tempo $O(n)$, nós temos que o algoritmo executa em tempo $O(n^2)$.

Algorithm 9: Algoritmo 3-aproximado proposto por Walter, Dias e Meidanis [46]

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de reversões e transposições aplicadas para ordenar π.

```
1  d ← 0;
2  enquanto π ≠ ι faça
3  |  Seja π_i o último elemento da primeira strip de π e seja π_j = π_i + 1;
4  |  se π_j pertence a strip crescente π_j π_{j+1} ... π_k então
5  |  |  π ← π ∘ t(i + 1, j, k + 1);
6  |  senão
7  |  |  π ← π ∘ r(i + 1, j);
8  |  fim
9  |  d ← d + 1;
10 fim
11 retorna d;
```

Para desenvolver um algoritmo com fator de aproximação menor do que 3, Rahman, Shatabda e Hasan [41] se basearam no grafo de decomposição de ciclos de uma permutação. O grafo de decomposição de ciclos $G(\pi) = (V, E)$ de uma permutação $\pi \in S_n$ é um grafo cujo conjunto de vértices é formado pelos elementos de π, isto é, $V = \{\pi_0, \ldots, \pi_{n+1}\}$, e cujo conjunto de arestas E é formado pela união de dois conjuntos disjuntos $E_p = \{(\pi_{i-1}, \pi_i) : 1 \leq i \leq n + 1\}$ e $E_c = \{(\pi_i, \pi_j) : 0 \leq i, j \leq n + 1 \text{ e } \pi_i = \pi_j + 1\}$. As arestas pertencentes ao conjunto E_p são ditas ser pretas e as arestas pertencentes ao conjunto E_c, cinzas. A Figura 2 ilustra o grafo de decomposição de ciclos da permutação $\pi = (3\ 2\ 1\ 4\ 6\ 9\ 5\ 7\ 8)$. Por conveniência, as arestas pretas são desenhadas como linhas horizontais e as arestas cinzas como arcos pontilhados.

Um ciclo alternado de $G(\pi)$ é um ciclo com pelo menos duas arestas no qual as arestas alternam de cor. Por razões de simplicidade, um ciclo alternado é referido apenas como ciclo. O grafo de decomposição de ciclos de uma permutação pode ser totalmente decomposto em ciclos com arestas disjuntas. Uma decomposição desse tipo é chamada de decomposição em ciclos de $G(\pi)$. Uma máxima decomposição em ciclos de $G(\pi)$ é aquela que possui um número de

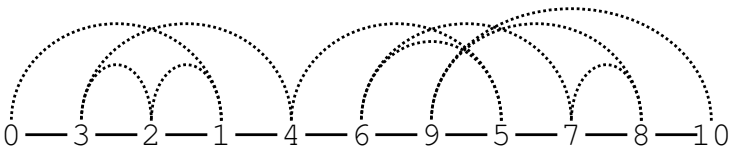


Figura 2: Grafo de decomposição de ciclos da permutação $\pi = (3\ 2\ 1\ 4\ 6\ 9\ 5\ 7\ 8)$.

ciclos maior ou igual ao número de ciclos de qualquer outra decomposição em ciclos de $G(\pi)$. O número de ciclos de uma máxima decomposição em ciclos de $G(\pi)$ é denotado por $c(\pi)$.

Dado um evento de rearranjo ρ, denotamos por $\delta(\pi,\ \rho)$ a variação no número de ciclos de uma máxima decomposição em ciclos de $G(\pi)$ devido à aplicação do evento de rearranjo ρ, isto é, $\delta(\pi,\ \rho) = c(\pi \circ \rho) - c(\pi)$. Rahman, Shatabda e Hasan [41] provaram os seguintes lemas a respeito da variação do número de ciclos de uma máxima decomposição em ciclos de $G(\pi)$.

Lema 7. $\delta(\pi,\ t(i,\ j,\ k)) \leq 2$.

