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

Um novo algoritmo 1.375-aproximativo baseado em grupos de permutações para o Problema da Ordenação por Transposições

Luiz Augusto Garcia da Silva

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Orientador
Prof.ª Dr.ª Maria Emília Machado Telles Walter

Brasília
2022

Ficha Catalográfica de Teses e Dissertações

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

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

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

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

Um novo algoritmo 1.375-aproximativo baseado em grupos de permutações para o Problema da Ordenação por Transposições

Luiz Augusto Garcia da Silva

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Prof.ª Dr.ª Maria Emília Machado Telles Walter (Orientador)
CIC/UnB

Prof. Dr. Mauricio Ayala Rincón
MAT/UnB

Prof. Dr. Luis Antonio Kowada
IC/UFF

Prof. Dr. Zanoni Dias
IC/Unicamp

Prof. Dr. Ricardo Pezzuol Jacobi
Coordenador do Programa de Pós-graduação em Informática

Brasília, 31 de maio de 2022

Dedicatória

Esta tese é dedicada ao meu pai, Paulo Venâncio (*in memoriam*), que nunca mediu esforços para que seus filhos tivessem uma boa educação.

Agradecimentos

À minha esposa, Juliana Capella, pelo apoio, incentivo e paciência durante todo o desenvolvimento deste trabalho.

À Marta Dias de Almeida, cujos conselhos, nos momentos críticos, me incentivaram a não desistir deste trabalho.

Ao Prof. Luis Antonio Kowada, por toda a ajuda e pelos ensinamentos.

Ao Prof. Noraí Rocco, pelos conselhos e pelas lições particulares de Álgebra.

À Prof.ª Maria Emília Walter, pela orientação, confiança e apoio, mesmo nos momentos mais difíceis.

Resumo

O Problema da Ordenação por Transposições (SBT, sigla dos termos em língua inglesa *Sorting By Transpositions*) é um problema clássico em rearranjos de genoma. Em 2012, Bulteau, Fertin e Rusu provaram que o SBT é $\mathcal{NP}$-difícil e o melhor algoritmo de aproximação com uma razão de 1.375 foi proposto em 2006 por Elias e Hartman. Seu algoritmo emprega *simplificação*, uma técnica usada para transformar uma permutação de entrada π em uma *permutação simples* $\hat{\pi}$, presumivelmente mais fácil de lidar. A permutação $\hat{\pi}$ é obtida inserindo novos símbolos em π de forma que o limite inferior da distância de transposição de π seja mantido em $\hat{\pi}$. A simplificação garantidamente mantém o limite inferior, mas não a distância de transposição. A sequência de transposições que ordena $\hat{\pi}$ pode ser "imitada" para ordenar π. Nesta tese, primeiro, mostramos que o algoritmo de Elias e Hartman (Algoritmo EH) pode requerer uma transposição extra acima da razão de aproximação de 1.375, dependendo de como a permutação de entrada é simplificada. Em seguida, usando uma abordagem algébrica, propomos um novo limite superior para a distância de transposição e um novo algoritmo de aproximação de 1.375 para o SBT, que não emprega simplificação e que garante a razão de aproximação de 1.375 para todo o grupo simétrico S_n. Implementamos o Algoritmo EH e o algoritmo proposto nesta tese. Em relação à implementação do Algoritmo EH, duas novas falhas foram identificadas e precisaram ser corrigidas. Testamos ambos os algoritmos com todas as permutações de tamanho n, $2 \leq n \leq 12$. Os resultados mostraram que o Algoritmo EH excede a razão de aproximação de 1.375 para permutações com tamanho maior que 7. Em seguida, investigamos o desempenho de ambas as implementações em permutações longas de comprimento máximo 500. Por fim, apresentamos os resultados obtidos de uma investigação visando encontrar uma solução aproximada para o SBT, com razão inferior a 1.375. Ao procurar por resultados que permitissem o desenvolvimento de uma solução $1.33\overline{3}$-aproximativa, apenas não foram encontradas sequências de transposições, proporcionando essa razão, para 11 casos, de um universo de ≈ 545.000. A conclusão da investigação foi de que, para baixar a razão de aproximação 1.375, será necessário buscar por sequências de transposições muito longas, o que pode ser impraticável, dependendo dos recursos computacionais disponíveis e da técnica utilizada.

Palavras-chave: Problema da Distância de Transposição, Ordenação por Transposições, rearranjos de genomas, algoritmos de aproximação, grupos de permutações

Abstract

Sorting by Transpositions (SBT) is a classical problem in genome rearrangements. In 2012, Bulteau, Fertin e Rusu have proven that the SBT is $\mathcal{NP}$-hard, and the best approximation algorithm with a 1.375 ratio was proposed in 2006 by Elias and Hartman. Their algorithm employs *simplification*, a technique used to transform an input permutation π into a *simple permutation* $\hat{\pi}$, presumably easier to handle with. The permutation $\hat{\pi}$ is obtained by inserting new symbols into π in a way that the lower bound of the transposition distance of π is kept on $\hat{\pi}$. The simplification is guaranteed to keep the lower bound, not the transposition distance. A sequence of operations sorting $\hat{\pi}$ can be mimicked to sort π. In this thesis, we first show that the algorithm of Elias and Hartman (EH algorithm) may require one extra transposition above the approximation ratio of 1.375, depending on how the input permutation is simplified. Next, using an algebraic approach, we propose a new upper bound for the transposition distance and a new 1.375-approximation algorithm to solve SBT skipping simplification and ensuring the approximation ratio of 1.375 for all symmetric group S_n. We implemented our algorithm and EH's. Regarding the implementation of the EH algorithm, two issues needed to be fixed. We tested both algorithms against all permutations of size n, $2 \leq n \leq 12$. The results showed that the EH algorithm exceeds the approximation ratio of 1.375 for permutations with a size greater than 7. Next, we investigate the performance of both implementations on long permutations of maximum length 500. Finally, we present the results obtained from an investigation aimed at finding an approximate solution for SBT with an approximation ratio lower than 1.375. When looking for results that would allow the development of a $1.33\overline{3}$-approximate solution, we only have not found transposition sequences, providing this ratio, for 11 cases, out of an universe of $\approx 545,000$. The conclusion of the investigation was that, in order to lower the approximation ratio of 1.375, it will be necessary to search for very long transposition sequences, which may be impractical, depending on the available computational resources and the technique used.

Keywords: Transposition Distance Problem, Sorting by Transpositions, genome rearrangements, approximation algorithms, permutation groups

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sabe-se, de pesquisas anteriores, que os genomas de diferentes espécies podem apresentar essencialmente o mesmo conjunto de genes em suas fitas de DNA, porém em ordens diferentes [5, 6], sugerindo a ocorrência de eventos mutacionais que afetam grandes porções de DNA. Estes eventos são presumivelmente raros e, portanto, podem prover pistas importantes para a reconstrução da história evolucionária entre diferentes espécies [7, 8]. Um destes eventos é a *transposição*, a qual troca de posição dois blocos adjacentes de genes em um cromossomo. Se considerarmos que não existem genes duplicados e que podemos representar um cromossomo como uma permutação sobre um conjunto de números inteiros, onde cada gene é representado por um número, então o Problema da Distância de Transposição (TDP, sigla dos termos em língua inglesa *Transposition Distance Problem*) consiste em encontrar o número mínimo de transposições (*distância de transposição*) necessárias para transformar um cromossomo em outro. O TDP pode ser reduzido ao Problema da Ordenação por Transposições (SBT, dos termos em língua inglesa *Sorting by Transpositions*), onde a permutação alvo é a permutação identidade ι. A distância de transposição entre uma permutação qualquer π sobre n símbolos e ι é denotada como $d(\pi)$.

O primeiro algoritmo de aproximação para o SBT foi concebido em 1998 por Bafna e Pevzer [9], com uma razão de aproximação 1.5. Para chegar a este resultado, Bafna e Pevzer investigaram as propriedades de uma estrutura chamada grafo de ciclos, que posteriormente se tornou a estrutura mais classicamente estudada na literatura relacionada ao SBT. Em 2006, Elias e Hartman [10] apresentaram um algoritmo 1.375-aproximativo para o SBT (Algoritmo EH) com complexidade de tempo $O(n^2)$, sendo este o melhor algoritmo de aproximação conhecido até então para o SBT. Em um estudo posterior, a complexidade de tempo do algoritmo de Elias e Hartman foi melhorada para $O(n \log n)$ por Cunha e outros [11]. Outras melhorias, incluindo heurísticas, foram propostas por Dias e Dias [4, 12]. Em 2012, Bulteau, Fertin e Rusu [13] demonstraram que o SBT é

$\mathcal{NP}$-difícil.

Outros estudos, porém, foram feitos usando estruturas diferentes. Por exemplo, Hausen e outros [14] estudaram o SBT usando uma estrutura chamada *grafo tórico*, que foi previamente concebida por Erikson e outros [15], no trabalho em que eles apresentaram o limite superior de $\left\lfloor \frac{2n-2}{3} \right\rfloor$ para o diâmetro de transposição, o melhor conhecido na literatura. Seja π uma permutação sobre n símbolos, o *diâmetro de transposição*, denotado $d(n)$, do grupo simétrico de ordem n, denotado S_n, corresponde ao $max_{\pi \in S_n} d(\pi)$. Por outro lado, Galvão e Dias [16] estudaram soluções aproximadas para o SBT usando três estruturas diferentes: *códigos de permutação*, um conceito previamente introduzido por Benoît-Gagné e Hamel [17]; *diagrama de pontos-de-quebra*, introduzida por Walter e outros [1]; e *subsequência crescente mais longa*, introduzida por Guyer e outros [18]. Rusu [19], por sua vez, usou uma estrutura chamada *log-list*, anteriormente concebida por Sleator e Tarjan [20], para derivar outro algoritmo 1.375-aproximativo para o SBT com complexidade de tempo $O(n \log n)$.

Além desses, recentemente outros trabalhos têm sido propostos envolvendo variações do evento mutacional de transposição. Como exemplo, podemos citar que Lintzmayer e outros [21] estudaram o Problema da Ordenação de Permutações por Transposições Por Prefixos e Sufixos, bem como outros problemas combinando variações da transposição com outros eventos mutacionais, por exemplo, reversão. Oliveira e outros [22], por sua vez, estudaram a distância de transposição entre dois genomas considerando regiões intergênicas, um problema o qual chamaram de Problema da Ordenação de Permutações por Transposições Intergênicas.

Meidanis e Dias [23] foram os primeiros autores a propor o uso de uma abordagem algébrica para resolver o SBT, como uma alternativa às abordagens baseadas no grafo de ciclos. O objetivo era o de prover uma abordagem mais formal para solucionar problemas de rearranjo de genomas usando resultados já conhecidos da teoria de grupos de permutações. Mira e outros [2] mostraram a viabilidade do uso de abordagem algébrica para solucionar o SBT através da formalização do algoritmo 1.5-aproximativo de Bafna e Pevzner [9], usando apenas ferramental algébrico.

Sobre os algoritmos baseados em grafo de ciclos, para o SBT, após o trabalho de Bafna e Pevzner [9], o uso da simplificação [24, 25, 26] tornou-se predominante. A simplificação é uma ferramenta que foi introduzida com o objetivo de tornar mais fácil a lida com ciclos longos no grafo de ciclos. Basicamente, consiste em inserir novos símbolos na permutação original, de forma que a permutação simples resultante contenha apenas ciclos curtos. A ordenação da permutação simples pode ser "imitada" para ordenar a permutação original. No Capítulo 3, será mostrado um efeito colateral da simplificação que, em última instância, faz com que o Algoritmo EH, em certos casos, produza uma

transposição extra, além da razão 1.375 prometida. No melhor de nosso conhecimento, não há na literatura, possivelmente pela predominância da simplificação, ferramental desenvolvido para a lida com ciclos longos. Assim sendo, nesta tese, o uso da simplificação foi evitado. Em seu lugar, preferimos utilizar uma abordagem algébrica, baseada no trabalho de Mira e outros [2]. Nela, é possível lidar com ciclos longos sem a necessidade de inserir novos símbolos na permutação original.

É importante ressaltar que não achamos que a abordagem algébrica deva substituir completamente a representação gráfica, através do grafo de ciclos. Isto, porque para termos uma melhor intuição sobre as propriedades de um ou mais ciclos e seus relacionamentos, é bastante útil, "desenhar" e "visualizar" o grafo de ciclos de uma permutação. Por outro lado, achamos que o uso da abordagem algébrica é mais conveniente para declarar e demonstrar fatos, além de ser útil também na implementação de algoritmos, onde muitas das computações podem ser efetuadas por meio da multiplicação de permutações. Dessa forma, acreditamos que ambas as abordagens são, na verdade, complementares.

Esta tese está organizada da seguinte forma: no Capítulo 2, formalizaremos o SBT e depois faremos uma revisão sobre a abordagem clássica utilizada para desenvolver soluções aproximadas para o SBT, baseada no grafo de ciclos, e uma revisão do algoritmo de Elias e Hartman. A seguir, será apresentada uma breve revisão dos conceitos de grupos de permutações utilizados nesta tese, seguida de uma formalização algébrica para o SBT. Por fim, mostraremos uma correspondência entre as duas abordagens. No Capítulo 3, apresentaremos nossa motivação para o desenvolvimento de uma nova solução 1.375-aproximativa para o SBT, que consiste de exemplos de permutações para as quais o Algoritmo EH produz uma transposição extra acima da razão de aproximação prometida de 1.375. Ainda no Capítulo 3, apresentaremos um novo limite superior para o SBT, melhorando o resultado alcançado por Bafna e Pevzner [9, 27] em 1998 e um novo algoritmo 1.375-aproximativo para o SBT. Este novo algoritmo não produz transposições extras e garante a razão de aproximação 1.375 para todo o grupo simétrico S_n. No Capítulo 4, serão apresentados resultados experimentais, para as permutações de comprimento n, $2 \leq n \leq 12$, obtidos de implementações do Algoritmo EH e do algoritmo proposto nesta tese. A porcentagem de distâncias computadas que são iguais à distância de transposição calculadas pelo Algoritmo EH e o nosso serão comparados com outras disponíveis na literatura. O desempenho das implementações de ambos os algoritmos com permutações mais longas, também foi investigado. Nesse contexto, serão apresentados resultados de experimentos envolvendo a ordenação de permutações geradas aleatoriamente, de comprimento variando de 20 a 500, e comparamos os resultados com experimentos semelhantes realizados em outros estudos. Por fim, serão mostrados outras duas falhas encontradas no Algoritmo EH, descobertas durante a sua implementação: um afetando ambas as versões publicadas [10, 28] e outro

afetando apenas a versão publicada em [10]. No Capítulo 5, serão apresentados resultados obtidos de uma investigação cujo objetivo foi o de desenvolver uma nova solução aproximada para o SBT, com razão de aproximação inferior a 1.375, a princípio $1.33\overline{3}$. Finalmente, conclusões e trabalhos futuros serão discutidos no Capítulo 6.

Capítulo 2

Abordagens clássica e algébrica para o SBT

Neste capítulo, formalizaremos o SBT na Seção 2.1. Depois, na Seção 2.2, revisaremos a abordagem clássica utilizada no desenvolvimento de soluções aproximadas para o SBT, baseada no grafo de ciclos e, a seguir, faremos uma revisão do Algoritmo EH. Na Seção 2.3, será feita uma breve revisão de conceitos de grupos de permutações, onde nos restringiremos apenas naqueles que são relevantes no âmbito desta tese. Em seguida, mostraremos uma formalização algébrica para o SBT. Por último, na Seção 2.4, será mostrada uma correspondência entre ambas as abordagens, clássica e algébrica.

2.1 Formalização do SBT

Seja $\pi = [\pi_1\ \pi_2 \dots \pi_n]$ uma permutação sobre o conjunto $E = \{1 \dots n\}$. Uma *transposição* $\tau(i, j, k)$, com $1 \leq i < j < k \leq n + 1$, troca de posição os símbolos do intervalo $[\pi_i, \pi_{j-1}]$ com os símbolos do intervalo $[\pi_j, \pi_{k-1}]$. Assim, a aplicação de $\tau(i, j, k)$ sobre π, denotada $\tau(i, j, k) \cdot \pi$, resulta em $[\pi_1 \dots \pi_{i-1}\ \pi_j \dots \pi_{k-1}\ \pi_i \dots \pi_{j-1}\ \pi_k \dots \pi_n]$.

Dadas duas permutações π e σ, o Problema da Distância de Transposição (TDP, sigla dos termos em língua inglesa *Transposition Distance Problem*) corresponde ao problema de encontrar o número mínimo t (i.e., a distância de transposição entre π e σ) tal que a sequência de transposições $\tau_1, \dots, \tau_t$ transforma π em σ, i.e., $\tau_t \dots \tau_1 \cdot \pi = \sigma$. Repare que a distância de transposição entre π e σ é igual à distância de transposição entre a $\sigma^{-1} \cdot \pi$, onde σ^{-1} é a inversa de σ e a permutação identidade $\iota = [1\ 2 \dots n]$. O Problema da Ordenação por Transposições (SBT, dos termos em língua inglesa *Sorting by Transpositions*) corresponde ao problema de encontrar a distância de transposição entre uma permutação π e ι, denotada por $d(\pi)$.

2.2 Grafos de ciclos e o Algoritmo EH

2.2.1 Grafos de ciclos

Na literatura de rearranjo de genomas, a representação gráfica mais amplamente utilizada para permutações, é o grafo de ciclos[1] [9]. Para construir o grafo de ciclos de $\pi = [\pi_1\ \pi_2\ \ldots\ \pi_n]$, primeiro estendemos π com dois símbolos extra $\pi_0 = 0$ e $\pi_{n+1} = n + 1$. Assim, o *grafo de ciclos* de π, denotado por $G(\pi)$, é um grafo dirigido consistindo do conjunto de vértices $\{+0,\ -1,\ +1,\ -2,\ +2,\ \ldots,\ -n,\ +n,\ -(n+1)\}$ e um conjunto de arestas coloridas (que podem ser pretas ou cinza). Para todo $1 \leq i \leq n + 1$, as arestas pretas ligam $-\pi_i$ à $+\pi_{i-1}$. Para todo $0 \leq i \leq n$, as arestas cinza ligam o vértice $+i$ ao vértice $-(i + 1)$. Intuitivamente, as arestas pretas indicam o estado atual dos genes, em relação ao seu posicionamento no cromossomo representado por π, enquanto as arestas cinza indicam a ordem desejada dos genes na segunda permutação, representada por $\iota = [1\ 2\ \ldots\ n]$. Nas figuras abaixo, a direção das arestas são omitidas, pois podem ser facilmente inferidas pela observação dos sinais dos vértices.

Exemplo 1. *A Figura 2.1 mostra o grafo de ciclos de* $\pi = [4\ 3\ 2\ 1\ 8\ 7\ 6\ 5]$ *com* 9 *arestas pretas,* $(-9, +5)$*,* $(-5, +6)$*,* $\ldots$*,* $(-3, +4)$*,* $(-4, +0)$*, e* 9 *arestas cinza,* $(+0, -1)$*,* $(+1, -2)$*,* $(+2, -3)$*,* $\ldots$*,* $(+7, -8)$*,* $(+8, -9)$*.*

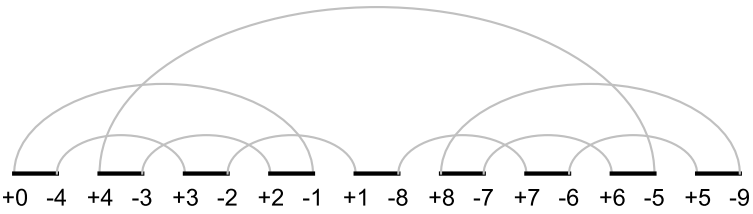

Figura 2.1: $G([4\ 3\ 2\ 1\ 8\ 7\ 6\ 5])$. As arestas pretas são as horizontais.