Lema 8. $\delta(\pi,\ r(i,\ j)) \leq 1$.

As $2(n + 1)$ arestas de $G(\pi)$ podem dar origem a, no máxmo, $n + 1$ ciclos. Além disso, não é difícil enxergar que $c(\pi) = n + 1$ se, e somente se, $\pi = \iota$. Dado que, pelos lemas 7 e 8, o máximo acréscimo no número de ciclos de uma máxima decomposição em ciclos de $G(\pi)$ é 2, o seguinte limite segue.

Lema 9. *Para toda permutação sem sinal* π, $d_{rt}(\pi) \geq \frac{n+1-c(\pi)}{2}$.

Dizemos que um ciclo contendo l arestas pretas (tal ciclo também contém l arestas cinzas) é um l-ciclo. Rahman, Shatabda e Hasan [41] provaram que existe uma máxima decomposição em ciclos de $G(\pi)$ que contém todos os 1-ciclos de $G(\pi)$. Sendo assim, seja $c_1(\pi)$ o número de 1-ciclos e seja $c_{2^+}(\pi)$ o número de l-ciclos, $l \geq 2$, em uma máxima decomposição em ciclos de $G(\pi)$. Então, nós temos que $c(\pi) = c_1(\pi) + c_{2^+}(\pi)$. Se uma aresta preta $(\pi_{i-1},\ \pi_i)$ não é um *breakpoint*, então ela dá origem a um 1-ciclo $G(\pi)$. Logo, nós temos que $n + 1 - c_1(\pi) = b_{rt}(\pi)$. A partir desses resultados, podemos derivar um limitante inferior para distância de reversão e transposição alternativo aquele do Lema 9, tal como descrito no lema a seguir.

Lema 10. *Para toda permutação sem sinal* π, $d_{rt}(\pi) \geq \frac{b_{rt}(\pi)-c_{2^+}(\pi)}{2}$.

Definimos uma função g: $E_p \rightarrow \{1,\ 2,\ \ldots,\ n + 1\}$ tal que $g(\pi_{i-1},\ \pi_i) = i$. Desse modo, um l-ciclo C de $G(\pi)$ pode ser representado pelo arranjo $(i_1,\ \ldots,\ i_l)$ tal que $i_k = g(\pi_{i_k-1},\ \pi_{i_k})$

para $1 \leq k \leq l$. Dado que existem vários arranjos possíveis para representar o mesmo ciclo, convencionamos que i_1 é sempre o maior número do arranjo e que a aresta preta (π_{i_1-1}, π_{i_1}) é sempre percorrida da direita para esquerda.

Exemplo 12. *Seja π a permutação cujo grafo de decomposição de ciclos foi ilustrado na Figura 2. Nós temos que o 2-ciclo $(\pi_4, \pi_3, \pi_0, \pi_1)$ pode ser representado pelo arranjo $(4, 1)$.*

Um l-ciclo $(i_1, \ldots, i_l)$, $l \geq 2$, pode ser classificado em três tipos: semi-orientado caso ele possua duas arestas pretas tais que uma delas é percorrida da esquerda para direita e a outra é percorrida da direita para esquerda; não orientado caso $i_1\ i_2\ \ldots\ i_l$ forme uma sequência decrescente; e orientado nos demais casos.

Exemplo 13. *Seja π a permutação cujo grafo de decomposição de ciclos foi ilustrado na Figura 2. Nós temos que o 2-ciclo $(4, 1)$ é um ciclo semi-orientado, o 2-ciclo $(4, 2)$ é um ciclo não-orientado e que o 3-ciclo $(9, 5, 7)$ é um ciclo orientado.*

Rahman, Shatabda e Hasan [41] provaram os seguintes lemas relacionando os eventos de rearranjos possíveis de serem aplicados em cada tipo de ciclo.