Ambos graus de entrada e saída de cada vértice em $G(\pi)$ são 1, correspondendo a uma aresta preta entrando no vértice v e outra aresta cinza deixando o mesmo vértice. Isto induz em $G(\pi)$ uma decomposição única em ciclos. Um κ-ciclo é um ciclo C em $G(\pi)$ com κ arestas pretas. Neste caso, também dizemos que o *comprimento* de C corresponde ao valor κ. Além disso, C é dito ser um *ciclo longo*, se $\kappa > 3$, caso contrário, C é dito ser um *ciclo curto*. Se κ é par (ímpar), então dizemos também que C é um ciclo *par* (*ímpar*).

O número máximo de $n+1$ ciclos em $G(\pi)$ é obtido se e somente se π é a permutação identidade ι. Neste caso, cada ciclo é composto exatamente de uma aresta preta e uma aresta cinza. Denote por $c_{\acute{i}mpar}(\pi)$ o número de ciclos ímpares em $G(\pi)$, e $\Delta c_{\acute{i}mpar}(\pi, \tau) =$

[1] Em seu trabalho, Elias e Hartman [10] utilizaram uma estrutura circular equivalente, a qual eles chamaram de grafo de pontos de quebra (em língua inglesa, *breakpoint graph*).

$c_{ímpar}(\tau \cdot \pi) - c_{ímpar}(\pi)$ a variação no número de ciclos ímpares em $G(\pi)$ e $G(\tau \cdot \pi)$, após a *aplicação* de uma transposição τ. Bafna e Pevzner [9] demonstraram o seguinte resultado.

Lema 2 (Bafna e Pevzner [9])**.** $\Delta c_{ímpar}(\pi, \tau) \in \{-2, 0, 2\}$.

Um *μ-movimento* é uma transposição τ tal que $\Delta c_{ímpar}(\pi, \tau) = \mu$. Repare que de acordo com o lema acima, os possíveis movimentos são: 2-movimento, 0-movimento e (-2)-movimento. A partir do Lema 2, Bafna e Pevzner [9] derivaram o seguinte limite inferior na distância de transposição.

Teorema 3 (Bafna e Pevzner [9])**.** $d(\pi) \geq \frac{n+1-c_{ímpar}(\pi)}{2}$.

As arestas pretas de $G(\pi)$ podem ser numeradas de 1 até $n+1$ pela atribuição de um rótulo i a cada aresta preta $(-\pi_i, +\pi_{i-1})$. Um κ-ciclo C visitando as arestas pretas $i_1, \ldots, i_\kappa$, na ordem imposta pelo ciclo, pode ser escrita de κ formas diferentes, dependendo da primeira aresta preta visitada. Se não especificado de outra forma, assumiremos que a aresta inicial i_1 de C é escolhida com o maior valor, i.e., i_1 é tal que $i_1 > i_s$, para todo $s \in \{2, \ldots, \kappa\}$. Com esta condição, se $i_1, \ldots, i_\kappa$ é uma sequência decrescente, então C é chamado de *ciclo não-orientado*, caso contrário C é um *ciclo orientado*. Dois pares de arestas pretas são ditos *cruzantes* se existem ciclos $C = (\ldots, a, b, \ldots)$ e $D = (\ldots, e, f, \ldots)$ em $G(\pi)$, tais que $a > e > b > f$ ou $e > a > f > b$. Neste caso, C e D também são ditos serem *ciclos cruzantes*. De maneira similar, as triplas de arestas pretas (a, b, c) e (d, e, f) são *intercalantes* se existem ciclos $C = (\ldots, a, b, c, \ldots)$ e $D = (\ldots, d, e, f, \ldots)$, tais que $a > d > b > e > c > f$ ou $d > a > e > b > f > c$. Neste caso, C e D também são ditos serem *ciclos intercalantes*.

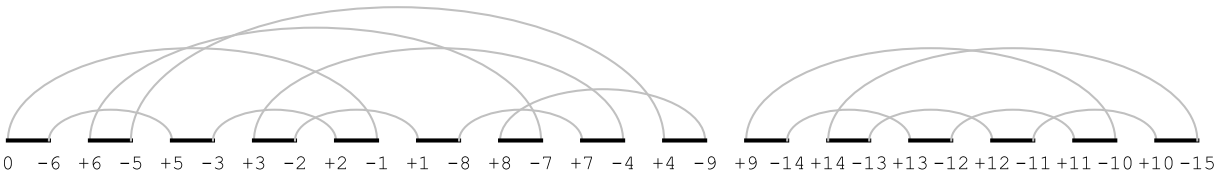

Figura 2.2: $G([6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10])$.

Exemplo 4. *Os ciclos* $(5, 3, 1)$*,* $(8, 6, 4)$*,* $(15, 13, 11)$ *e* $(14, 12, 10)$ *de* $G([6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10])$ *(Figura 2.2) são não-orientados, enquanto que* $(9, 2, 7)$ *é um ciclo orientado. Além disso,* $(5, 3, 1)$ *e* $(8, 6, 4)$ *são ciclos cruzantes e os ciclos* $(15, 13, 11)$ *e* $(14, 12, 10)$ *são ciclos intercalantes.*

2.2.2 Simplificação

A simplificação é uma técnica que foi introduzida com o objetivo de facilitar a lida com ciclos longos de $G(\pi)$ [24]. Ela consiste na inserção de novos elementos, usualmente

fracionários, em π, transformando-a em uma nova *permutação simples* $\hat{\pi}$, de forma que $G(\hat{\pi})$ contém apenas ciclos curtos. Além disso, as posições dos novos símbolos podem variar, mas as inserções devem ser feitas por meio de *transformações seguras*. Após a transformação, os elementos de $\hat{\pi}$ podem ser mapeados para inteiros consecutivos.

Uma transformação de π em $\hat{\pi}$ é dita *segura* se, após a inserção dos novos elementos, o limite inferior do Teorema 3 é mantido, i.e., $n(\pi) - c_{\acute{i}mpar}(\pi) = n(\hat{\pi}) - c_{\acute{i}mpar}(\hat{\pi})$, onde $n(\pi)$ e $n(\hat{\pi})$ denotam o número de arestas pretas em π e $\hat{\pi}$, respectivamente. Se $\hat{\pi}$ é uma permutação obtida de π através de transformações seguras, então dizemos que π e $\hat{\pi}$ são *equivalentes*. Lin e Xue [25] mostraram que toda permutação pode ser transformada em uma permutação simples equivalente. Uma ordenação de $\hat{\pi}$ pode ser "imitada" para ordenar π usando o mesmo número de transposições [24].

É importante notar que uma permutação pode ser simplificada de muitas maneiras diferentes, de forma segura. A Figura 2.3 mostra o grafo de ciclos de uma possível permutação simples obtida da simplificação de [4 3 2 1 8 7 6 5] (Figura 2.1). Para uma descrição completa da simplificação e resultados relacionados, o leitor pode consultar [24, 25, 26].

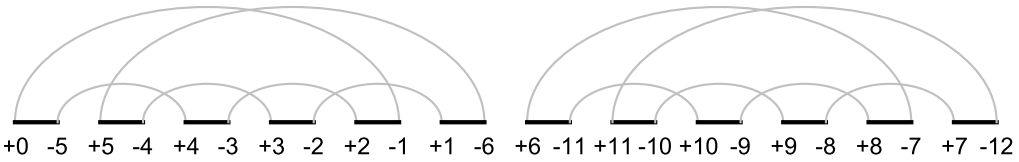

Figura 2.3: $G([5\ 4\ 3\ 2\ 1\ 6\ 11\ 10\ 9\ 8\ 7])$, mapeado de [4.1 4 3 2 1 4.2 8.1 8 7 6 5] usando inteiros consecutivos, obtidos da simplificação de [4 3 2 1 8 7 6 5] (Figura 2.1).

2.2.3 Configurações e componentes

Os conceitos apresentados nesta seção foram originalmente introduzidos por Elias e Hartman [10], no contexto das permutações simples, com foco especial nas 3-*permutações* (permutações cujos grafos de ciclos contêm apenas 3-ciclos), das quais derivaram seus principais resultados. Como nosso trabalho não envolve simplificação, modificamos alguns destes conceitos para que pudessem ser estendidos a qualquer permutação em S_n e também para facilitar a correlação entre o método de Elias e Hartman [10] e a abordagem algébrica utilizada nesta tese.

Uma *configuração* de ciclos é um subgrafo de $G(\pi)$ induzido por um ou mais ciclos. Uma configuração A é dita *conectada*, se para quaisquer dois ciclos C_1 e C_m de A, existem ciclos $C_2, \ldots, C_{m-1}$ em A, tal que, para cada $i \in [1, m-1]$, C_i cruza ou intercala com C_{i+1}. Se uma configuração é constituída apenas de ciclos não-orientados, então a chamamos de *configuração não-orientada*. Um *componente* é uma configuração que consiste de apenas

um ciclo orientado que não se cruza ou se intercala com nenhum outro ciclo de $G(\pi)$; ou de uma configuração conectada maximal em $G(\pi)$.

Seja A uma configuração de $G(\pi)$, tal que o número de ciclos de comprimento par em A seja par. A 3-*norma* de A, denotada por $||A||$, é o valor $\frac{b-ci(A)}{2}$, onde b é o número de arestas pretas de A e $ci(A)$ é o número de ciclos de comprimento ímpar em A. Se $||A|| \leq 8$, então A é referida como sendo uma *configuração pequena*; caso contrário, uma *configuração grande*. É importante reforçar que o conceito da 3-norma não foi definido no trabalho de Elias e Hartman [10]. A intuição por trás dela é que a mesma reflete o número de ciclos que uma configuração contendo ciclos de comprimentos arbitrários teria se fosse "simplificada".

Exemplo 5. *A* 3*-norma da configuração* $\{(9, 6, 8, 2, 4, 1, 3, 5, 7)\}$ *de* $G([4\ 3\ 2\ 1\ 8\ 7\ 6\ 5])$ *(Figura 2.1) é* 4 *e, consequentemente, é uma configuração pequena.*

Exemplo 6. *As* 3*-normas das configurações* $\{(7, 4, 1), (8, 5, 2), (9, 6, 3)\}$ *e* $\{(14, 12, 10), (15, 13, 11)\}$ *de* $G([4\ 8\ 3\ 7\ 2\ 6\ 1\ 5\ 9\ 14\ 13\ 12\ 11\ 10])$ *(Figura 3.4) são* 3 *e* 2*, respectivamente.*

Uma *porta aberta* é um par de arestas pretas (a, b) de um ciclo C em A tal que uma de suas formas cíclicas é $C = (a, b, \dots)$, que não cruza com nenhum outro ciclo em A e não há aresta preta c em C, tal que, se $a > b$, então a, b, c não é uma sequência decrescente; ou, se $b > a$, então b, c, a também não é uma sequência decrescente. Uma configuração que não contém portas abertas é chamada de *configuração completa.*

Exemplo 7. *As configurações* $\{(7, 4, 1), (8, 5, 2), (9,\ 6, 3)\}$ *e* $\{(14, 12, 10), (15, 13, 11)\}$ *são componentes completos e pequenos de* $G([4\ 8\ 3\ 7\ 2\ 6\ 1\ 5\ 9\ 14\ 13\ 12\ 11\ 10])$ *(Figura 3.4).*

2.2.4 Sequências de transposições

Uma sequência de transposições $\tau_1, \dots, \tau_x$ é dita uma (x, y)*-sequência*, onde $x \geq y$, é uma sequência de x transposições, tais que, pelo menos y delas sejam 2-movimentos. Uma sequência (x, y) é uma $\frac{a}{b}$*-sequência* se $\frac{x}{y} \leq \frac{a}{b}$ e $x \leq a$.

Exemplo 8. *A sequência* $\tau_1 = \tau(1, 4, 7)$*,* $\tau_2 = \tau(2, 5, 8)$*,* $\tau_3 = \tau(1, 4, 7)$*,* $\tau_4 = \tau(3, 6, 9)$ *é uma* $(4, 3)$*-sequência, que é também uma* $\frac{11}{8}$*-sequência para a permutação* [4 8 3 7 2 6 1 5 9 14 13 12 11 10] *(Figura 3.4).*

2.2.5 O algoritmo de Elias e Hartman

Elias e Hartman [10] utilizaram um programa de computador para enumerar todas as configurações conectadas de 3-ciclos não-orientados, contendo até 9 ciclos, que permitem

a aplicação de $\frac{11}{8}$-sequências. Partindo de um único 3-ciclo, as configurações conectadas são obtidas por meio de *extensões suficientes*, descritas a seguir.

1. Se uma configuração A contém portas abertas, podemos adicionar um 3-ciclo C não-orientado à A de modo a fechar uma porta aberta.

2. Se, por outro lado, A é uma configuração completa, então podemos adicionar um 3-ciclo C não orientado à A de maneira que a extensão resultante tenha no máximo uma porta aberta.

Uma configuração obtida por uma série de extensões suficientes é uma *configuração suficiente*, se for possível aplicar-lhe uma $\frac{11}{8}$-sequência.

Lema 9 (Elias e Hartman [10])**.** *Toda configuração não-orientada suficiente de* 9 *ciclos permite a aplicação de uma* $\frac{11}{8}$*-sequência.*

Elias e Hartman [10] descobriram que, dentre todos componentes pequenos, constituídos apenas de 3-ciclos não-orientados, apenas 5 não permitem a aplicação de $\frac{11}{8}$-sequências, os quais foram chamados de *componentes pequenos ruins*.

Lema 10 (Elias e Hartman [10])**.** *Os componentes pequenos ruins encontrados por Elias e Hartman [10] são:*

1. *o par intercalante não-orientado (Figura 3.6);*

2. *os colares não-orientados de tamanho* 4, 5 *e* 6 *(Figuras 3.7, 3.8 e 3.9); e*

3. *o colar torcido de tamanho* 4 *(Figura 3.10).*

Entretanto, ao investigar configurações consistindo de combinações de componentes pequenos ruins, Elias e Hartman [10] descobriram o importante resultado a seguir.

Lema 11 (Elias e Hartman [10])**.** *Se uma permutação, cujo grafo de ciclos consiste apenas de componentes pequenos ruins, tem pelo menos oito* 3*-ciclos, então essa permutação permite a aplicação de uma* $\frac{11}{8}$*-sequência.*

Juntas, as demonstrações dos resultados acima consistem na análise computadorizada de cerca de 80.000 casos.

A partir dos dois resultados anteriores foi derivado o seguinte corolário:

Corolário 12 (Elias e Hartman [10])**.** *Toda* 3*-permutação com pelo menos oito* 3*-ciclos permite a aplicação de uma* $\frac{11}{8}$*-sequência.*

Estes resultados são a base do algoritmo de Elias e Hartman [10], apresentado a seguir, conforme esboço apresentado em [10].

Algoritmo 1 Algoritmo EH

```
1: procedimento EHSORT(π)
2:     transforme π em uma permutação simples π̂
3:     procure por uma (2, 2)-sequência e a aplique, caso exista
4:     enquanto G(π̂) contém um 2-ciclo faça
5:         aplique um 2-movimento
6:     fim enquanto
7:     marque todos os 3-ciclos de G(π̂)                    ▷ π̂ é uma 3-permutação
8:     enquanto G(π̂) contém um 3-ciclo marcado C faça
9:         se C é orientado então
10:            aplique um 2-move
11:        senão
12:            tente estender C oito vezes
13:            se uma configuração grande foi alcançada então
14:                aplique uma 11/8-sequência
15:            senão                          ▷ foi alcançado um componente pequeno
16:                se existe uma 11/8-sequência então
17:                    aplique uma 11/8-sequência
18:                senão
19:                    desmarque todos os ciclos do componente   ▷ trata-se de um comp. pequeno ruim
20:                fim se
21:            fim se
22:        fim se
23:    fim enquanto
24:    enquanto G(π̂) contém pelo menos 8 ciclos faça
25:        aplique uma 11/8-sequência
26:    fim enquanto
27:    enquanto G(π̂) contém um 3-ciclo faça
28:        aplique uma (3, 2)-sequência
29:    fim enquanto
30:    imite a ordenação de π̂ em π
31: fim procedimento
```

2.3 Grupos de permutações e formalização algébrica do SBT

2.3.1 Grupos de permutações

Os resultados apresentados a seguir são clássicos na literatura de grupos de permutações e suas demonstrações podem ser encontradas em livros básicos de álgebra abstrata [29, 30].

Uma *permutação* de um conjunto E é uma bijeção de E sobre si mesmo. Um *grupo de permutações* de um conjunto E corresponde ao conjunto das permutações de E que formam um grupo sob a operação de composição de funções. Se a cardinalidade de E é n, então chamamos o grupo formado de *grupo simétrico*, denotado S_n. Assim, se α e β são permutações em S_n, então $\alpha \cdot \beta$, ou simplesmente $\alpha\beta$, é a função que mapeia qualquer elemento x de E para $\alpha(\beta(x))$.

Um elemento $x \in E$ é dito ser um elemento *fixo* de $\alpha \in S_n$, se $\alpha(x) = x$. Se existe um subconjunto $\{c_1, c_2, \ldots, c_{\kappa-1}, c_\kappa\}$ de elementos distintos de E, tal que

$$\alpha(c_1) = c_2, \alpha(c_2) = c_3, \ldots, \alpha(c_{\kappa-1}) = c_\kappa, \alpha(c_\kappa) = c_1,$$

e α fixa todos os outros elementos, então nós chamamos α de *ciclo*. Na *notação cíclica*, este ciclo é escrito como $\alpha = (c_1\ c_2 \ldots c_{\kappa-1}\ c_\kappa)$, mas qualquer um dos $(c_2 \ldots c_{\kappa-1}\ c_\kappa\ c_1)$, $\ldots$, $(c_\kappa\ c_1\ c_2 \ldots c_{\kappa-1})$ denota o mesmo ciclo α. O número κ é o *comprimento* de α, denotado por $|\alpha|$. Neste caso, α também é chamado de κ-*ciclo*.

O *suporte* de uma permutação α, denotado $Supp(\alpha)$, é o subconjunto de elementos movidos (não fixos) de E. Duas permutações α e β são ditas *disjuntas*, se $Supp(\alpha) \cap Supp(\beta) = \varnothing$, ou seja, se todo elemento movido por uma permutação está fixo na outra. Sabe-se que, se α e β são disjuntos, então eles comutam como elementos de S_n.

Lema 13. *Cada permutação em S_n pode ser escrita como um produto de ciclos disjuntos. Esta representação, chamada* decomposição cíclica disjunta*, é única, independentemente da ordem em que os ciclos são escritos na representação.*

Por simplicidade, um ciclo β *em* ou *de* uma permutação α é um ciclo na decomposição cíclica disjunta de α.

A permutação identidade ι é a permutação que fixa todos os elementos de E. Elementos fixos às vezes são omitidos na notação cíclicos. No entanto, quando necessário, eles são escritos como 1-ciclos.

Teorema 14. *Toda permutação em S_n pode ser escrita como um produto (não único) de 2-ciclos.*

Uma permutação α é dita ser *par* (*ímpar*) se a mesma pode ser escrita como um produto de um número par (ímpar) de 2-ciclos[2]. A seguir, apresentamos alguns resultados importantes relacionados à paridade de permutações.

Teorema 15. *Se uma permutação α é escrita como um produto de um número par (ímpar) de 2-ciclos, então ela não pode ser escrita como um produto de um número ímpar (par) de 2-ciclos.*

Exemplo 16. *A permutação $\gamma = [4\ 8\ 3\ 7\ 2\ 6\ 1\ 5]$, em notação cíclica, é representada por $(1\ 4\ 7)(2\ 8\ 5)(3)(6)$. Neste caso, 3 e 6 são elementos fixos e poderiam ser omitidos nesta notação. Podemos dizer que γ pode ser escrita, de forma única, como um produto*

[2]Um 2-ciclo é comumente referido como transposição na literatura de álgebra. Para evitar mal-entendidos com a terminologia, nesta tese, "transposição" sempre se refere à troca de dois blocos adjacentes de símbolos em uma permutação.

de dois 3-ciclos disjuntos. Essa permutação poderia ser escrita como produto de outros ciclos também, mas esses ciclos não seriam disjuntos. Além disso, γ *pode ser escrito como* $(1\ 7)(1\ 4)(2\ 5)(2\ 8)$*, usando quatro* 2*-ciclos, e também como* $(1\ 7)(4\ 7)(1\ 7)(4\ 7)(2\ 5)(2\ 8)$*, usando seis* 2*-ciclos.*

Teorema 17. *Se* α*,* $\beta \in S_n$ *são permutações com a mesma paridade, então o produto* $\alpha\beta$ *é par.*

Proposição 18. *Seja* γ *um* κ*-ciclo. Se* κ *é ímpar, então* γ *é uma permutação par, caso contrário,* γ *é uma permutação ímpar.*

Para evitar mal-entendidos com a paridade de ciclos no formalismo do grafo de ciclos, que é oposto ao classicamente usado em grupos de permutação, na abordagem algébrica, sempre nos referiremos aos comprimentos dos ciclos e não à sua paridade.

2.3.2 Formalização algébrica do SBT

Uma permutação $\pi = [\pi_1\ \pi_2 \dots \pi_n]$ pode ser representada de muitas maneiras diferentes. Como dito no capítulo anterior, no contexto de rearranjo de genomas, onde π modela um cromossomo, possivelmente a representação de π mais amplamente utilizada é o grafo de ciclos [9]. Uma representação alternativa ao grafo de ciclos é usando um abordagem inspirada na abordagem algébrica proposta por Mira e outros [2]. Nessa abordagem, a permutação π é representada como o $(n+1)$-ciclo $\bar{\pi} = (0\ \pi_1\ \pi_2 \dots \pi_n)$ e a identidade como $\bar{\iota} = (0\ 1\ 2\ \dots\ n)$[3]. Uma correspondência entre os ciclos do produto $\bar{\iota}\bar{\pi}^{-1}$ (apresentado a seguir) e o conjunto de ciclos de $G(\pi)$ é mostrada no final deste capítulo.

Um 3-ciclo $\tau = (\pi_i\ \pi_j\ \pi_k)$ é *aplicável* em $\bar{\pi}$, se os símbolos π_i, π_j e π_k aparecem em $\bar{\pi}$ na mesma ordem cíclica em que estão em τ, ou seja, $\bar{\pi} = (\pi_i \dots \pi_j \dots \pi_k \dots)$ [2]. A *aplicação* de τ em $\bar{\pi}$ significa multiplicar τ por $\bar{\pi}$. Assim, e somente neste caso, o produto $\tau\bar{\pi}$ é um $(n+1)$-ciclo, tal que os símbolos entre π_i e π_{j-1} são trocados de posição com os símbolos entre π_j e π_{k-1}, simulando dessa forma uma transposição em $\bar{\pi}$. Assim, $\tau\bar{\pi} = (\pi_i\ \pi_j\ \pi_k)(\pi_0\pi_1 \dots \pi_{i-1}\pi_i\pi_{i+1} \dots \pi_{j-1}\pi_j\pi_{j+1} \dots \pi_{k-1}\pi_k \dots \pi_n) = (\pi_0\pi_1 \dots \pi_{i-1}\pi_j\pi_{j+1} \dots \pi_{k-1}\pi_i\pi_{i+1} \dots \pi_{j-1}\pi_k \dots \pi_n)$.

Exemplo 19. *Seja* $\bar{\pi} = (0\ 4\ 3\ 2\ 1\ 8\ 7\ 6\ 5)$*. O* 3*-ciclo* $\tau = (0\ 2\ 7)$ *é aplicável a* $\bar{\pi}$ *e assim simula uma transposição. A aplicação* $\tau\bar{\pi}$ *produz* $(0\ 4\ 3\ 7\ 6\ 5\ 2\ 1\ 8)$*. Agora, considere o* 3*-ciclo* $\tau' = (0\ 1\ 2)$*. Observe que* τ' *não é aplicável a* $\bar{\pi}$*, e o resultado do produto* $\tau'\bar{\pi}$ *é* $(0\ 4\ 3\ 7\ 6\ 5)(1\ 8)(2)$*, o qual não é um* $(n+1)$*-ciclo e, portanto, não representa um cromossomo em nossa abordagem.*

[3]Observe que $\bar{\iota} = (0\ 1\ 2\ \dots\ n)$ não é $\iota = (0)(1)\cdots(n)$.

O problema SBT, no formalismo algébrico, pode ser visto como: dado um $(n+1)$-ciclo $\bar{\pi}$, o problema de encontrar o número mínimo t, denotado $d(\bar{\pi})$, de transposições representadas como 3-ciclos aplicáveis necessários para transformar $\bar{\pi}$ em $\bar{\iota} = (0\ 1\ 2\ \ldots\ n)$, ou seja,

$$\tau_t \ldots \tau_1 \bar{\pi} = \bar{\iota}. \tag{2.1}$$

Da igualdade acima, multiplicando ambos os lados por $\bar{\pi}^{-1}$, temos que

$$\tau_t \ldots \tau_1 = \bar{\iota}\bar{\pi}^{-1}. \tag{2.2}$$

Observe que pela Proposição 18 e Teorema 17, o produto de dois ciclos com o mesmo comprimento é uma permutação par.

Proposição 20. *A permutação $\bar{\iota}\bar{\pi}^{-1}$ é uma permutação par.*

A 3-*norma* [31] de uma permutação par $\alpha \in S_n$, denotada por $||\alpha||_3$, corresponde ao menor ℓ tal que $\beta_\ell \ldots \beta_1 = \alpha$, onde cada β_i, $1 \leq i \leq \ell$, é um 3-ciclo. Denote por $c^\circ{}_{\acute{i}mpar}(\alpha)$, o número de ciclos de comprimento ímpar, incluindo também 1-ciclos, em α, respectivamente. Mira e Meidanis [31] demonstraram o seguinte resultado:

Lema 21 (Mira e Meidanis [31])**.**

$$||\alpha||_3 = \frac{n - c^\circ{}_{\acute{i}mpar}(\alpha)}{2}.$$

Como $\bar{\iota}\bar{\pi}^{-1}$ é uma permutação par (Proposição 20), então, como corolário, um limite inferior para o SBT é derivado a seguir.

Lema 22 (Mira e Meidanis [31])**.** *Se $\bar{\pi}$ é um $(n+1)$-ciclo, então*

$$\begin{aligned} d(\bar{\pi}) &\geq ||\bar{\iota}\bar{\pi}^{-1}||_3 \\ &\geq \frac{n + 1 - c^\circ{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1})}{2}. \end{aligned}$$

2.4 Correspondência entre o grafo de ciclos e a permutação $\bar{\iota}\bar{\pi}^{-1}$

O produto $\bar{\iota}\bar{\pi}^{-1}$, na abordagem algébrica, produz ciclos que correspondem exatamente aos mesmos ciclos de $G(\pi)$. Se seguirmos as arestas dos ciclos em $G(\pi)$, anotando os

rótulos dos vértices onde entram as arestas cinza, desconsiderando o sinal e alterando o rótulo $-(n+1)$ para 0, obtemos exatamente os mesmos ciclos de $\bar{\iota}\bar{\pi}^{-1}$. É fácil ver, portanto, que $\bar{\iota}\bar{\pi}^{-1}$ e $G(\pi)$ têm o mesmo número de ciclos. Além disso, os ciclos de $\bar{\iota}\bar{\pi}^{-1}$ têm as mesmas propriedades relevantes dos ciclos correspondentes de $G(\pi)$, como comprimento e orientação. As relações entre os ciclos, ou seja, cruzamento e intercalação, também são idênticas. Finalmente, configurações e componentes também são conceitos correspondentes em ambas estruturas. Todos estes conceitos, na abordagem algébrica, serão formalizados na Seção 3.2.

Exemplo 23. *Seja* $\pi = [6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10]$ *(*$G(\pi)$ *representado na Figura 2.2). Como visto no Exemplo 4, os ciclos de* $G(\pi)$ *são* $(5,3,1)$*,* $(8,6,4)$*,* $(9,2,7)$*,* $(14,12,10)$ *e* $(15,13,11)$*. Agora, seja* $\bar{\pi} = (0\ 6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10)$*, de modo que* $\bar{\iota}\bar{\pi}^{-1} = (0\ 11\ 13)(1\ 3\ 6)(2\ 4\ 8)(5\ 7\ 9)(10\ 12\ 14)$*. Considere o ciclo* $(5,3,1)$ *de* $G(\pi)$*. Se seguirmos o procedimento acima, obtemos o ciclo de* $\bar{\iota}\bar{\pi}^{-1}$ $(1\ 3\ 6)$*. O mesmo procedimento leva* $(8,6,4)$ *para* $(2\ 4\ 8)$*;* $(9,2,7)$ *para* $(5\ 7\ 9)$ *(observe que esses ciclos são igualmente orientados);* $(14,12,10)$ *para* $(10\ 12\ 14)$*; e finalmente* $(15,13,11)$ *para* $(0\ 11\ 13)$*. Além disso, observe que* $(5,3,1)$ *e* $(8,6,4)$ *são ciclos cruzantes, da mesma forma que* $(1\ 3\ 6)$ *e* $(2\ 4\ 8)$*. Não só, os pares* $(14,12,10)$ *e* $(15,13,11)$*; e* $(10\ 12\ 14)$ *e* $(0\ 11\ 13)$ *são intercalantes. Por fim, os mesmos pares de ciclos formam componentes pequenos em ambas as estruturas.*

Capítulo 3

Novo algoritmo 1.375-aproximativo para o SBT

Neste capítulo, primeiro, na Seção 3.1, apresentaremos a motivação para o desenvolvimento de uma nova solução 1.375-aproximativa para o SBT, que é o fato de o Algoritmo EH poder produzir uma transposição extra, além da razão de aproximação prometida de 1.375, dependendo de como a permutação a ser ordenada é simplificada. A seguir, na Seção 3.2, iremos apresentar conceitos e propriedades dos ciclos de $\overline{\iota}\overline{\pi}^{-1}$, onde a maior parte são versões algébricas dos conceitos e propriedades dos ciclos de $G(\pi)$. Por fim, nas seções 3.3.2 e 3.4, exporemos nossos principais resultados, que são um novo limite superior para o SBT e um novo algoritmo 1.375-aproximativo que garante a razão de aproximação para todo o grupo simétrico S_n, sem transposições extras.

3.1 Motivação

O primeiro passo do Algoritmo EH é a simplificação da permutação de entrada. Nesta seção, mostramos que existem simplificações que, embora produzam permutações simples equivalentes, fazem com que o Algoritmo EH produza uma transposição extra acima da razão de aproximação prometida de 1.375. Dois exemplos são explorados a seguir.

Considere a permutação π = [4 3 2 1 8 7 6 5] mostrada na Figura 2.1. O limite inferior dado pelo Teorema 3 é 4, também sua distância exata, correspondente à aplicação de quatro 2-movimentos, mostrados na Figura 3.1. Uma simplificação possível de π é a permutação [4.1 4 3 2 1 4.2 8.1 8 7 6 5], que mapeada para inteiros consecutivos, resulta em $\hat{\pi}$ = [5 4 3 2 1 6 11 10 9 8 7] (Figura 2.3). Observe que o limite inferior de $\hat{\pi}$ também é 4. No entanto, não há $\frac{11}{8}$-sequência que possa ser aplicada em $\hat{\pi}$. Na verdade, para ordenar $\hat{\pi}$ de maneira ótima, duas $(3, 2)$-sequências são necessárias. Portanto, o Algoritmo EH, usando π = [4 3 2 1 8 7 6 5] como entrada, mesmo aplicando uma ordenação ótima em

$\hat{\pi}$ = [5 4 3 2 1 6 11 10 9 8 7], produz 6 transposições. No entanto, o algoritmo deveria produzir no máximo 5 transposições para não exceder a razão de aproximação 1.375.

$$\tau(4,6,9) \cdot [4\ 3\ 2\ \underline{1\ 8}\ \underline{\underline{7\ 6\ 5}}] = [4\ 3\ 2\ 7\ 6\ 5\ 1\ 8]$$
$$\tau(3,5,8) \cdot [4\ 3\ \underline{2\ 7}\ \underline{\underline{6\ 5\ 1}}\ 8] = [4\ 3\ 6\ 5\ 1\ 2\ 7\ 8]$$
$$\tau(2,4,7) \cdot [4\ \underline{3\ 6}\ \underline{\underline{5\ 1\ 2}}\ 7\ 8] = [4\ 5\ 1\ 2\ 3\ 6\ 7\ 8]$$
$$\tau(1,3,6) \cdot [\underline{4\ 5}\ \underline{\underline{1\ 2\ 3}}\ 6\ 7\ 8] = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$$

Figura 3.1: Ordenação de π = [4 3 2 1 8 7 6 5] usando 4 transposições.

O exemplo a seguir mostra que, mesmo que existam $\frac{11}{8}$-sequências de transposições possíveis de serem aplicadas em $\hat{\pi}$, o Algoritmo EH pode produzir uma transposição acima da razão de aproximação 1.375. Considere a permutação π' = [3 6 2 5 1 4 10 9 8 7] (Figura 3.2), com limite inferior e distância iguais a 5, correspondendo à aplicação de cinco 2-movimentos, mostrados na Figura 3.3. Uma versão simplificada de π' é [3.1 6.1 3 6 2 5 1 4 6.2 10.1 10 9 8 7], que mapeada para números inteiros consecutivos, resulta em $\hat{\pi}'$ = [4 8 3 7 2 6 1 5 9 14 13 12 11 10] (Figura 3.4). O Algoritmo EH ordena $\hat{\pi}'$ de maneira ótima aplicando uma $(4,3)$-sequência, seguida de uma $(3,2)$-sequência, em um total de 7 transposições. No entanto, o algoritmo não deveria produzir mais do que 6 transposições para não exceder a razão de aproximação 1.375.

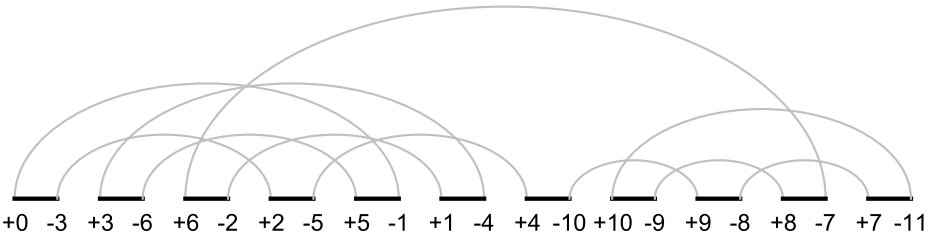

Figura 3.2: G([3 6 2 5 1 4 10 9 8 7]).

$$\tau(6,8,11) \cdot [3\ 6\ 2\ 5\ 1\ \underline{4\ 10}\ \underline{\underline{9\ 8\ 7}}] = [3\ 6\ 2\ 5\ 1\ 9\ 8\ 7\ 4\ 10]$$
$$\tau(5,7,10) \cdot [3\ 6\ 2\ 5\ \underline{1\ 9}\ \underline{\underline{8\ 7\ 4}}\ 10] = [3\ 6\ 2\ 5\ 8\ 7\ 4\ 1\ 9\ 10]$$
$$\tau(3,6,9) \cdot [3\ 6\ \underline{2\ 5\ 8}\ \underline{\underline{7\ 4\ 1}}\ 9\ 10] = [3\ 6\ 7\ 4\ 1\ 2\ 5\ 8\ 9\ 10]$$
$$\tau(2,4,8) \cdot [3\ \underline{6\ 7}\ \underline{\underline{4\ 1\ 2\ 5}}\ 8\ 9\ 10] = [3\ 4\ 1\ 2\ 5\ 6\ 7\ 8\ 9\ 10]$$
$$\tau(1,3,5) \cdot [\underline{3\ 4}\ \underline{\underline{1\ 2}}\ 5\ 6\ 7\ 8\ 9\ 10] = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$$

Figura 3.3: Ordenação de π' = [3 6 2 5 1 4 10 9 8 7] usando 5 transposições.

Em ambos os exemplos acima, uma $(2,2)$-sequência inicial é "perdida" durante o processo de simplificação. Esta sequência é essencial para garantir a razão de aproximação 1.375 quando componentes pequenos ruins permanecem em $G(\hat{\pi})$ após a aplicação de um número qualquer de $\frac{11}{8}$-sequências (Teorema 22 [10]). É importante ressaltar que a

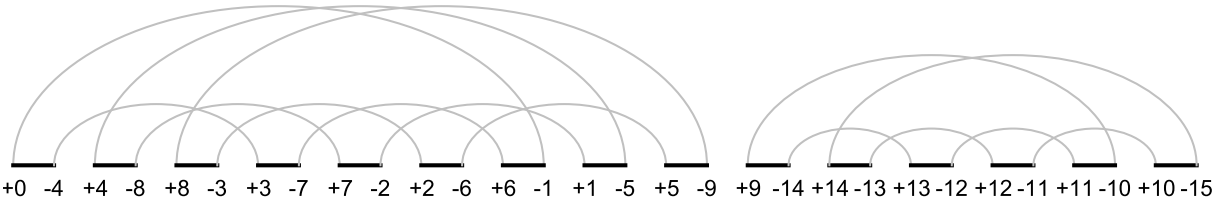

Figura 3.4: $G([4\ 8\ 3\ 7\ 2\ 6\ 1\ 5\ 9\ 14\ 13\ 12\ 11\ 10])$, mapeado da permutação $\hat{\pi}' = [3.1\ 6.1\ 3\ 6\ 2\ 5\ 1\ 4\ 6.2\ 10.1\ 10\ 9\ 8\ 7]$ usando inteiros consecutivos.

transposição extra será gerada independentemente do número de componentes pequenos ruins restantes no grafo de ciclos, após a aplicação de uma série (qualquer número) de $\frac{11}{8}$-sequências, desde que o número total de 3-ciclos remanescentes seja menor que 8 e a sequência inicial $(2, 2)$, que possivelmente existia inicialmente, tenha sido "perdida" durante a simplificação.

Já era sabido da literatura que a simplificação mantinha o limite inferior, mas não a distância de transposição. No entanto, não se sabia que a simplificação poderia ter o efeito de perder uma $(2, 2)$-sequência inicial. Em princípio, o Algoritmo EH poderia ser modificado para garantir a razão de aproximação 1.375, sem nenhuma transposição extra. Para isso, a procura pela $(2, 2)$-sequência deveria ser feita no primeiro passo do algoritmo e aplicada no caso de ser encontrada. Só então a permutação resultante da aplicação poderia ser simplificada. No entanto, usando as técnicas conhecidas na literatura, este novo Algoritmo EH "modificado" não manteria a complexidade de tempo original de $O(n^2)$.

3.2 Os ciclos da permutação $\bar{\iota}\bar{\pi}^{-1}$

Nesta seção, juntamente com alguns novos conceitos, iremos redefinir, em termos algébricos, de forma equivalente, os conceitos e propriedades relacionados aos ciclos do grafo de ciclos.

Seja γ um ciclo em $\bar{\iota}\bar{\pi}^{-1}$. Se $\gamma = (a \dots b \dots c \dots)$ e $\bar{\pi}^{-1} = (a \dots c \dots b \dots)$, ou seja, se os símbolos a, b e c aparecem em γ em uma ordem cíclica distinta daquela em $\bar{\pi}^{-1}$, então dizemos que (a, b, c) é uma *tripla orientada* e γ é um *ciclo orientado*. Caso contrário, se não houver uma tripla orientada em γ, então γ é um *ciclo não-orientado*. Um ciclo $\eta = (\eta_1\ \eta_2 \dots \eta_{|\eta|})$ é um *segmento* de γ se $\gamma = (\eta_1\ \eta_2 \dots \eta_{|\eta|} \dots)$. Observe que, por definição, um ciclo em $\bar{\iota}\bar{\pi}^{-1}$ é um segmento de si mesmo. Analogamente, definimos um segmento de um ciclo γ de $\bar{\iota}\bar{\pi}^{-1}$ como *orientado* ou *não-orientado*.