Lema 11. *Se $G(\pi)$ possui um ciclo orientado, então é possível aplicar uma transposição em π que quebra este ciclo, gerando 3 novos ciclos.*

Lema 12. *Se $G(\pi)$ possui um ciclo semi-orientado, então é possível aplicar uma reversão em π que quebra este ciclo, gerando 2 novos ciclos.*

Lema 13. *Se $G(\pi)$ possui apenas ciclos não-orientados, então é possível aplicar uma transposição em π que transforma 2 destes ciclos em 2 outros ciclos tais que um deles é orientado.*

O algoritmo aproximado proposto por Rahman, Shatabda e Hasan [41] (Algoritmo 10) ordena uma permutação sem sinal π da seguinte forma. Primeiro ele constrói o grafo de decomposição de ciclos $G(\pi)$. Depois, todos os 1-ciclos de $G(\pi)$ são removidos e, em seguida, é feita uma decomposição em ciclos de $G(\pi)$. Os ciclos resultantes dessa decomposição juntamente com os 1-ciclos removidos dão origem a um conjunto de ciclos denominado H. Por fim, o algoritmo aplica transposições e reversões em π de modo a aumentar o número de ciclos em H até que ele seja igual a $n + 1$, momento em que a permutação estará ordenada.

Idealmente, H deveria ser formado por ciclos oriundos de uma máxima decomposição em ciclos de $G(\pi)$, porém encontrá-la é um problema NP-difícil [10]. Por isso, a decomposição em ciclos realizada na linha 4 é feita por um algoritmo k-aproximado. É importante ressaltar que

Algorithm 10: Algoritmo $2k$-aproximado proposto por Rahman, Shatabda e Hasan [41]

Entrada: Uma permutação $\pi \in S_n$.
Saída: Número de reversões e transposições aplicadas para ordenar π.

```
1  d ← 0;
2  Construa G(π);
3  Remova todos os 1-ciclos de G(π);
4  Realize uma decomposição em ciclos de G(π);
5  Construa um conjunto de ciclos H contendo os 1-ciclos removidos e os ciclos resultantes
   da decomposição;
6  enquanto H não possui n + 1 ciclos faça
7  |  se H possui um ciclo orientado então
8  |  |  Aplique em π a transposição do Lema 11;
9  |  senão se H possui um ciclo semi-orientado então
10 |  |  Aplique em π a reversção do Lema 12;
11 |  senão
12 |  |  Aplique em π a transposição do Lema 13;
13 |  fim
14 |  Atualize H com os novos ciclos criados;
15 |  d ← d + 1;
16 fim
17 retorna d;
```

o problema da máxima decomposição em ciclos de $G(\pi)$ não é tratado na literatura como um problema de maximização, mas sim como um problema de minimização cuja função objetivo é minimizar o valor de $b_{rt}(\pi) - c$, sendo c o número de ciclos da decomposição [11]. Por essa razão, dizer que um algoritmo de decomposição em ciclos de $G(\pi)$ possui fator de aproximação k significa dizer que $\frac{b_{rt}(\pi)-c}{b_{rt}(\pi)-c(\pi)} \leq k$.

Seja c_1^H o número de 1-ciclos e seja c_2^H o número de l-ciclos, $l \geq 2$, existentes inicialmente em H. No pior caso, o Algoritmo 10 cria 2 novos ciclos a cada 2 iterações, o que resulta em uma média de 1 novo ciclo por iteração. Isso significa que o Algoritmo 10 ordena a permutação π em no máximo $n + 1 - (c_1^H + c_2^H) = b_{rt}(\pi) - c_2^H$ iterações. Logo, pelo Lema 10, nós temos que o fator de aproximação do Algoritmo 10 é igual a $2\frac{b_{rt}(\pi)-c_2^H}{b_{rt}(\pi)-c_{2^+}(\pi)}$. Como a decomposição em ciclos de $G(\pi)$ foi feita por um algoritmo k-aproximado, nós temos que $\frac{b_{rt}(\pi)-c_2^H}{b_{rt}(\pi)-c_{2^+}(\pi)} \leq k$, portanto o fator de aproximação é igual a $2k$. Para o melhor valor de k conhecido [39], que é igual a 1.4193 $+ \epsilon$, $\epsilon > 0$, nós temos que o fator de aproximação do Algoritmo 10 é igual a 2.8386 $+ \gamma$, para qualquer $\gamma > 0$.