Sejam $\delta = (a\ b \dots)$ e $\epsilon = (d\ e \dots)$ dois ciclos de $\bar{\iota}\bar{\pi}^{-1}$. Se $\bar{\pi}^{-1} = (a \dots e \dots b \dots d \dots)$, ou seja, se os símbolos dos pares (a, b) e (d, e) ocorrerem em ordem ciclicamente alter-

nada em $\bar{\pi}^{-1}$, dizemos que esses pares são *cruzantes* e que δ e ϵ são ciclos *cruzantes* também. Um caso especial é quando $\delta = (a\ b\ c\dots)$ e $\epsilon = (d\ e\ f\dots)$ são tais que, $\bar{\pi}^{-1} = (a\dots e\dots b\dots f\dots c\dots\ d\dots)$. Ou seja, os símbolos das triplas (a, b, c) e (d, e, f) ocorrem em ordem alternada em $\bar{\pi}^{-1}$. Neste caso, diz-se que δ e ϵ são ciclos *intercalantes*. Analogamente, definimos dois segmentos de dois ciclos $\bar{\iota}\bar{\pi}^{-1}$ como *cruzantes* ou *intercalantes*.

Exemplo 24. *Seja* $\bar{\pi} = (0\ 8\ 7\ 6\ 5\ 1\ 4\ 9\ 3\ 2)$ *e* $\bar{\iota}\bar{\pi}^{-1} = (0\ 3)(1\ 6\ 8)(2\ 4)(5\ 7\ 9)$. *Os ciclos* $(0\ 3)$ *e* $(2\ 4)$ *são exemplos de ciclos cruzantes, enquanto que* $(1\ 6\ 8)$ *e* $(5\ 7\ 9)$ *são ciclos intercalantes.*

Um κ-ciclo em $\bar{\iota}\bar{\pi}^{-1}$ é chamado de *ciclo curto* se $\kappa \leq 3$; caso contrário, é chamado *ciclo longo*. Da mesma forma, um segmento de um ciclo de $\bar{\iota}\bar{\pi}^{-1}$ pode ser *curto* ou *longo*.

Observe que, da Equação 2.2, $\bar{\iota}\bar{\pi}^{-1}{\tau_1}^{-1}\dots{\tau_t}^{-1} = \iota$, ou seja, a aplicação das transposições $\tau_1,\dots,\tau_t$ que ordenam $\bar{\pi}$ (ou seja, que transformam $\bar{\pi}$ em $\bar{\iota}$) pode ser vista como a multiplicação incremental de $\bar{\iota}\bar{\pi}^{-1}$ por ${\tau_1}^{-1}$, …, ${\tau_t}^{-1}$.

Denote por $\Delta c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1}, \tau)$, a diferença $c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1}\ \tau^{-1}) - c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1})$.

Proposição 25 (Meidanis, Dias e Mira [23, 31])**.** *Se* τ *é um* 3*-ciclo aplicável, então* $\Delta c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1}, \tau) \in \{-2, 0, 2\}$.

O número máximo de ciclos em $\bar{\iota}\bar{\pi}^{-1}$ é obtido se e somente se $\bar{\iota}\bar{\pi}^{-1}$ for a permutação identidade ι. Neste caso, ι tem ciclos $n + 1$, sendo todos de comprimento ímpar (em particular, todos têm comprimento 1).

Denotamos por μ*-movimento* um 3-ciclo τ aplicável, tal que $\Delta c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1}, \tau) = \mu$. De acordo com a Proposição 25, os movimentos possíveis são (-2)-movimento, 0-movimento e 2-movimento.

3.2.1 Configurações e componentes

Uma *configuração* Γ é um produto disjunto de segmentos de ciclos de $\bar{\iota}\bar{\pi}^{-1}$, tal que não há dois segmentos em Γ do mesmo ciclo de $\bar{\iota}\bar{\pi}^{-1}$. Se $||\Gamma||_3 \leq 8$, então Γ é uma *configuração pequena*; caso contrário, uma *configuração grande*.

Exemplo 26. *Seja* $\bar{\pi} = (0\ 6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10)$, *portanto* $\bar{\iota}\bar{\pi}^{-1} = (0\ 11\ 13)(1\ 3\ 6)(2\ 4\ 8)(5\ 7\ 9)(10\ 12\ 14)$. *O produto* $(1\ 3\ 6)(2\ 4\ 8)$ *é uma configuração pequena* $\bar{\iota}\bar{\pi}^{-1}$.

Uma configuração Γ é *conectada* se para quaisquer dois segmentos γ_1 e γ_m de Γ, existem segmentos $\gamma_2, \dots, \gamma_{m-1}$ em Γ, tal que para cada $i \in [1, m-1]$, γ_i cruza ou intercala com γ_{i+1}. Γ é dito ser um *componente* se ele consiste de apenas um ciclo orientado que não se

cruza ou se intercala com qualquer outro ciclo de $\bar{\iota}\bar{\pi}^{-1}$; ou consiste de uma configuração conectada maximal de $\bar{\iota}\bar{\pi}^{-1}$.

Exemplo 27. *Seja* $\bar{\pi} = (0\ 6\ 5\ 3\ 2\ 1\ 8\ 7\ 4\ 9\ 14\ 13\ 12\ 11\ 10)$. *Como* $\bar{\iota}\bar{\pi}^{-1} = (0\ 11\ 13)(1\ 3\ 6)(2\ 4\ 8)(5\ 7\ 9)(10\ 12\ \ 14)$, *então* $(0\ 11\ 13)(10\ 12\ 14)$ *e* $(1\ 3\ 6)(2\ 4\ 8)(5\ 7\ 9)$ *são ambos componentes de* $\bar{\iota}\bar{\pi}^{-1}$.

Seja $(a\ b\ c)(d\ e\ f)$ uma configuração de $\bar{\iota}\bar{\pi}^{-1}$ que consiste de dois segmentos cruzantes. Se $\bar{\pi}^{-1} = (a \dots e \dots b \dots f \dots c \dots d \dots)$, ou seja, se $(a\ b\ c)$ e $(d\ e\ f)$ intercalam-se, então o chamamos de *par intercalante não-orientado*. Por outro lado, se $\bar{\pi}^{-1} = (a \dots f \dots b \dots c \dots d \dots e \dots)$, ou seja, $(a\ b\ c)$ e $(d\ e\ f)$ apenas se cruzam, mas não se intercalam, então o chamamos de *par cruzante não-orientado*.

Seja $\epsilon = (a\ b \dots)$ um segmento de uma configuração Γ. Chamamos o par (a, b) de *porta aberta* em Γ, se não houver ciclo $(c\ d \dots)$ em Γ, tal que (a, b) e (c, d) cruzam-se; e não existe $e \in Supp(\epsilon)$, tal que (a, b, e) seja uma tripla orientada. Se Γ é uma configuração que não contém portas abertas, então é uma *configuração completa*. Observe que o par intercalante não-orientado não possui portas abertas e, portanto, é uma configuração completa. O par cruzante não-orientado, por sua vez, tem duas portas abertas.

3.2.2 Sequências de 3-ciclos aplicáveis

Denotamos por (x, y)-*sequência*, para $x \geq y$, uma sequência de x 3-ciclos aplicáveis τ_1, $\dots$, τ_x, tal que, pelo menos y deles são 2-movimentos. Diz-se que uma sequência (x, y) é uma $\frac{a}{b}$-*sequência* se $x \leq a$ e $\frac{x}{y} \leq \frac{a}{b}$.

Exemplo 28. *Seja* $\bar{\pi} = (0\ 4\ 8\ 3\ 7\ 2\ 6\ 1\ 5\ 9\ 14\ 13\ 12\ 11\ 10)$, *portanto* $\bar{\iota}\bar{\pi}^{-1} = (0\ 11\ 13)(1\ 7\ 4)(2\ 8\ 5)(3\ 9\ 6)(10\ 12\ 14)$. *A sequência* $\tau_1 = (1\ 4\ 7)$, $\tau_2 = (2\ 8\ 5)$, $\tau_3 = (1\ 4\ 7)$, $\tau_4 = (3\ 9\ 6)$ *é uma* $(4, 3)$-*sequência que, por sua vez, também é uma sequência* $\frac{11}{8}$.

Dizemos que uma configuração Γ *permite* a aplicação de uma $\frac{a}{b}$-sequência, se for possível escrever esta sequência usando os símbolos de $Supp(\Gamma)$.

3.3 Novo limite superior

3.3.1 Resultados auxiliares

Nesta seção, serão mostrados resultados que irão auxiliar a demonstração de um novo limite superior para o SBT, na Seção 3.3.2.

Corolário 29. *Se existir um* 3*-ciclo orientado* $\gamma = (a\ b\ c)$ *em* $\bar{\iota}\bar{\pi}^{-1}$*, então* $(a\ b\ c)$ *é um* 2*-movimento.*

Proposição 30. *Se existe um ciclo de comprimento par em* $\bar{\iota}\bar{\pi}^{-1}$*, então existe um* 2*-movimento.*

Demonstração. Como $\bar{\iota}\bar{\pi}^{-1}$ é uma permutação par (Proposição 20), então existe um número par de ciclos de comprimento par em $\bar{\iota}\bar{\pi}^{-1}$. Sejam $\gamma = (a\ b \dots)$ e $\delta = (c\ d \dots)$ dois ciclos de comprimento par de $\bar{\iota}\bar{\pi}^{-1}$. Temos dois casos:

1. γ e δ cruzam-se. Neste caso, temos que $\bar{\pi}^{-1} = (a \dots d \dots b \dots c \dots)$. Então $(a\ b\ c)$ é um 2-movimento.

2. γ e δ não se cruzam. Sem perda de generalidade, suponha que $\bar{\pi}^{-1} = (a \dots b \dots c \dots d \dots)$. Neste caso, $(a\ c\ b)$ é um 2-movimento.

□

Lema 31. *Se existe um* 5*-ciclo* $\gamma = (a\ d\ b\ e\ c)$ *em* $\bar{\iota}\bar{\pi}^{-1}$*, tal que* (a, b, c) *é uma tripla orientada, então existe um* 2*-movimento ou uma* $(3, 2)$*-sequência.*

Demonstração. As possíveis formas distintas de $\bar{\pi}$ em relação às posições dos símbolos de $Supp(\gamma)$ estão listadas abaixo. Para cada uma, há um 2-movimento ou uma $(3, 2)$-sequência.

1. $\bar{\pi} = (a \dots b \dots c \dots d \dots e \dots)$. $\tau_1 = (a\ b\ c)$, $\tau_2 = (b\ c\ d)$, $\tau_3 = (c\ d\ e)$.
2. $\bar{\pi} = (a \dots b \dots c \dots e \dots d \dots)$. $\tau_1 = (b\ e\ d)$.
3. $\bar{\pi} = (a \dots b \dots e \dots c \dots d \dots)$. $\tau_1 = (a\ e\ c)$.
4. $\bar{\pi} = (a \dots e \dots b \dots d \dots c \dots)$. $\tau_1 = (a\ d\ c)$.
5. $\bar{\pi} = (a \dots b \dots e \dots d \dots c \dots)$. $\tau_1 = (a\ d\ c)$.
6. $\bar{\pi} = (a \dots d \dots b \dots e \dots c \dots)$. $\tau_1 = (a\ d\ b)$.

□

Observe que, pelo Lema 31, se $\gamma = (a\ d\ b\ e\ c)$ é um 5-ciclo orientado em $\bar{\iota}\bar{\pi}^{-1}$ tal que (a, b, c) é uma tripla orientada, então $\bar{\pi} = (a \dots b \dots c \dots d \dots e \dots)$ é a única forma de $\bar{\pi}$, relativamente às posições dos símbolos de $Supp(\gamma)$, para a qual não há um 2-movimento. Neste caso, chamamos γ de 5*-ciclo orientado ruim.*

Lema 32. *Se há um* κ*-ciclo* $\gamma = (a \dots b \dots c \dots)$ *de comprimento ímpar em* $\bar{\iota}\bar{\pi}^{-1}$*, tal que* $\kappa \geq 7$ *e* (a, b, c) *é uma tripla orientada, então existe um* 2*-movimento ou uma* $(4, 3)$*-sequência.*

Demonstração. Se $(a\ b\ c)$ for um 2-movimento, então o lema vale. Existe apenas um caso em que $(a\ b\ c)$ não seria um 2-movimento. Sem perda de generalidade, suponha que este caso seja

$$\gamma = (\underbrace{d\ e \dots b}_{\text{ímpar}} \mid \underbrace{f \dots c}_{\text{par}} \mid \underbrace{g \dots a}_{\text{par}} \mid).$$

Barras verticais são usadas para indicar os locais onde γ seria "quebrado" se $(a\ b\ c)$ fosse aplicado em $\bar{\pi}$ e subscritos para indicar a paridade do comprimento dos ciclos resultantes. Porém, observe que o ciclo γ pode ser reescrito como o produto

$$\gamma = (\underbrace{a \dots}_{\text{ímpar}})(\underbrace{b \dots}_{\text{ímpar}})(\underbrace{c \dots}_{\text{ímpar}})(a\ d\ e\ b\ f\ c\ g).$$

Existe apenas uma forma de $\bar{\pi}$, relativamente aos símbolos do suporte de $(a\ d\ e\ b\ f\ c\ g)$, que não permite a aplicação de um 2-movimento, que é $\bar{\pi} = (a \dots e \dots f \dots g \dots d \dots b \dots c \dots)$. Para este $\bar{\pi}$, $\tau_1 = (a\ e\ f)$, $\tau_2 = (d\ e\ f)$, $\tau_3 = (b\ f\ d)$, $\tau_4 = (a\ c\ g)$ é uma $(4, 3)$-sequência de transposições. □

Lema 33. *Se $\bar{\iota}\bar{\pi}^{-1} \neq \iota$, existe um 2-movimento ou uma $(3, 2)$-sequência.*

Demonstração. Se houver um ciclo de comprimento par em $\bar{\iota}\bar{\pi}^{-1}$, então pela Proposição 30, existe um 2-movimento. Assim, assumimos que $\bar{\iota}\bar{\pi}^{-1}$ contenha apenas ciclos de comprimento ímpar.

1. Existe um κ-ciclo orientado γ em $\bar{\iota}\bar{\pi}^{-1}$. Se $\kappa = 3$, então o Corolário 29 dá um 2-movimento e o lema vale. Se $\kappa = 5$, então o Lema 31 dá um 2-movimento ou γ é o 5-ciclo orientado ruim. No último caso, existe uma $(3, 2)$-sequência. Por outro lado, se $\kappa \geq 7$, então um 2-movimento ou uma $(4, 3)$-sequência, que contém uma $(3, 2)$-sequência, é dada pelo Lema 32.

2. Todos os ciclos de $\bar{\iota}\bar{\pi}^{-1}$ são não orientados. Seja $\gamma = (a\ b\ c)$ um segmento de um ciclo de $\bar{\iota}\bar{\pi}^{-1}$. Temos dois casos:

   (a) γ intercala-se com um outro segmento $\delta = (d\ e\ f)$. Neste caso, temos que $\bar{\pi} = (a \dots f \dots c \dots e \dots b \dots d \dots)$. Então, $\tau_1 = (a\ c\ b)$, $\tau_2 = (d\ e\ f)$ e $\tau_3 = (a\ c\ b)$ é uma $(3, 2)$-sequência.

   (b) γ cruza com dois segmentos $\delta = (d\ e\ f)$ e $\epsilon = (g\ h\ i)$. Para cada uma das 15 formas distintas de $\bar{\pi}$ (enumeradas em [32]), relativamente às posições possíveis dos símbolos de γ, δ e ϵ, existe uma $(3, 2)$-sequência.

□

3.3.2 Análise das configurações

As provas de alguns dos resultados desta seção dependem da análise de um grande número de casos. Como é inviável enumerar e verificar manualmente todos os casos, implementamos, como Elias e Hartman [10], alguns programas de computador [33] para gerar sistematicamente as provas. Para facilitar a visualização e compreensão geral, as provas estão publicamente disponíveis na forma de uma interface web amigável [32].

Neste ponto, consideramos $\bar{\iota}\bar{\pi}^{-1}$ consistindo apenas de ciclos não orientados de qualquer comprimento ímpar, ou 5-ciclos orientados ruins. Para os outros casos, Corolário 29, Proposição 30 e Lema 32 dão um 2-movimento ou uma $(4, 3)$-sequência.

Nosso objetivo é provar que, se $||\bar{\iota}\bar{\pi}^{-1}||_3 \geq 8$, então existe uma $\frac{11}{8}$-sequência de transposições. Tal qual em Elias e Hartman [10], a análise é dividida em duas partes. Na primeira parte, analisamos as configurações obtidas a partir das configurações básicas (definidas abaixo), por extensão. Na segunda parte, analisamos $\bar{\iota}\bar{\pi}^{-1}$ composto apenas por componentes pequenos que não permitem a aplicação de $\frac{11}{8}$-sequências.

Extensão de configurações básicas

A análise começa com o 5-ciclo orientado ruim e as duas únicas configurações conectadas de 3-norma igual a 2: o par cruzante não-orientado; e o par intercalante não-orientado. A partir dessas três *configurações básicas*, é possível construir qualquer outra configuração conectada de $\bar{\iota}\bar{\pi}^{-1}$ por extensões sucessivas. De uma configuração Γ, podemos obter uma configuração maior Γ', tal que $||\Gamma'||_3 = ||\Gamma||_3 + 1$, estendendo Γ por meio de três *extensões suficientes* diferentes, definidas abaixo.

1. Se Γ tiver portas abertas, podemos adicionar um novo segmento não-orientado de comprimento 3 a Γ, fechando pelo menos uma porta aberta.

2. Se Γ não tiver portas abertas, podemos adicionar um novo segmento não-orientado de comprimento 3 a Γ, de modo que este segmento cruze ou intercale outro em Γ.

3. Seja γ um segmento em Γ. Podemos aumentar o comprimento de γ em 2, originando um 5-ciclo orientado ruim; ou um segmento não-orientado mais longo, de modo que pelo menos uma porta aberta seja fechada, se Γ tiver portas abertas; ou criando até duas portas abertas, caso contrário.

Exemplo 34. *Podemos estender a configuração* Γ *de Exemplo 26 usando a extensão 1, gerando* $\Gamma' = (1\ 8\ 10)(5\ 7\ 12)(9\ 11\ 13)$. *Então, com a extensão 2, obtemos* $\Gamma'' = (1\ 8\ 10)(2\ 4\ 6)(5\ 7\ 12)(9\ 11\ 13)$. *Finalmente, com a extensão 3, obtemos* $\Gamma''' = (0\ 3\ 5\ 7\ 12)(1\ 8\ 10)(2\ 4\ 6)(9\ 11\ 13)$.

Uma *configuração suficiente* é uma configuração obtida estendendo-se sucessivamente uma das configurações básicas referidas acima. A análise computadorizada prova o seguinte resultado.

Lema 35. *Se for possível construir uma configuração suficiente* Γ *de* $\bar{\iota}\bar{\pi}^{-1}$ *tal que* Γ *seja grande, então* Γ *permite uma* $\frac{11}{8}$*-sequência.*

Observe que nossa definição de extensão de configurações é semelhante à desenvolvida por Elias e Hartman [10]. No entanto, Elias e Hartman [10] só lidaram com extensões de configurações consistindo de 3-ciclos não-orientados, enquanto que nossa definição permite a extensão e geração de configurações contendo segmentos longos.

O Lema 35 poderia ser provado gerando todas as possíveis configurações grandes de 3-norma igual a 9, estendendo as três configurações básicas, e então, para cada uma, procurar por um $\frac{11}{8}$-sequência. No entanto, isso consumiria muito tempo. Em vez disso, nosso programa de computador [33] emprega uma abordagem de busca em profundidade, na qual, a partir das configurações básicas, se conseguirmos encontrar uma $\frac{11}{8}$-sequência para uma configuração suficiente, então não a estendemos mais (Algoritmo 2). O Algoritmo 2 (cuja implementação encontra-se em [33]), aplicado a cada uma das configurações básicas, produz 348.318 arquivos HTML, que juntos, são a prova do Lema 35. O Algoritmo 3 pode ser utilizado para verificar que os 348.318 referidos acima compreendem a totalidade dos casos que deveriam ser analisados, i.e., que nenhum caso escapou da análise, e que todas as $\frac{11}{8}$-sequências encontradas estão corretas.

Algoritmo 2 Extensão de uma configuração básica

```
 1: procedimento ordeneOuEstendaCB(Γ)                ▷ Γ é uma configuração básica
 2:     se Γ permite uma 11/8-sequência então
 3:         renderize a 11/8-sequência em um arquivo HTML
 4:     senão
 5:         para cada extensão suficiente Γ′ do tipo 1 de Γ faça
 6:             ordeneOuEstendaCB(Γ′)
 7:         fim para
 8:         para cada extensão suficiente Γ″ do tipo 2 de Γ faça
 9:             ordeneOuEstendaCB(Γ″)
10:         fim para
11:         para cada extensão suficiente Γ‴ do tipo 3 de Γ faça
12:             ordeneOuEstendaCB(Γ‴)
13:         fim para
14:     fim se
15: fim procedimento
```

Análise de configurações completas pequenas que não permitem $\frac{11}{8}$-sequências

Para concluir a análise, agora tratamos das configurações completas pequenas para as quais o programa [33] não encontrou $\frac{11}{8}$-sequências, e que podem ocorrer como compo-

Algoritmo 3 Verificação das extensões das configurações básicas

```
1: procedimento VERIFIQUECB(Γ)                          ▷ Γ é uma configuração básica
2:     se existe um arquivo contendo uma 11/8-sequência aplicável à Γ então
3:         se a 11/8-sequência está correta então
4:             retorne
5:         senão
6:             retorne um erro
7:         fim se
8:     senão se não existe um arquivo listando todas as extensões possíveis de Γ então
9:         retorne um erro
10:    fim se
11:    para cada extensão suficiente Γ′ do tipo 1 de Γ faça
12:        VERIFIQUECB(Γ′)
13:    fim para
14:    para cada extensão suficiente Γ″ do tipo 2 de Γ faça
15:        VERIFIQUECB(Γ″)
16:    fim para
17:    para cada extensão suficiente Γ‴ do tipo 3 de Γ faça
18:        VERIFIQUECB(Γ‴)
19:    fim para
20: fim procedimento
```

nentes pequenos em $\bar{\iota}\bar{\pi}^{-1}$. Componentes pequenos que não permitem $\frac{11}{8}$-sequências são chamados de *componentes pequenos ruins*.

Lema 36. *Os componentes pequenos ruins são os seguintes:*

1. *o* 5*-ciclo orientado ruim (Figura 3.5);*
2. *o par intercalante não-orientado (Figura 3.6);*
3. *os colares não-orientados de tamanho* 4*,* 5 *e* 6 *(Figuras 3.7, 3.8 e 3.9); e*
4. *o colar torcido de tamanho* 4 *(Figura 3.10).*

Para ilustrar configurações e componentes nas próximas figuras, é mais intuitivo colorir os ciclos com cores diferentes, para facilitar a visualização de cruzamentos e intercalações, e re-rotular as arestas pretas (as horizontais), de forma que, para a aresta preta i, $1 \leq i \leq n+1$, o novo rótulo é $i-1$.

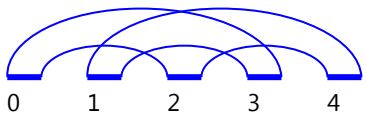

Figura 3.5: O 5-ciclo orientado ruim.

Com exceção do 5-ciclo orientado ruim, os componentes pequenos ruins listados acima são os mesmos encontrados por Elias e Hartman [10], apesar da geração de configurações contendo segmentos longos em nossa análise.

Com a ajuda do programa de computador [33], provamos o seguinte resultado:

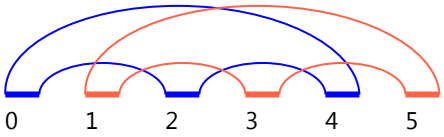

Figura 3.6: O par intercalante não-orientado.

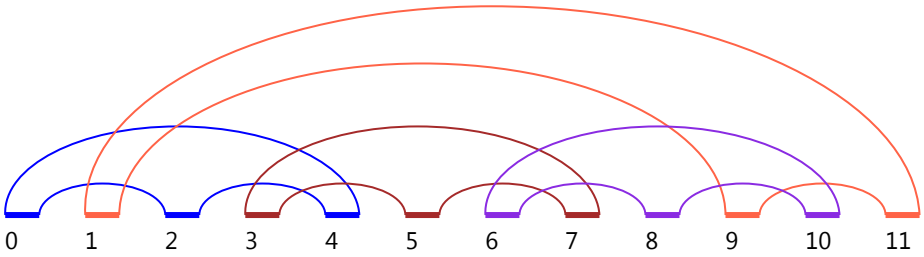

Figura 3.7: O colar não-orientado de tamanho 4.

Lema 37. *Se existir uma configuração* Λ *de* $\bar{\iota}\bar{\pi}^{-1}$ *consistindo apenas de componentes pequenos ruins, tal que* $||\Lambda||_3 \geq 8$*, então* Λ *permite um* $\frac{11}{8}$*-sequência.*

Para provar o Lema 37, a partir de cada um dos componentes ruins listados acima, nós os estendemos sucessivamente adicionando outro componente pequeno ruim à configuração, até encontrar uma $\frac{11}{8}$-sequência (Algoritmo 4). Acontece que nenhuma combinação de componentes pequenos ruins com 3-norma maior que 7 foi estendida durante a análise.

O Algoritmo 4 (cuja implementação encontra-se em [33]), aplicado a cada um dos componentes pequenos ruins, produz 778 arquivos HTML, que juntos, são a prova do Lema 37. De forma similar, o Algoritmo 5 pode ser utilizado para verificar que não existe nenhum caso que não tenha sido analisado e que todas as $\frac{11}{8}$-sequências encontradas estão corretas.

Algoritmo 4 Extensão de um componente pequeno ruim

```
1: procedimento ordeneOuEstendaCPR(Γ)                ▷ Γ é um componente pequeno ruim
2:     se Γ permite uma 11/8-sequência então
3:         renderize a 11/8-sequência em um arquivo HTML
4:     senão
5:         para cada componente pequeno ruim Λ existente faça
6:             para cada símbolo s de π̄ faça
7:                 crie Γ′ adicionando Λ a Γ de forma que os símbolos de Λ, em π̄, venham antes s e
   nenhum ciclo de Λ cruza ou intercala com nenhum outro ciclo de Γ
8:                 ordeneOuEstendaCPR(Γ′)
9:             fim para
10:        fim para
11:    fim se
12: fim procedimento
```

Os resultados apresentados acima permitem provar o corolário abaixo. Ele segue da Proposição 30; parte 1 do Lema 33, o qual implica que se tivermos um ciclo orientado de

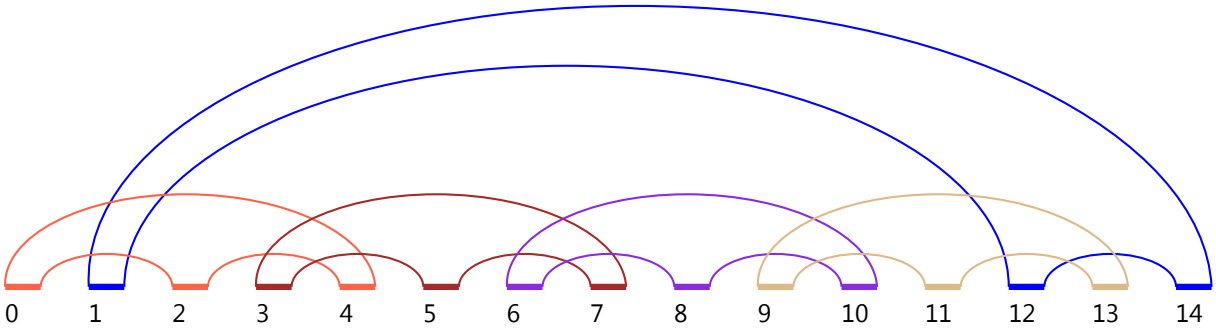

Figura 3.8: O colar não-orientado de tamanho 5.

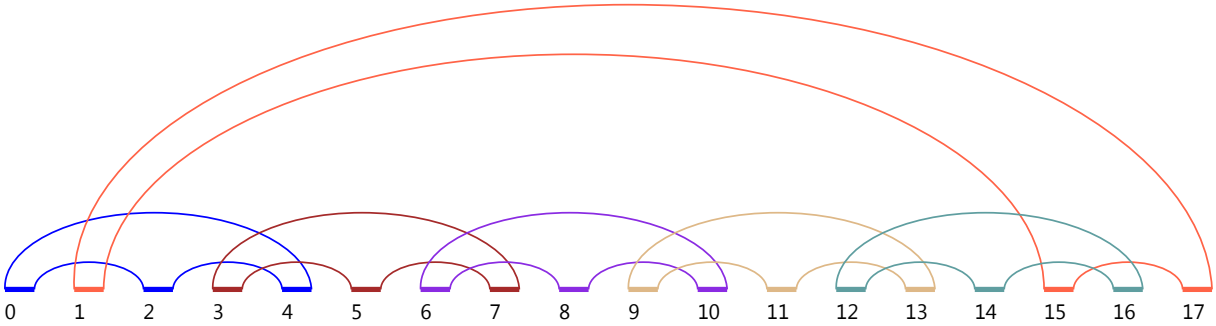

Figura 3.9: O colar não-orientado de tamanho 6.

Algoritmo 5 Verificação das extensões de um componente pequeno ruim

```
1: procedimento VERIFIQUECPR(Γ)                                  ▷ Γ é uma configuração básica
2:     se existe um arquivo contendo uma 11/8-sequência aplicável à Γ então
3:         se se a 11/8-sequência está correta então
4:             retorne
5:         senão
6:             lance um erro
7:         fim se
8:     senão se não existe um arquivo listando todas as extensões possíveis de Γ então
9:         lance um erro
10:    fim se
11:    para cada componente pequeno ruim Λ existente faça
12:        para cada símbolo s de π̄ faça
13:            crie Γ′ adicionando Λ a Γ de forma que os símbolos de Λ, em π̄, venham antes s e nenhum
       ciclo de Λ cruza ou intercala com nenhum outro ciclo de Γ
14:            VERIFIQUECPR(Γ′)
15:        fim para
16:    fim para
17: fim procedimento
```

comprimento ímpar em $\bar{\iota}\bar{\pi}^{-1}$, então temos um 2-movimento, uma $(4,3)$-sequência, ou este ciclo é o 5-ciclo orientado ruim; e Lemas 35 e 37.

Corolário 38. *Se* $||\bar{\iota}\bar{\pi}^{-1}||_3 \geq 8$*, então existe uma* $\frac{11}{8}$*-sequência.*

Por outro lado, se $||\bar{\iota}\bar{\pi}^{-1}||_3 < 8$, garantimos apenas a existência de $\frac{3}{2}$-sequências. Na próxima seção, mostramos que mesmo nesse cenário, a razão de aproximação obtida pelo

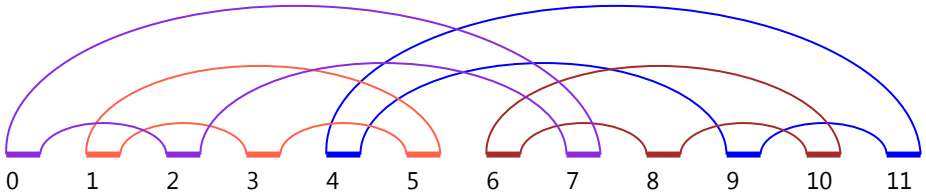

Figura 3.10: O colar torcido de tamanho 4.

nosso algoritmo é no máximo 1.375.

Por fim, os últimos resultados provam o seguinte limite superior para o SBT:

Teorema 39.

$$\begin{aligned} d(\bar{\pi}) &\leq 11\left\lfloor \frac{||\bar{\iota}\bar{\pi}^{-1}||_3}{8} \right\rfloor + \left\lfloor \frac{3(||\bar{\iota}\bar{\pi}^{-1}||_3 \bmod 8)}{2} \right\rfloor \\ &\leq 11\left\lfloor \frac{n+1-c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1})}{16} \right\rfloor + \left\lfloor \frac{3((n+1-c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1})) \bmod 16)}{4} \right\rfloor. \end{aligned}$$

Como $c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1}) = c_{\acute{i}mpar}(\pi)$, o resultado acima pode ser reformulado substituindo $\bar{\pi}$ e $c^{\circ}{}_{\acute{i}mpar}(\bar{\iota}\bar{\pi}^{-1})$, por π e $c_{\acute{i}mpar}(\pi)$, respectivamente. Assim, derivamos o seguinte limite superior para SBT, dependendo apenas de n e $c_{\acute{i}mpar}(\pi)$.

Teorema 40.

$$d(\pi) \leq 11\left\lfloor \frac{n+1-c_{\acute{i}mpar}(\pi)}{16} \right\rfloor + \left\lfloor \frac{3((n+1-c_{\acute{i}mpar}(\pi)) \bmod 16)}{4} \right\rfloor.$$

O novo limite superior acima melhora o limite superior para a distância de transposição encontrado por Bafna e Pevzner [9] em 1998, válido para todo o S_n, com base em seu algoritmo 1.5-aproximativo [27]. Este limite permite obter o seguinte limite superior no diâmetro de transposição (TD):

Corolário 41. $TD(n) \leq 11\left\lfloor \frac{n}{16} \right\rfloor + \left\lfloor \frac{3(n \bmod 16)}{4} \right\rfloor$.

O limite superior para o diâmetro de transposição acima, embora mais "justo", para $n \geq 16$, do que o encontrado por Bafna e Pevzner [9] de $\left\lfloor \frac{3}{4}n \right\rfloor$, não é mais "justo" do que o encontrado por Erikson e outros [15] de $\left\lfloor \frac{2n-2}{3} \right\rfloor$, para $n \geq 9$.

3.4 Novo algoritmo 1.375-aproximativo

Nesta seção, apresentamos um novo algoritmo 1.375-aproximativo de para o SBT (Algoritmo 6). Para uma permutação $\pi \in S_n$, o algoritmo retorna uma distância aproximada entre $\bar{\pi}$ e $\bar{\iota}$ ou, equivalentemente, entre π e ι. Intuitivamente, enquanto $||\bar{\iota}\bar{\pi}^{-1}||_3 \geq 8$, ele aplica repetidamente $\frac{11}{8}$-sequências de transposições em $\bar{\pi}$. Quando $||\bar{\iota}\bar{\pi}^{-1}||_3 < 8$, o algoritmo garante apenas a aplicação de $\frac{3}{2}$-sequências.

Algoritmo 6 Um novo algoritmo 1.375-aproximativo

função SBT1375($\bar{\pi}$)
  $d \leftarrow 0$
  se existe uma $(2,2)$-sequência **então**
    aplique uma $(2,2)$-sequência
    $d \leftarrow d + 2$
  fim se
  enquanto existe um ciclo de comprimento par em $\bar{\iota}\bar{\pi}^{-1}$ **faça**
    aplique um 2-movimento ▷ Proposição 30
    $d \leftarrow d + 1$
  fim enquanto
  Seja Θ o produto de ciclos não marcados de $\bar{\iota}\bar{\pi}^{-1}$
  enquanto $\Theta \neq \bar{\iota}$ **faça**
    se existe um ciclo orientado em Θ que permite um 2-movimento **então**
      aplique um 2-movimento
      $d \leftarrow d + 1$
    senão se existe um κ-ciclo de comprimento ímpar em Θ tal que $\kappa \geq 7$ **então**
      aplique uma $(4,3)$-sequência ▷ Lema 32
      $d \leftarrow d + 4$
    senão
      pegue um segmento γ de comprimento 3 de um ciclo de Θ
      $\Gamma \leftarrow \gamma$
      tente estender Γ oito vezes
      se Γ é grande **então**
        aplique uma $\frac{11}{8}$-sequência de x 3-ciclos ▷ Lema 35
        $d \leftarrow d + x$
      senão se Γ permite uma $\frac{11}{8}$-sequência of x 3-ciclos **então**
        aplique uma $\frac{11}{8}$-sequência de x 3-ciclos
        $d \leftarrow d + x$
      senão
        marque os ciclos de Γ ▷ Γ é uma componente pequeno ruim de $\bar{\iota}\bar{\pi}^{-1}$
      fim se
    fim se
    seja Λ o produto de ciclos marcados de $\bar{\iota}\bar{\pi}^{-1}$
    se $||\Lambda||_3 \geq 8$ **então**
      desmarque os ciclos de Λ
      aplique uma $\frac{11}{8}$-sequência de x 3-ciclos ▷ Lema 37
      $d \leftarrow d + x$
    fim se
  fim enquanto
  enquanto $\bar{\iota}\bar{\pi}^{-1} \neq \bar{\iota}$ **faça**
    aplique uma $\frac{3}{2}$-sequência com x 3-ciclos ▷ Lema 33 (pode ser um 2-movimento, ou seja, uma $(1,1)$-sequência, ou uma $(3,2)$-sequência)
    $d \leftarrow d + x$
  fim enquanto
  retorne d
fim função

Para alcançar a razão de aproximação pretendida de 1.375, mesmo quando $||\bar{\iota}\bar{\pi}^{-1}||_3 < 8$, o algoritmo deve buscar uma $(2,2)$-sequência em seu primeiro passo. Para identificar tal sequência, utiliza-se uma abordagem "olhar-à-frente", ou seja, o algoritmo verifica a existência um segundo 2-movimento, após aplicar um primeiro 2-movimento, gerado a

partir de um ciclo orientado ou de dois ciclos de comprimento par de $\bar{\iota}\bar{\pi}^{-1}$.

Teorema 42. *A complexidade de tempo do Algoritmo 6 é* $O(n^6)$.

Demonstração. A complexidade de tempo de $O(n^6)$ é determinada pela busca de uma $(2,2)$-sequência. Para não perder um 2-movimento, todos as triplas de símbolos de um ciclo orientado devem ser verificados para detectar uma tripla orientada levando a um 2-movimento, que exige tempo $O(n^3)$. Encontrar um 2-movimento combinando três símbolos de dois ciclos de $\bar{\iota}\bar{\pi}^{-1}$ de comprimento par requer $O(n^2)$. Assim, buscar uma $(2,2)$-sequência com a técnica de "olhar-à-frente", para verificar se há um 2-movimento extra, precisa de tempo $O(n^6)$.

O maior laço do algoritmo (linha 12) precisa de tempo $O(n^4)$, enquanto o último laço é $O(n)$. □

Teorema 43. *O Algoritmo 6 é um algoritmo* 1.375*-aproximativo para o SBT.*

Demonstração. É importante ressaltar que esta prova segue uma abordagem muito semelhante à utilizada por Elias e Hartman [10]. Seja $f(x) = 11\left\lfloor \frac{x}{8} \right\rfloor + \left\lfloor \frac{3(x \bmod 8)}{2} \right\rfloor$. Dependendo da condição testada na linha 3, existem dois casos.

1. Existe uma $(2,2)$-sequência. O Lema 22 afirma que não é possível ordenar $\bar{\pi}$ usando uma sequência com menos de $||\bar{\iota}\bar{\pi}^{-1}||_3$ 2-moves. Seja $m = ||\bar{\iota}\bar{\pi}^{-1}||_3 - 2$ a 3-norma de $\bar{\iota}\bar{\pi}^{-1}$ após a aplicação de uma $(2,2)$-sequência. O Algoritmo 6 ordena $\bar{\pi}$ usando no máximo $f(m) + 2$ transposições, dando uma razão de aproximação de no máximo $\frac{f(m)+2}{m+2}$. Na Tabela 3.1, podemos ver que $\frac{f(m)+2}{m+2} \leq \frac{11}{8}$ para todo $0 \leq r \leq 7$, onde $m = 8l + r$ e $l \geq 0$.