4 Cronograma

A Tabela 2 descreve a distribuição das atividades a serem realizadas durante a execução desta proposta. As atividades foram dividas em grupos que consideram tanto os objetivos levantados na Seção 2 quanto os requisitos que o aluno deve cumprir para obter o grau de doutor pelo Instituto de Computação da UNICAMP.

	2012	**2013**		**2014**		**2015**		**2016**
Grupo	ago-dez	jan-jun	jul-dez	jan-jun	jul-dez	jan-jun	jul-dez	jan-jul
1	a	a		b				c
2	a	b		c				
3		a	a,b	c				
4				a	a,b,d	b,c,d	c,d	d

Tabela 2: Cronograma de atividades.

Grupo 1: atividades relacionadas ao cumprimento dos requisitos para a obtenção do grau de doutor.

a) Obtenção dos créditos obrigatórios (incluindo os necessários para dispensa do Exame de Qualificação Geral) e realização de estágio docente.

b) Defesa do Exame de Qualificação Específico de Doutorado.

c) Escrita e defesa da tese.

Grupo 2: atividades relacionadas à Seção 2.1.

a) Obtenção de novos resultados teóricos relativos aos algoritmos alternativos para o Problema da Ordenação por Transposições.

b) Comparação experimental entre algoritmos alternativos e os melhores algoritmos conhecidos, que se baseiam no grafo de ciclos.

c) Publicação de artigo em periódico.

Grupo 3: atividades relacionadas à Seção 2.2.

a) Desenvolvimento de heurísticas.

b) Avaliação experimental das heurísticas desenvolvidas.

c) Publicação de artigo em periódico.

Grupo 4: atividades relacionadas à Seção 2.3.

a) Desenvolvimento de algoritmos aproximados e heurísticas para o Problema da Ordenação por Reversões Curtas.

b) Desenvolvimento de algoritmos aproximados e heurísticas para o Problema da Ordenação por Blocos.

c) Desenvolvimento de algoritmos aproximados e heurísticas para o Problema da Ordenação por Reversões e Transposições.

d) Publicação de artigos em conferências e periódicos.

Referências

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

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

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

[4] W. W. Bein, L. L. Larmore, S. Latifi, and I. H. Sudborough. Block sorting is hard. *International Journal of Foundations of Computer Science*, 14(3):425–437, 2003.

[5] W. W. Bein, L. L. Larmore, L. Morales, and I. H. Sudborough. A quadratic time 2-approximation algorithm for block sorting. *Theoretical Computer Science*, 410(8-10):711–717, 2009.

[6] M. Benoît-Gagné and S. Hamel. A new and faster method of sorting by transpositions. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM'2007)*, volume 4580 of *Lecture Notes in Computer Science*, pages 131–141, London, Ontario, Canada, 2007. Springer-Verlag.

[7] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms*

(ESA'2002), volume 2461 of *Lecture Notes in Computer Science*, pages 200–210, Rome, Italy, 2002. Springer-Verlag.

[8] L. Bulteau, G. Fertin, and I. Rusu. Pancake flipping is hard. In *37th International Symposium on Mathematical Foundations of Computer Science (MFCS'2012)*, volume 7464 of *Lecture Notes in Computer Science*, pages 247–258, Bratislava, Slovakia, 2012. Springer-Verlag.

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

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

[11] A. Caprara and R. Rizzi. Improved approximation for breakpoint graph decomposition and sorting by reversals. *Journal of Combinatorial Optimization*, 6(2):157–182, 2002.