2. Não há $(2,2)$-sequência. Se $||\bar{\iota}\bar{\pi}^{-1}||_3 = 1$, então existe apenas um ciclo 3 orientado em $\bar{\iota}\bar{\pi}^{-1}$. Neste caso, há um 2-movimento e o teorema vale. Caso contrário, podemos aumentar o limite inferior do Lema 22 em 1, visto que pelo menos um 0-movimento é necessário para ordenar $\bar{\pi}$. Seja $m = ||\bar{\iota}\bar{\pi}^{-1}||_3$. A razão de aproximação dada por Algoritmo 6 é no máximo $\frac{f(m)}{m+1}$. A tabela 3.1 também mostra que, $\frac{f(m)}{m+1} \leq \frac{11}{8}$, para todo $0 \leq r \leq 7$, onde $m = 8l + r$ e $l \geq 0$.

□

Tabela 3.1: Para todo $0 \leq r \leq 7$, onde $m = 8l + r$ e $l \geq 0$, a razão de aproximação dada pelo Algoritmo 6 é no máximo $\frac{11}{8} = 1,375$.

r	0	1	2	3	4	5	6	7
$\frac{f(m)+2}{m+2}$	$\frac{11l+2}{8l+2}$	$\frac{11l+4}{8l+3}$	$\frac{11l+5}{8l+4}$	$\frac{11l+6}{8l+5}$	$\frac{11l+8}{8l+6}$	$\frac{11l+9}{8l+7}$	$\frac{11l+11}{8l+8}$	$\frac{11l+12}{8l+9}$
$\frac{f(m)}{m+1}$	$\frac{11l}{8l+1}$	$\frac{11l+2}{8l+2}$	$\frac{11l+3}{8l+3}$	$\frac{11l+4}{8l+4}$	$\frac{11l+6}{8l+5}$	$\frac{11l+7}{8l+6}$	$\frac{11l+9}{8l+7}$	$\frac{11l+10}{8l+8}$

Capítulo 4

Resultados experimentais

Neste capítulo, serão apresentados, na Seção 4.1, resultados experimentais obtidos para todas as permutações de comprimento n, $2 \leq n \leq 12$, de implementações do Algoritmo EH e do Algoritmo 6. O desempenho de ambos os algoritmos, relativamente à porcentagem de distâncias computadas que são iguais à distância de transposição, será comparado com outros disponíveis na literatura. Na Seção 4.2, serão apresentados resultados experimentais envolvendo a ordenação de permutações longas de comprimento variando entre 20 e 500. Os resultados são comparados com experimentos semelhantes realizados em outros estudos. Finalmente, na Seção 4.3, serão mostrados outras duas falhas encontradas no Algoritmo EH, descobertas durante a sua implementação: um afetando ambas as versões publicadas [10, 28] e outro afetando apenas a versão publicada em [28].

4.1 Resultados obtidos para permutações curtas de comprimento n, $2 \leq n \leq 12$

As implementações do Algoritmo 6 e do Algoritmo EH foram testadas usando o *Rearrangement Distance Database*, fornecido pela ferramenta GRAAu [34]. Calculamos todas as distâncias aproximadas de transposição usando ambos os algoritmos para todas as permutações de comprimento n, $2 \leq n \leq 12$.

Conforme apresentado pela Tabela 4.1, a razão de aproximação obtida pelo Algoritmo EH excede 1.375. Por outro lado, o Algoritmo 6 não excede a razão de $1.33\bar{3}$. No entanto, presumimos que aproximações de 1.375 podem aparecer para permutações em S_n, $n \geq 16$, pois para existir uma $(11, 8)$-sequência, $Supp(\bar{\iota}\bar{\pi}^{-1})$ tem que ter pelo menos 17 símbolos.

Também comparamos a porcentagem de distâncias aproximadas calculadas que são iguais à distância de transposição, computadas pelo Algoritmo 6 e o Algoritmo EH, com outras disponíveis na literatura (Tabela 4.2). Em particular, adicionamos à comparação

Tabela 4.1: Comparação das razões de aproximação máximas dadas pelo Algoritmo EH (EH) e o Algoritmo 6 (Alg6). A tabela inclui outras métricas, como a taxa de aproximação média e a distância aproximada média dada por cada algoritmo e o número de vezes que o Algoritmo EH excede a taxa de aproximação de 1.375, bem como o tempo consumido por cada algoritmo para ordenar todas as permutações de cada comprimento. Os valores decimais estão truncados em 4 casas.

n	**Diâmetro**	**Razão de aprox. max.**		**Razão de aprox. média**		**Distância aprox. média**		**Número de vezes EH excedeu a** 1.375**-aproximação**	**Tempo para ordenar as permutações**[*]	
		EH	**Alg6**	**EH**	**Alg6**	**EH**	**Alg6**		**EH**	**Alg6**
2	1	1.00	1.00	1.0	1.0	1.00	1.00	0	$< 1s$	$< 1s$
3	2	1.00	1.00	1.0	1.0	1.20	1.20	0	$< 1s$	$< 1s$
4	3	1.00	1.00	1.0	1.0	1.6086	1.6086	0	$< 1s$	$< 1s$
5	3	1.00	1.00	1.0	1.0	2.0924	2.0924	0	$< 1s$	$< 1s$
6	4	$1.33\overline{3}$	1.00	1.0004	1.0	2.6063	2.6050	0	$< 1s$	$< 1s$
7	5	$1.33\overline{3}$	1.25	1.0129	1.0113	3.1762	3.1704	0	$< 1s$	$< 1s$
8	6	1.5	1.25	1.0210	1.0183	3.7178	3.7076	2	$< 2s$	$< 2s$
9	6	1.5	1.25	1.0301	1.0256	4.2796	4.2603	20	$\approx 10s$	$\approx 13s$
10	7	1.5	1.25	1.0341	1.0282	4.8051	4.7772	110	$\approx 3m$	$\approx 2m$
11	8	1.5	$1.33\overline{3}$	1.0392	1.0321	5.3526	5.3157	440	$\approx 35m$	$\approx 30m$
12	9	1.5	$1.33\overline{3}$	1.0415	1.0336	5.8694	5.8248	1448	$\approx 8.5h$	$\approx 8.1h$

[*] As permutações de cada comprimento foram ordenadas em paralelo usando um conjunto de 8 *threads*.

um algoritmo com uma razão de aproximação superior a 1.5, mas com bons resultados [3]; um 1.5-aproximativo usando uma abordagem algébrica similar [2]; e outro algoritmo 1.375-aproximativo, que utiliza uma estratégia semelhante à do Algoritmo EH.

Tabela 4.2: Comparação das porcentagens de distâncias computadas que são iguais à distância de transposição, dadas por diferentes algoritmos (WDM [1], M [2], BPwh [3] e DD [4]), em comparação com o Algoritmo EH e o Algoritmo 6. Os valores decimais estão truncados em 2 casas.

n	**WDM**	**M**	**BPwh**	**DD**	**EH**	**Alg6**
2	-	100.00	100.00	-	100.00	**100.00**
3	-	100.00	100.00	-	100.00	**100.00**
4	-	100.00	100.00	100.00	100.00	**100.00**
5	-	100.00	100.00	100.00	100.00	**100.00**
6	99.17	100.00	100.00	100.00	99.86	**100.00**
7	98.58	100.00	100.00	100.00	94.90	**95.47**
8	97.11	99.69	99.91	100.00	91.64	**92.65**
9	96.05	99.17	99.72	99.99	86.62	**88.54**
10	94.12	98.09	-	99.97	83.80	**86.53**
11	92.81	96.90	-	-	79.40	**82.98**
12	-	-	-	-	76.67	**80.91**

Conforme mostrado na Tabela 4.2, em relação à porcentagem de distâncias computadas que são iguais à distância de transposição, o melhor algoritmo parece ser o algoritmo de Dias e Dias [4], embora não se tenha resultados para $n > 10$. É importante ressaltar que este algoritmo emprega diversas heurísticas, algumas introduzidas em um trabalho anterior [35], para melhorar o desempenho do Algoritmo EH. Em princípio, um subconjunto das heurísticas empregadas de forma gulosa pelo algoritmo de Dias e Dias [4] garantiriam uma $(2, 2)$-sequência inicial, caso existisse. Entretanto, em simulações feitas com este algoritmo, observou-se que ele não considera todos os casos possíveis e, eventualmente, perde uma $(2, 2)$-sequência inicialmente existente. Uma outra heurística empregada por esse algoritmo consiste na busca por um segundo 2-movimento usando uma técnica similar de "olhar-à-frente" (*lookahead*). Entretanto, a aplicação desta heurística é opcional[1] e, quando aplicada, faz a complexidade de tempo do algoritmo subir de $O(n^4)$ para $O(n^7)$.

[1]Para garantir a razão de aproximação 1.375, a aplicação da heurística de *lookahead* deveria ser incondicionalmente o primeiro passo do algoritmo, visto que a aplicação das outras heurísticas podem conduzir à perda de uma eventual $(2, 2)$-sequência inicialmente existente.

4.2 Resultados obtidos para permutações longas de comprimento máximo 500

Nesta seção, mostramos o desempenho de ambos Algoritmo 6 e Algoritmo EH para permutações longas de comprimento máximo 500. Para isso, criamos um conjunto de dados de permutações longas com comprimentos variando entre 20 e 500, com incrementos de 10. Para cada um dos 49 conjuntos, 1000 instâncias foram geradas aleatoriamente, usando o algoritmo Fisher-Yates[2] [36]. A Figura 4.1 mostra as razões de aproximação máxima e média obtidas. Deve-se notar que as razões de aproximação foram calculadas em relação ao limite inferior dado pelo Teorema 3, uma vez que é impraticável calcular a distância exata de permutações tão longas. Um experimento semelhante foi realizado por Dias e Dias [35], mas em seu experimento, eles trabalharam com conjuntos menores, também variando de 20 a 500, com incrementos de 10, mas contendo apenas 100 de instâncias de cada comprimento. Ao comparar os resultados, podemos concluir que o Algoritmo 6 e o de Dias e Dias [35] alcançam resultados semelhantes. Dias e Dias [4] também realizaram experimentos com permutações longas, mas com comprimentos variando apenas de 10 a 100, com incrementos de 10, onde cada conjunto continha 100 de instâncias e coletaram os tempos de execução. Ao comparar os resultados apresentados em seu artigo, podemos concluir que o Algoritmo 6 tem um desempenho superior.

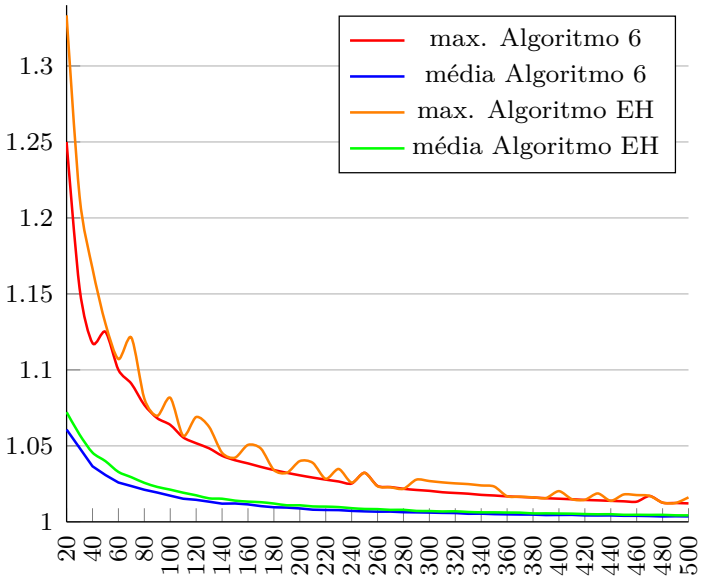

Figura 4.1: Razões de aproximação média e máxima obtidas para cada comprimento em nosso conjunto de permutações longas.

[2]Este algoritmo produz permutações aleatórias de forma imparcial, i.e., cada permutação é igualmente provável de ser gerada.

A Figura 4.2 mostra quanto tempo (em minutos) cada algoritmo (Algoritmo 6 e Algoritmo EH) levou para ordenar todas as 1000 instâncias de cada um dos 49 conjuntos. Os resultados apresentados nessa esta figura mostram que, apesar de possuir uma complexidade de pior caso alta (que acontece quando $\overline{\iota\overline{\pi}}^{-1}$, ou equivalentemente, $G(\pi)$, tem apenas um único ciclo orientado) o Algoritmo 6 tem bom desempenho na prática, até mesmo superando o Algoritmo EH neste experimento.

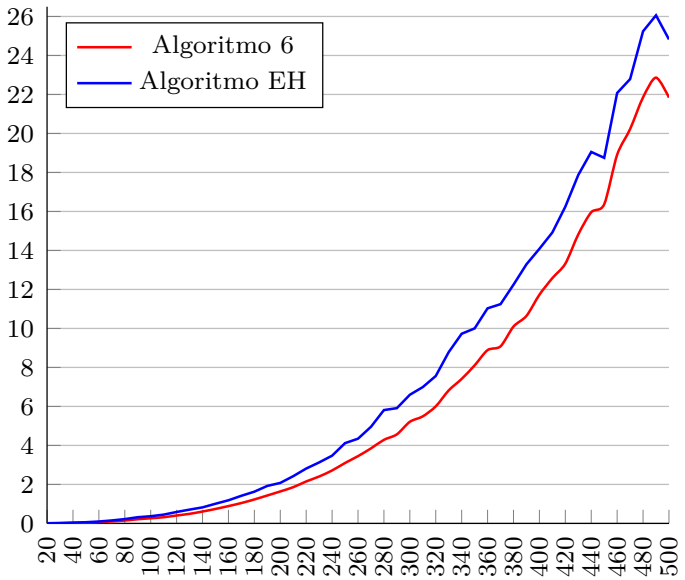

Figura 4.2: Tempo em minutos que cada algoritmo levou para ordenar todas as 1000 instâncias de cada comprimento de nosso conjunto de permutações longas.

O conjunto de permutações longas usado nos experimentos, as estatísticas computadas, bem como os códigos fontes de ambas implementações, do Algoritmo 6 e do Algoritmo EH, estão disponíveis em [33]. Todos os experimentos foram executados em um computador equipado com processador Intel Core™ i7-8665U, com 8 *cores* lógicos e 48GB de RAM.

4.3 Outras falhas encontradas durante a implementação do Algoritmo EH

Deve-se notar que Elias e Hartman [10] não forneceram uma implementação publicamente disponível de seu algoritmo, que poderíamos usar como implementação de referência. Até onde sabemos, a única implementação do Algoritmo EH relatada na literatura, sem o uso de heurísticas, é a de Dias e Dias [35]. Embora disponível publicamente, o algoritmo de Dias e Dias [35] não se assemelha ao algoritmo originalmente proposto por Elias e Hartman [10], pois este fia-se em um conjunto com alguns milhões de casos, enquanto que o conjunto de casos gerados no âmbito de Elias e Hartman [10] consiste em apenas

$\approx$ 80.000 casos. Além disso, o algoritmo apresenta baixo desempenho. Isso nos levou a implementar o Algoritmo EH de raiz.

É de se destacar que nossa implementação do Algoritmo EH está mais próxima da versão[3] apresentada anteriormente no WABI em 2005 [28], pois encontramos uma falha no algoritmo descrito em [10] (os algoritmos são apresentados de forma diferente em ambas as versões do trabalho). A falha tem a ver com a aplicação de $\frac{11}{8}$-sequências quando $G(\hat{\pi})$ contém apenas componentes pequenos ruins. Conforme apresentado em [10], uma vez que todos os componentes pequenos ruins são identificados, o algoritmo entra em um laço e aplica continuamente $\frac{11}{8}$-sequências (dadas por seu Lema 17 [10]), até que o número de ciclos em $G(\hat{\pi})$ seja menor que 8. No entanto, encontramos casos em que a aplicação de $\frac{11}{8}$-sequências dadas pelo seu Lema 17 [10] pode criar componentes pequenos em $G(\hat{\pi})$ que não são ruins, o que pode eventualmente impedir a aplicação do lema nas próximas iterações. Um desses casos é quando temos uma permutação que consiste em vários colares não-orientados de comprimento 6 [10], lado a lado. Como ilustração, considere a permutação $\hat{\pi}$ = [17 16 3 2 1 6 5 4 9 8 7 12 11 10 15 14 13 18 35 34 21 20 19 24 23 22 27 26 25 30 29 28 33 32 31], cujo $G(\hat{\pi})$ (Figura 4.3) consiste precisamente de dois colares não-orientados de comprimento 6, lado a lado. Como a soma de 3-ciclos é 12, o Lema 17 [10] nos garante a existência de uma $\frac{11}{8}$-sequência. A $(11, 8)$-sequência dada por Elias e Hartman [10] para esta permutação é $\tau_1 = \tau(19, 21, 23)$, $\tau_2 = \tau(8, 25, 29)$, $\tau_3 = \tau(21, 31, 35)$, $\tau_4 = \tau(24, 30, 36)$, $\tau_5 = \tau(9, 26, 34)$, $\tau_6 = \tau(16, 33, 35)$, $\tau_7 = \tau(11, 21, 32)$, $\tau_8 = \tau(4, 12, 31)$, $\tau_9 = \tau(9, 17, 33)$, $\tau_{10} = \tau(2, 10, 28)$, $\tau_{11} = \tau(1, 8, 21)$. Após a aplicação desta sequência, temos que $\hat{\pi}$ = [1 6 5 23 24 25 26 27 28 11 10 15 16 17 18 19 20 21 22 2 3 4 9 8 7 12 13 14 29 30 31 32 33 34 35]. Observe que agora $G(\hat{\pi})$ (Figura 4.4) contém um componente pequeno de quatro 3-ciclos não orientados, que apesar de pequeno, não é ruim.

Para evitar o problema descrito acima, em nossa implementação do Algoritmo EH, fizemos uma alteração, na qual aplicamos uma $\frac{11}{8}$-sequência tão logo a soma do número de 3-ciclos dos componentes pequenos ruins, assim que são identificados no laço principal, é maior que 7, mas dentro do próprio laço (linha 5 do algoritmo descrito em [10]), em oposição à sua posição dentro de um laço próprio (linha 6 [10]). Uma solução semelhante é empregada no Algoritmo 6 (linha 34).

Encontramos outra falha no último laço de ambas as versões do Algoritmo EH [10, 28]. Nem sempre é possível aplicar uma $(3, 2)$-sequência naquele ponto. Eventualmente, pode existir apenas um 2-movimento, como o próprio Lema 7 [10] afirma. Tomemos, por exemplo, a permutação $\hat{\pi}$ = [14 13 3 2 1 6 5 4 9 8 7 12 11 10] cujo $G(\hat{\pi})$ (Figura 4.5) consiste em um colar não orientado de comprimento 5. Observe que não há $\frac{11}{8}$-sequência para aplicar em $\hat{\pi}$. No último laço [10, 28], Elias e Hartman aplicam

[3]Uma vez que esta versão utiliza um único laço para aplicar $\frac{11}{8}$-sequências.

Figura 4.3: $G([17\ 16\ 3\ 2\ 1\ 6\ 5\ 4\ 9\ 8\ 7\ 12\ 11\ 10\ 15\ 14\ 13\ 18\ 35\ 34\ 21\ 20\ 19\ 24\ 23\ 22\ 27\ 26\ 25\ \ 30\ 29\ 28\ 33\ 32\ 31])$, consistindo de dois colares não-orientados de comprimento 6, lado a lado.