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

[13] D. S. Cohen and M. Blum. On the problem of sorting burnt pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[15] U. Dias and Z. Dias. Sorting by prefix reversals and prefix transpositions. To be published.

[16] U. Dias and Z. Dias. Heuristics for the transposition distance problem. *Journal of Bioinformatics and Computational Biology*, 11:1350013, 2013.

[17] Z. Dias and J. Meidanis. Sorting by prefix transpositions. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE'2002)*, volume 2476 of *Lecture Notes in Computer Science*, pages 65–76, Lisbon, Portugal, 2002. Springer-Verlag.

[18] H. Dweighter. Problem e2569. *American Mathematical Monthly*, 82:1010, 1975.

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

[20] S. Even and O. Goldreich. The minimum-length generator sequence problem is NP-hard. *Journal of Algorithms*, 2(3):311–313, 1981.

[21] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. The MIT Press, 2009.

[22] J. Fischer and S. W. Ginzinger. A 2-approximation algorithm for sorting by prefix reversals. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA'2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 415–425, Mallorca, Spain, 2005. Springer-Verlag.

[23] G. R. Galvão. Uma ferramenta de auditoria para algoritmos de rearranjo de genomas. Master's thesis, University of Campinas, Campias, Brazil, 2012.

[24] G. R. Galvão and Z. Dias. Rearrangement distance database. `http://mirza.ic.unicamp.br:8080`.

[25] G. R. Galvão and Z. Dias. GRAAu: Genome Rearrangement Algorithm Auditor. In *Proceedings of the 4th International Conference on Bioinformatics and Computational Biology (BICoB'2012)*, pages 97–101, Las Vegas, Nevada, USA, 2012. Curran Associates, Inc.

[26] G. R. Galvão and Z. Dias. On the approximation ratio of algorithms for sorting by transpositions without using cycle graphs. In *Proceedings of the 7th Brazilian Symposium on Bioinformatics (BSB'2012)*, volume 7049 of *Lecture Notes in Computer Science*, pages 25–36, Campo Grande, Mato Grosso do Sul, Brazil, 2012. Springer-Verlag.

[27] O. Gascuel. *Mathematics of Evolution and Phylogeny*. Oxford University Press, Inc., New York, NY, USA, 2005.

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

[29] S. A. Guyer, L. S. Heath, and J. P. C. Vergara. Subsequence and run heuristics for sorting by transpositions. Technical Report TR-97-20, Virginia Polytechnic Institute & State University, 1997.

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

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

[32] L. S. Heath and J. P. C. Vergara. Sorting by bounded blockmoves. *Discrete Applied Mathematics*, 88:181–206, 1998.

[33] L. S. Heath and J. P. C. Vergara. Sorting by short blockmoves. *Algorithmica*, 28(3):323–354, 2000.

[34] L. S. Heath and J. P. C. Vergara. Sorting by short swaps. *Journal of Computational Biology*, 10(5):775–789, 2003.

[35] M.R Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.

[36] H. Jiang, D. Zhu, and B. Zhu. A $(1+\epsilon)$-approximation algorithm for sorting by short block-moves. *Theoretical Computer Science*, 439:1–8, 2012.

[37] J. D. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1-2):80–110, 1995.

[38] S. Latifi. How can permutations be used in the evaluation of zoning algorithms? *International Journal of Pattern Recognition and Artificial Intelligence*, 10(3):223–237, 1996.

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

[40] M. Mahajan, R. Rama, V. Raman, and S. Vijaykumar. Approximate block sorting. *International Journal of Foundations of Computer Science*, 17(2):337–356, 2006.

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

[42] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(1):406–425, 1987.

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

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

[45] J. P. C. Vergara. *Sorting by bounded permutations*. PhD thesis, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA, 1998.

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

[47] M. E. M. T. Walter, Z. Dias, and J. Meidanis. A new approach for approximating the transposition distance. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'2000)*, pages 199–208, Washington, DC, USA, 2000. IEEE Computer Society.

[48] G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1–7, 1982.