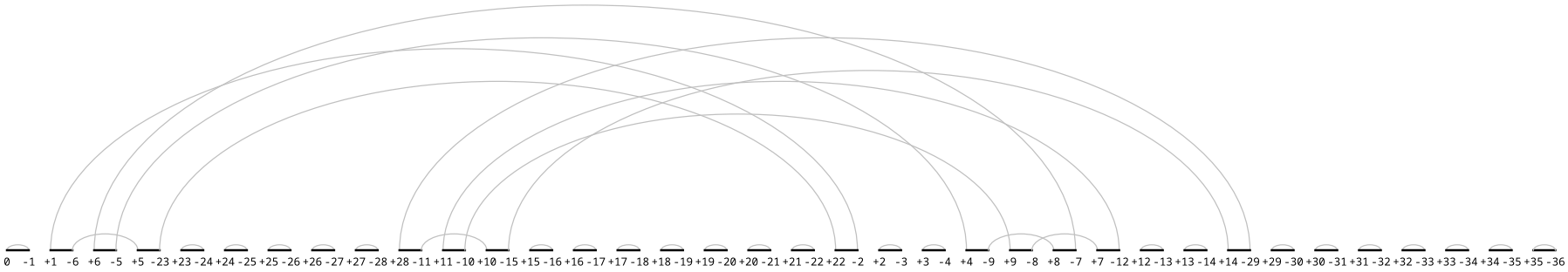

Figura 4.4: $G([1\ 6\ 5\ 23\ 24\ 25\ 26\ 27\ 28\ 11\ 10\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 2\ 3\ 4\ 9\ 8\ 7\ 12\ 13\ 14\ 29\ \ 30\ 31\ 32\ 33\ 34\ 35])$, contendo um componente que embora pequeno, não é ruim.

duas $(3,2)$-sequências: $\tau_1 = \tau(1,10,14)$, $\tau_2 = \tau(4,6,15)$, $\tau_3 = \tau(3,5,14)$, e a seguir $\tau_1 = \tau(4,8,9)$, $\tau_2 = \tau(2,5,8)$, $\tau_3 = \tau(1,3,6)$. Após aplicar essas duas sequências, temos $\hat{\pi} = [1\ 6\ 7\ 8\ 2\ 3\ 4\ 5\ 9\ 10\ 11\ 12\ 13\ 14]$ cujo $G(\hat{\pi})$ (Figura 4.6) contém apenas um 3-ciclo orientado, impossibilitando a aplicação de mais uma $(3,2)$-sequência. Nesse caso particular, o 2-movimento $\tau(2,5,9)$ conclui a ordenação de $\hat{\pi}$. Nossa implementação [33] do Algoritmo EH inclui uma "correção" para esta falha, aplicando um $(3,2)$-sequência ou um 2-movimento, dependendo do caso.

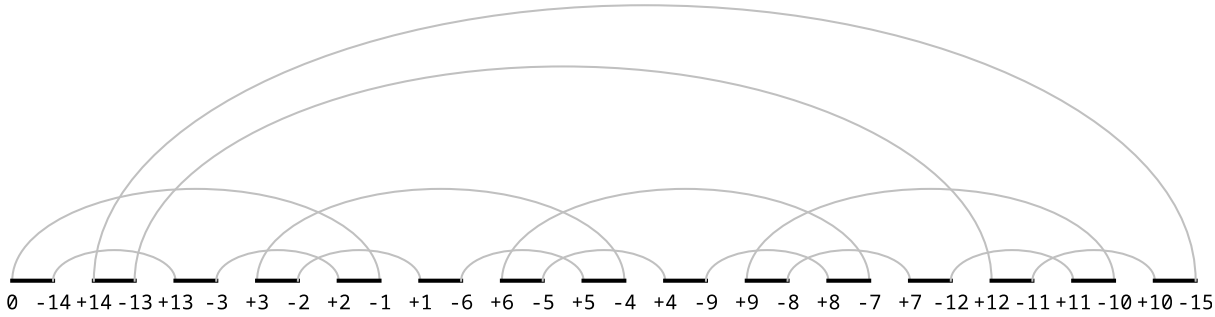

Figura 4.5: $G([14\ 13\ 3\ 2\ 1\ 6\ 5\ 4\ 9\ 8\ 7\ 12\ 11\ 10])$.

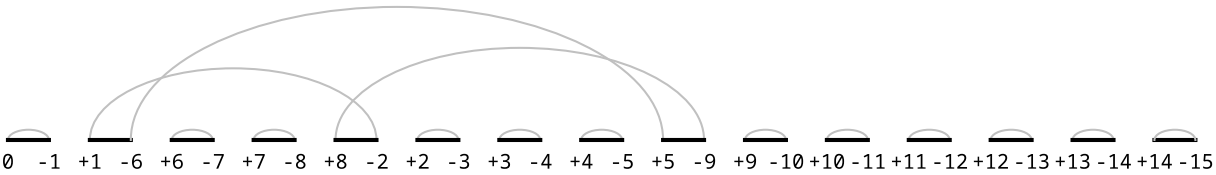

Figura 4.6: $G([1\ 6\ 7\ 8\ 2\ 3\ 4\ 5\ 9\ 10\ 11\ 12\ 13\ 14])$.

Capítulo 5

Avanços em direção a uma razão de aproximação inferior a 1.375

Neste capítulo, serão apresentados os resultados obtidos de uma investigação com o objetivo de desenvolver uma solução aproximada para o SBT, com razão de aproximação inferior a 1.375. A princípio, visamos a razão $1.33\overline{3}$. Essa investigação baseou-se na generalização do método apresentado na Seção 3.3.2, que originalmente procurava por $\frac{11}{8}$-sequências, método este delineado no início da Seção 5.1. Nessa mesma seção, mostramos os resultados obtidos da busca por sequências de transposição provendo razão de aproximação $1.33\overline{3}$, onde apenas não encontramos tais sequências para 11 combinações problemáticas de componentes pequenos ruins, que podem ser visualizados no Anexo I. Na Seção 5.2, apresentamos os detalhes de uma busca por uma $(19, 14)$-sequência, para uma possível extensão de uma das combinações problemáticas, usando uma abordagem de computação distribuída que, em tese, poderá ser utilizada na busca por sequências mais longas. A busca por uma $(19, 14)$-sequência, para a configuração em causa, levou 38 dias para ser concluída e a sequência procurada não foi encontrada.

5.1 Busca por sequências de transposição provendo razão de aproximação $1.33\overline{3}$

O método empregado nesta seção, derivado do método utilizado na Seção 3.3.2, pode ser dividido em dois passos. O primeiro consiste em estender as configurações básicas até que se atinjam configurações suficientes que permitam a aplicação de uma sequência de transposições provendo razão de aproximação $1.33\overline{3}$. O segundo passo consiste em estender as combinações de componentes pequenos ruins encontrados no passo anterior, até que

também se atinjam combinações permitindo uma sequência de transposições provendo a mesma razão de aproximação.

Antes de apresentar os resultados obtidos, é necessário redefinir alguns conceitos, uma vez que agora lidamos com configurações maiores que 9. Seja Γ uma configuração de $\overline{\iota}\overline{\pi}^{-1}$. Se $||\Gamma||_3 \leq 10$, então Γ é uma *configuração pequena*; caso contrário, uma *configuração grande*. *Componentes pequenos ruins* são componentes pequenos que não permitem a aplicação de $\frac{12}{9}$-sequências.

5.1.1 Extensão de configurações básicas

O programa [33] foi adaptado para procurar por sequências provendo a razão de aproximação $1.33\overline{3}$ por meio de extensões suficientes das configurações básicas. Este novo programa [37] nos permitiu obter o importante resultado a seguir:

Lema 44. *Toda configuração suficiente* Γ *de* $\overline{\iota}\overline{\pi}^{-1}$*, tal que* Γ *seja grande (ou seja,* $||\Gamma||_3 \geq 11$*), permite uma* $\frac{12}{9}$*-sequência.*

Durante a análise que levou ao Lema 44, dois novos componentes pequenos ruins surgiram, além dos já enumerados no Lema 36. São eles: o *colar não-orientado de tamanho* 8 (Figura 5.1); e o *colar não-orientado de tamanho* 10 (Figura 5.2).

O resultado da análise de casos que prova o Lema 44 consiste de 484.707 arquivos HTML e está disponível na forma de uma interface web em [38].

5.1.2 Análise das combinações de componentes pequenos ruins

Nesta etapa, lidamos com as combinações dos componentes pequenos ruins encontrados na seção anterior. Em suma, o programa [37] encontrou $\frac{16}{12}$-sequências para todas as combinações geradas de 3-norma maior ou igual a 12, exceto para 11 *combinações problemáticas* (figuras no Anexo I). Curiosamente, todas as 11 configurações problemáticas são combinações do par intercalante não-orientado (Figura 3.6).

A análise das combinações de componentes pequenos ruins feita nesta seção produziu 60.937 arquivos HTML e também está disponível na interface web [38].

Toda a análise, incluindo a análise das extensões das configurações básicas realizada na seção anterior, leva cerca de 5 dias para ser executada, usando todos os *cores*, em paralelo, de um computador equipado com um processador AMD Ryzen Threadripper 2990WX, com 64 *cores* lógicos e 64GB de RAM.

Todo o código implementado relativo às análises feitas nesta seção e na anterior está publicamente disponível em [37].

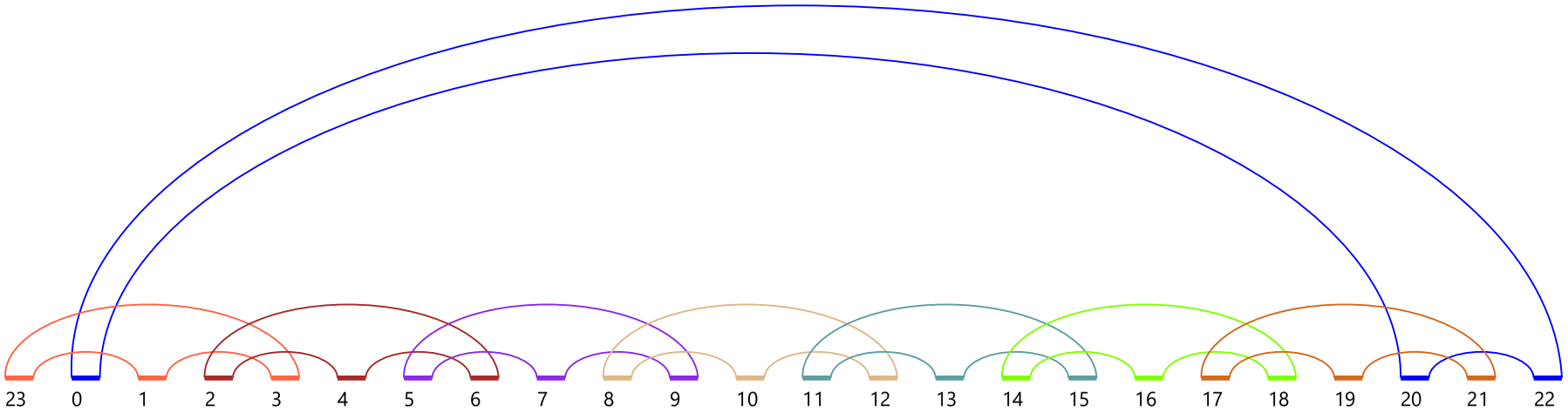

Figura 5.1: O colar não-orientado de tamanho 8.

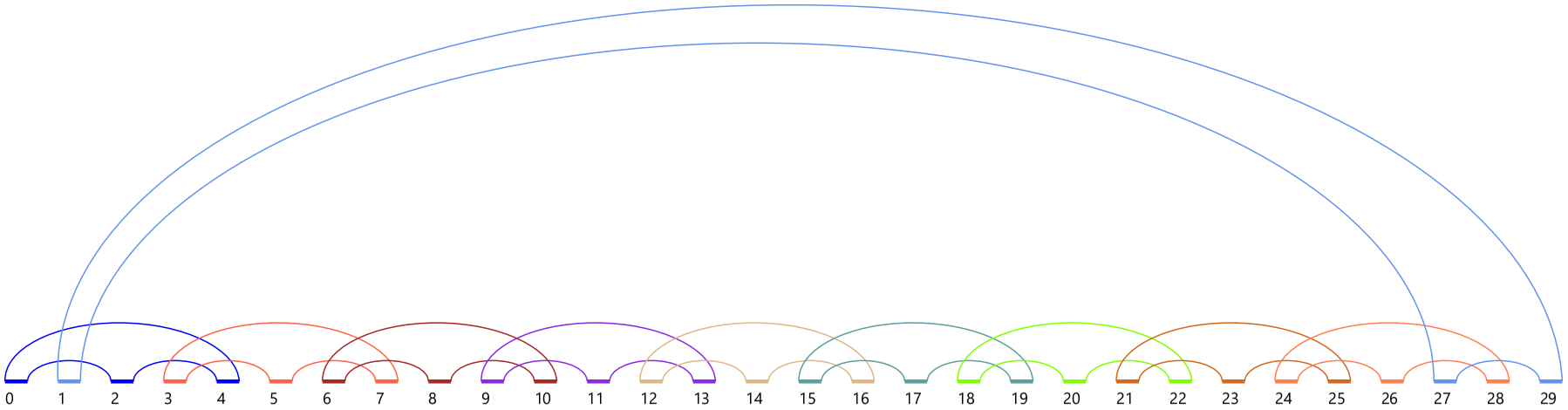

Figura 5.2: O colar não-orientado de tamanho 10.

5.2 Busca por uma $(19, 14)$-sequência

Entre as combinações problemáticas, intuímos que as "mais difíceis de ordenar" são aquelas formadas por múltiplos pares intercalantes não-orientados (Figura 3.6) lado a lado, como a combinação problemática n^{o} 1 (Figura I.1 do Anexo I). Acreditamos que essas combinações são as "mais problemáticas", porque são as que parecem ter o menor potencial de geração de configurações com muitas arestas cinza a se cruzarem durante o processo de ordenação, à medida em que os 0-movimentos são aplicados, o que conjecturamos ser essencial para existência posterior de 2-movimentos.

De acordo com o método proposto, como não existem $\frac{16}{12}$-sequências a aplicar nas combinações problemáticas, o próximo passo consiste em estendê-las. Neste caso, considerando apenas a adição de um par intercalante não-orientado, uma possível extensão da combinação n^{o} 1 é uma outra combinação consistindo de 42 símbolos, com 7 pares intercalantes não-orientados lado a lado, cuja 3-norma é igual a 14. Primeiramente, foram buscadas $(4, 3)$-, $(8, 6)$-, $(12, 9)$- e $(16, 12)$-sequências[1] para essa nova combinação, porém nenhuma foi encontrada. Também buscamos por uma $(15, 11)$-sequência, provendo razão de aproximação $1.\overline{36}$, mas outra vez, não fomos bem sucedidos. Resolvemos, então, experimentar a busca por uma $(19, 14)$-sequência, provendo uma razão de aproximação $1.3\overline{571428}$. Note que as $(19, 14)$-sequências eram as únicas sequências restantes possíveis de serem buscadas, provendo uma razão de aproximação inferior a 1.375. Conjecturamos que se fôssemos bem sucedidos na busca por uma tal sequência, então não haveria outras combinações problemáticas com 3-norma maior ou igual a 14. Isso abriria o caminho para o desenvolvimento de uma solução $1.3\overline{571428}$-aproximativa para o SBT.

Como o espaço a ser explorado na busca por uma $(19, 14)$-sequência é enorme, foi desenvolvida uma solução de computação distribuída, de maneira a permitir que múltiplos computadores trabalhassem em paralelo, com o objetivo de acelerá-la. Os detalhes são apresentados a seguir.

5.2.1 Usando uma abordagem de computação distribuída

A busca começa com a enumeração de todas as possíveis $(19, 14)$-sequências a aplicar quando se sabe que não é possível aplicar uma sequência mais curta, provendo razão de aproximação máxima $\frac{19}{14}$. Uma classe[2] foi escrita apenas com o propósito de gerar tais sequências. Nesse caso, a lista de todas as $(19, 14)$-sequências possíveis consiste de

[1] Existem outras $\frac{16}{12}$-sequências além das que foram listadas (e.g. $(9, 7)$- e $(13, 10)$-sequências), porém não são buscadas, pois todas começam com uma $(4, 3)$-, $(8, 6)$- ou $(12, 9)$-sequência.

[2] *br.unb.cic.tdp.util.ListSequences* [37]

611 itens. Como a lista é muito longa, não convém enumerá-la aqui, mas ela pode ser visualizada em sua própria classe[3] em [37].

Após a enumeração das sequências, que podem ser transformadas em *strings* por meio do simples encadeamento do valor μ de cada movimento da sequência, elas são convertidas em uma árvore de prefixos. Apenas como ilustração, a Figura 5.3 mostra a árvore de prefixos gerada a partir das possíveis $(8, 6)$-sequências aplicáveis a uma configuração que não permite uma $(4, 3)$-sequência. Nesse caso particular, as possíveis $(8, 6)$-sequências, representadas como *strings*, são "02202222", "02022222" e "00222222".

O algoritmo de busca (Algoritmo 7) funciona, essencialmente, como uma busca em profundidade, caminhando pela árvore de prefixos e chamando recursivamente a função de busca para cada 0- ou 2-movimento possível, possível de ser aplicado à configuração atual e determinado pelo nó corrente da árvore. Cada 0- ou 2-movimento aplicado à configuração, durante a busca, é empilhado e desempilhado (se a busca for infrutífera para aquele galho da árvore) de uma pilha que é fornecida como argumento para função de busca. Se se atingir, durante o caminhamento, um nó folha da árvore de prefixos, significa que a sequência de ordenação foi encontrada e ela estará na pilha.

Algoritmo 7 Algoritmo utilizado na busca por sequências longas de transposições

```
1: função BUSCA(Γ, p, g)        ▷ Γ é uma configuração, p é uma pilha de μ-movimentos e g é um nó da árvore de prefixos
2:     para cada μ-movimento m, indicado por g, possível de ser aplicado em Γ faça
3:         empilhe m em p
4:         se g é um nó folha então
5:             retorne p                                   ▷ a sequência procurada foi encontrada
6:         senão
7:             seja Γ' o resultado de Γm^{-1}                                   ▷ m é aplicado
8:             para cada nó g' filho de g faça
9:                 seja s o resultado de BUSCA(Γ', p, g')
10:                se s é uma pilha vazia então
11:                    desempilhe m de p
12:                senão
13:                    retorne s
14:                fim se
15:            fim para
16:        fim se
17:    fim para
18:    retorne uma pilha vazia
19: fim função
```

Para dividir o problema e, assim, permitir que vários computadores trabalhem em paralelo na busca pela $(19, 14)$-sequência, utilizamos a abordagem *Fork/Join* [39] provida pela biblioteca padrão do Java, onde, ao caminhar pela árvore de prefixos, se ainda não passamos por três 0-movimentos, fazemos *fork*. Caso contrário, ao invés de computar

[3] *br.unb.cic.tdp.util._19_14Seqs* [37]

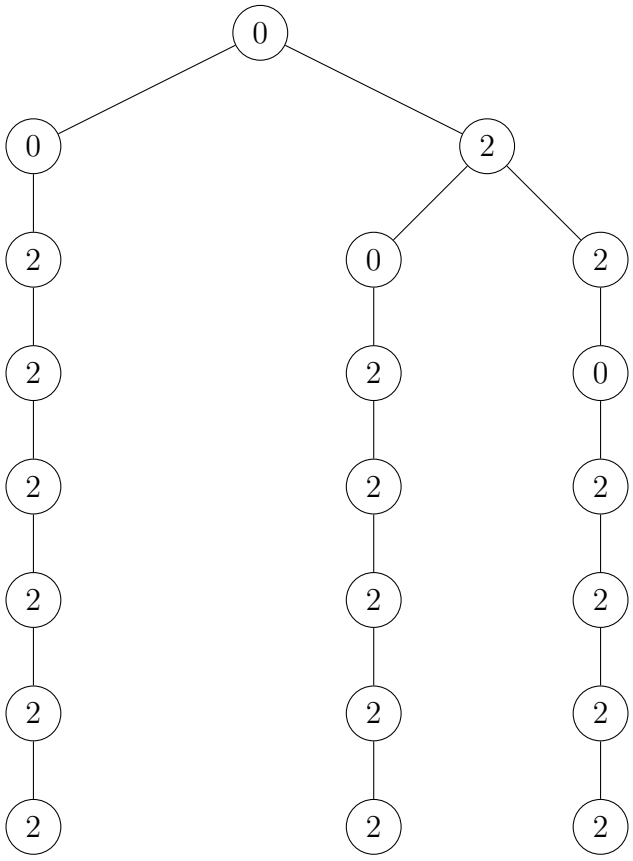

Figura 5.3: Árvore de prefixos gerada a partir das *strings* “02202222”, “02022222” e “00222222”, que correspondem às possíveis $(8, 6)$-sequências a aplicar em uma configuração que não permite uma $(4, 3)$-sequência.

o restante da busca diretamente na *thread*, salvamos em um arquivo: a configuração parcialmente ordenada, uma cópia da pilha e uma referência para o nó atual da árvore de prefixos. Através deste procedimento, foram geradas aproximadamente 38 milhões de entradas no arquivo. O número de três 0-movimentos foi escolhido de forma empírica. Um número menor gera menos entradas no arquivo, mas faz com que o tempo de computação restante da busca (os detalhes desta etapa são explicados na sequência), para cada entrada, seja muito elevado. Por outro lado, um número maior implica em mais entradas no arquivo e tem o efeito de fazer com que o procedimento de divisão do problema leve quase tanto tempo quanto a própria busca em si. O procedimento divisão da busca, conforme esboço acima, está representado no Algoritmo 8.

Em seguida, as entradas do arquivo foram divididas em lotes de 20 e enviadas como uma única mensagem para uma fila em um sistema de mensageria *RabbitMQ* [40], instalado em um computador na plataforma *Amazon Web Services* (AWS). Com a divisão das entradas em lotes, cerca de $1,9$ milhões de mensagens foram criadas. O número de 20 entradas no lote também foi escolhido de forma empírica. Um número menor implica em mais tráfego

Algoritmo 8 Algoritmo para a divisão da busca

```
1: procedimento DIVISÃO(Γ, p, g, f)      ▷ Γ é uma configuração, p é uma pilha de μ-movimentos, g é um nó da árvore de prefixos e f é um arquivo
2:     se a quantidade de 0-movimentos até a raiz de g é igual a 3 então
3:         salve no arquivo f: Γ, p, e uma referência para o nó g na árvore de prefixos
4:     senão
5:         para cada μ-movimento m, indicado por g, possível de ser aplicado em Γ faça
6:             empilhe m em p
7:             seja Γ' o resultado de Γm^{-1}                                   ▷ m é aplicado
8:             seja p' uma cópia da pilha p
9:             para cada nó g' filho de g faça
10:                fork de DIVISÃO(Γ', p', g')
11:            fim para
12:            desempilhe m de p
13:        fim para
14:    fim se
15: fim procedimento
```

de rede, a ponto de impactar, de forma significativa, o tempo total de processamento da busca. Um número maior, por outro lado, faz com que o consumidor demore mais tempo para consumir uma mensagem, o que, em última instância, faz com que surjam erros de *timeout* na comunicação consumidor-*RabbitMQ*.

Com as mensagens disponíveis no sistema de mensageria, qualquer computador com acesso à *internet*, e com as credenciais necessárias para a ligação com o *RabbitMQ*, pode participar da busca por meio do consumo de mensagens. Consumir uma mensagem, essencialmente, consiste em tentar "terminar", para cada entrada do lote, a ordenação parcial que foi iniciada anteriormente no processo de *Fork/Join*. Se algum consumidor conseguir encontrar a $(19, 14)$-sequência, então o mesmo envia uma mensagem, com a sequência encontrada, para uma outra fila, que é constantemente monitorada.

Os computadores que participam da busca podem se ligar e desligar da fila a qualquer momento. O *RabbitMQ* provê um controle transacional na confirmação de consumo das mensagem. Isso significa que ele garante que, se um consumidor se desliga subitamente, então as mensagens não confirmadas pelo mesmo sejam automaticamente re-enfileiradas.

A busca pela $(19, 14)$-sequência, seguindo o método apresentado acima, começou no dia 13 de fevereiro de 2022 e terminou, sem encontrar a sequência procurada, no dia 23 de março de 2022. Durante esses 38 dias, 5 computadores se dedicaram à busca em tempo integral, totalizando 144 *cores* (uma *thread* consumidora de mensagens é instanciada para cada *core* lógico disponível no computador que participa da busca). Foram eles:

1. um computador equipado com processador AMD Epyc 7742, instalado na Universidade Federal de Goiás (UFG), com 64 *cores* lógicos e 300GB de RAM emprestados à busca (o computador tem, no total, 256 *cores* lógicos e 2TB de RAM, mas entretanto serve a outras pesquisas internas do Instituto de Informática da UFG);

2. um computador equipado com processador AMD Ryzen Threadripper 2990WX, com 64 *cores* lógicos e 64GB de RAM;
3. um com processador Intel Core™ i7-8665U, com 8 *cores* lógicos e 16GB de RAM;
4. um com processador Intel Core™ i3-6006U, com 4 *cores* lógicos e 12GB de RAM;
5. e outro com processador Intel Core™ i3-4005U, com 4 *cores* lógicos e 8GB de RAM.

Outros dois computadores também ajudaram regularmente na busca, mas apenas no período noturno. Foram eles:

1. um computador com processador Intel Core™ i9-11950H, com 16 *cores* lógicos e 32GB de RAM, e
2. um outro com processador Intel Core™ i7-10610U, com 8 *cores* lógicos e 48GB de RAM.

Contamos também com o auxílio, por duas semanas, de um computador alugado na AWS, equipado com um processador de arquitetura ARM, o Amazon Graviton2, contendo 64 *cores* e 128GB de RAM. Com isso, chegamos a ter picos de 232 *cores* trabalhando paralelamente na busca pela $(19, 14)$-sequência.

Vale ressaltar que, para acelerar a execução, o algoritmo de busca utiliza um *cache* interno para "lembrar" dos galhos infrutíferos da árvore de prefixos, para determinadas configurações, e ignorá-los quando reaparecem durante a execução. Por isso, além da quantidade de *cores*, a quantidade de memória RAM disponível em cada computador tem papel fundamental no desempenho da busca. Durante a implementação do algoritmo de busca, foi considerada a utilização de uma solução de *cache* distribuído, com o intuito de unificar os *caches* internos de todos os consumidores em um único *cache*. Porém, não chegamos a experimentar tal solução, pois assumimos que o tempo de latência de rede no acesso ao *cache*, imposto à execução da busca, pudesse ser muito significativo. Porém, no futuro, talvez possa fazer sentido realizar algumas experiências e reavaliar essa decisão.

As consequências da não existência da $(19, 14)$-sequência buscada nesta seção serão discutidas no próximo capítulo.

Em tese, a solução utilizada na busca por uma $(19, 14)$-sequência pode ser empregada na busca por sequências ainda mais longas, em combinações com 3-norma maior que 14, provendo razões de aproximação inferiores a 1.375, desde que haja recursos computacionais disponíveis.

Todo o código implementado para a realização da busca, descrito nesta seção, está publicamente disponível em [37].

Capítulo 6

Conclusões e trabalhos futuros

Primeiro, nesta tese mostramos que o Algoritmo EH pode produzir uma transposição extra acima da razão de aproximação prometida de 1.375. Isso ocorre quando há uma $(2, 2)$-sequência inicial na permutação original, que é "perdida" durante o processo de simplificação, e componentes pequenos ruins permanecem no grafo de ciclos após a aplicação de qualquer número de $\frac{11}{8}$-sequências.

Em seguida, propusemos um novo limite superior para a distância de transposição, usando uma abordagem algébrica, que vale para todo S_n, melhorando o limite encontrado anteriormente por Bafna e Pevzner [9, 27]. Na sequência, propusemos um novo algoritmo de aproximação para o SBT, garantindo a razão de aproximação 1.375, também para todo S_n. No melhor de nosso conhecimento, este é o primeiro algoritmo que garante uma razão de aproximação abaixo de 1.5, que não usa simplificação.

Implementações do Algoritmo EH e do Algoritmo 6 foram testadas com todas as permutações de comprimento máximo 12. Os resultados mostraram que o Algoritmo 6 não excede a razão de aproximação de 1.375 e produz uma porcentagem maior de distâncias calculadas que são iguais à distância de transposição, quando comparadas às computadas pelo Algoritmo EH. Esses percentuais também foram comparados com outros disponíveis na literatura. Considerando essa métrica, o algoritmo com melhores resultados parece ser o de Dias e Dias [4], embora não apresentem resultados para $n > 10$.

Realizamos outro experimento envolvendo permutações longas de comprimento máximo 500, geradas aleatoriamente. Os resultados mostraram que o Algoritmo 6 supera o Algoritmo EH, tanto em relação às razões de aproximação obtidas, quanto aos tempos de execução. Ainda sobre esse experimento, o Algoritmo 6 parece ser comparável ao de Dias e Dias [35], quando consideramos as razões de aproximação obtidas por ambos. Com relação aos tempos de execução, Dias e Dias [35] também realizaram algumas simulações para permutações com comprimento máximo de 100. Considerando apenas os resultados para permutações com esse comprimento máximo, nosso algoritmo parece ser mais rápido.

Dois outros problemas foram identificados ao implementar o Algoritmo EH. O primeiro tem a ver com a aplicação de $\frac{11}{8}$-sequências, quando o grafo de ciclos contém apenas componentes pequenos ruins e afeta apenas a versão publicada em [10]. O segundo está relacionado à aplicação de $(3, 2)$-sequências, quando não há $\frac{11}{8}$-sequências a aplicar e afeta ambas as versões do algoritmo descritas em [10] e [28].

A complexidade de tempo do Algoritmo 6 é alta. Um possível trabalho futuro pode ser a investigação de uma maneira mais eficiente de encontrar uma $(2, 2)$-sequência no início do algoritmo. Seguindo uma direção diferente, outro trabalho futuro pode ser a investigação de simplificações "boas", ou seja, simplificações que não têm o efeito de perder uma $(2, 2)$-sequência inicial, quando ela existe. Não temos ideia se uma tal simplificação "boa" sempre existe ou não. De qualquer forma, conjecturamos que, para buscá-la, o custo computacional seja o mesmo de procurar por uma $(2, 2)$-sequência no início.

Todos os resultados acima referidos foram publicados em um artigo científico [41] no jornal *Algorithms for Molecular Biology*.

Outra contribuição desta tese são os resultados obtidos a partir de uma investigação com o objetivo de desenvolver uma solução aproximada para o SBT com razão de aproximação inferior a 1.375, a princípio $1.33\bar{3}$. O primeiro resultado encontrado é o fato de todas as configurações suficientes de 3-norma maior ou igual a 11 permitirem a aplicação de uma $\frac{12}{9}$-sequência. Porém, ao analisar as combinações de componentes pequenos ruins descobertas durante a análise que levou ao resultado anterior, nos deparamos com 11 combinações problemáticas que não permitem a aplicação de $\frac{16}{12}$-sequências (Anexo I). De acordo com o método proposto, as combinações problemáticas tinham que ser estendidas e sequências mais longas buscadas. Experimentamos a busca por uma $(19, 14)$-sequência para uma possível extensão de uma das combinações problemáticas. Essa busca levou 38 dias para ser executada, com vários computadores a trabalhar em paralelo, porém a sequência não foi encontrada.

A não existência da $(19, 14)$-sequência reportada acima implica que, para continuar a procura por uma solução aproximada para o SBT com razão inferior a 1.375, uma nova extensão será necessária. Nas novas combinações geradas por essa extensão, terá que se buscar novamente por $\frac{19}{14}$-sequências – e no caso de não existirem, terá que se buscar por $(20, 15)$-sequências (apenas aquelas que não começam com uma $\frac{19}{14}$-sequência buscada no passo anterior). Consequentemente, espaços de busca ainda maiores poderão ter que ser explorados, uma vez que as novas combinações terão 3-norma maior ou igual a 16, com pelo menos 48 símbolos. Dependendo dos recursos computacionais disponíveis e da técnica utilizada, uma tal busca poderá levar muito tempo. Nesse sentido, a aplicabilidade de GPU's e/ou FPGA's para acelerar a busca poderá ser investigada com a colaboração de especialistas nessas áreas. Se, por acaso, não for possível acelerar a busca e as sequên-

cias mencionadas acima não forem encontradas (o que acarretará em mais extensões), então pode ser que, eventualmente, seja impraticável procurar e desenvolver uma solução aproximada para o SBT com razão inferior a 1.375.

Finalmente, pretendemos, no futuro, usar a abordagem algébrica utilizada nesta tese para estudar e resolver outros eventos de rearranjo afetando um cromossomo, como por exemplo: reversões e trocas de blocos.

Referências

[1] Walter, M. E. M. T., Z. Dias e J. Meidanis: *A new approach for approximating the transposition distance*. Em *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, páginas 199–, Washington, DC, USA, 2000. IEEE Computer Society, ISBN 0-7695-0746-8. `http://dl.acm.org/citation.cfm?id=829519.830850`. xiii, 2, 34

[2] Mira, C. V. G., Z. Dias, H. P. Santos, G. A. Pinto e M. E. M. T. Walter: *Transposition distance based on the algebraic formalism*. Em *Advances in Bioinformatics and Computational Biology, Proceedings of the Third Brazilian Symposium on Bioinformatics*, páginas 115–126, Berlin Heidelberg, Germany, 2008. Springer. xiii, 2, 3, 13, 34

[3] Walter, M. E. M. T., M. C. Sobrinho, E. T. G Oliveira, L. S. Soares, A. G Oliveira, T. E. S. Martins e T. M. Fonseca: *Improving the algorithm of Bafna and Pevzner for the problem of sorting by transpositions: a practical approach*. Journal of Discrete Algorithms, 3(2):342–361, 2005. xiii, 34

[4] Dias, U. e Z. Dias: *An improved 1.375-approximation algorithm for the transposition distance problem*. Em *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, páginas 334–337, 2010. xiii, 1, 34, 35, 50

[5] Nadeau, J. H. e B. A. Taylor: *Lengths of chromosomal segments conserved since divergence of man and mouse*. Proceedings of the National Academy of Sciences, 81(3):814–818, 1984. 1

[6] Palmer, J. D. e L. A. Herbon: *Plant mitochondrial dna evolves rapidly in structure, but slowly in sequence*. Journal of Molecular Evolution, 28:87–97, 1988. 1

[7] Koonin, E. V.: *Orthologs, paralogs, and evolutionary genomics*. Annual Review of Genetics, 39:309–338, 2005. 1

[8] Yue, F., M. Zhang e J. Tang: *Phylogenetic reconstruction from transpositions*. BMC Genomics, 9(S15):https://doi.org/10.1186/1471–2164–9–S2–S15, 2008. 1

[9] Bafna, V. e P. A. Pevzner: *Sorting by transpositions*. SIAM Journal on Discrete Mathematics, 11(2):224–240, 1998. 1, 2, 3, 6, 7, 13, 28, 50

[10] Elias, I. e T. Hartman: *A 1.375-approximation algorithm for sorting by transpositions*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 3(4):369–379, 2006. 1, 3, 4, 6, 8, 9, 10, 17, 23, 24, 25, 30, 32, 36, 37, 51

[11] Cunha, L. F. I., L. A. B. Kowada, Hausen R. A. e C. M. H. de Figueiredo: *A Faster* 1.375*-Approximation Algorithm for Sorting by Transposition*. Journal of Computational Biology, 22(11):1044–1056, 2015. `https://doi.org/10.1089/cmb.2014.0298`. 1

[12] Dias, U. e Z. Dias: *Heuristics for the transposition distance problem*. Journal of Bioinformatics and Computational Biology, 11(5):1–17, October 2013. 1

[13] Bulteau, L., G. Fertin e I. Rusu: *Sorting by transpositions is difficult*. SIAM Journal on Discrete Mathematics, 26(3):1148–1180, 2012. 1

[14] Hausen, R. A., L. Faria, C. M. H. Figueiredo e L. A. B. Kowada: *Unitary toric classes, the reality and desire diagram, and sorting by transpositions*. SIAM Journal on Discrete Mathematics, 24(3):792–807, 2010. 2

[15] Eriksson, H., K. Eriksson, J. Karlander, L. Svensson e J. Wästlund: *Sorting a bridge hand*. Discrete Mathematics, 241(1-3):289–300, 2001. 2, 28

[16] Galvão, G. e Z. Dias: *On the approximation ratio of algorithms for sorting by transpositions without using cycle graphs*. Em *Brazilian Symposium on Bioinformatics*, páginas 25–36. Springer, 2012. 2

[17] Benoît-Gagné, M. e S. Hamel: *A new and faster method of sorting by transpositions*. Em *Annual Symposium on Combinatorial Pattern Matching*, páginas 131–141. Springer, 2007. 2

[18] Guyer, S. A., L. S. Heath e J. P. C. Vergara: *Subsequence and run heuristics for sorting by transpositions*. Relatório Técnico, Virginia Polytechnic Institute & State University, 1997. 2

[19] Rusu, I.: *log-lists and their applications to sorting by transpositions, reversals and block-interchanges*. Theoretical Computer Science, 660:1–15, 2017. 2

[20] Sleator, D. D. e R. E. Tarjan: *A data structure for dynamic trees*. Journal of Computer and System Sciences, 26(3):362–391, 1983. 2

[21] Lintzmayer, C. N., G. Fertin e Z. Dias: *Sorting permutations by prefix and suffix rearrangements*. Journal of Bioinformatics and Computational Biology, 15(01):1750002, 2017. 2

[22] Oliveira, A. R., G. Jean, G. Fertin, K. L. Brito, U. Dias e Z. Dias: *A 3.5-approximation algorithm for sorting by intergenic transpositions*. Em *International Conference on Algorithms for Computational Biology*, páginas 16–28. Springer, 2020. 2

[23] Meidanis, J. e Z. Dias: *An Alternative Algebraic Formalism for Genome Rearrangements*, páginas 213–223. Springer Netherlands, Dordrecht, 2000, ISBN 978-94-011-4309-7. 2, 19

[24] Hannenhalli, S. e P. A. Pevzner: *Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals.* Journal of the ACM, 46(1):1–27, 1999. 2, 7, 8

[25] Lin, G. H. e G. Xue: *Signed genome rearrangement by reversals and transpositions: models and approximations.* Theoretical Computer Science, 259(1):513–531, 2001. 2, 8

[26] Hartman, T. e R. Shamir: *A simpler and faster 1.5-approximation algorithm for sorting by transpositions.* Information and Computation, 204(2):275–290, 2006. 2, 8

[27] Fertin, G., A. Labarre, I. Rusu, S. Vialette e E. Tannier: *Combinatorics of genome rearrangements.* MIT press, London, En, 2009. 3, 28, 50

[28] Elias, I. e T. Hartman: *A 1.375-approximation algorithm for sorting by transpositions.* Em *International Workshop on Algorithms in Bioinformatics*, páginas 204–215. Springer, 2005. 3, 32, 37, 51

[29] Dummit, D. S. e R. M. Foote: *Abstract algebra.* Wiley, Hoboken, NJ, 2004. 11

[30] Gallian, J.: *Contemporary Abstract Algebra.* Brooks Cole, Boston, MA, 7ª edição, janeiro 2009, ISBN 0547165099. 11

[31] Mira, C. V. G. e J. Meidanis: *Algebraic formalism for genome rearrangements.* Technical Report, Institute of Computing, University of Campinas, June 2005. 14, 19

[32] `http://tdp1375proof.s3-website.us-east-2.amazonaws.com/`, 2022. 22, 23

[33] `https://github.com/luizaugustogarcia/tdp1375/`, 2022. 23, 24, 25, 26, 36, 40, 42

[34] Galvão, G. R. e Z. Dias: *An audit tool for genome rearrangement algorithms.* Journal of Experimental Algorithmics (JEA), 19:1–7, 2015. 32

[35] Dias, U. e Z. Dias: *Extending Bafna-Pevzner algorithm.* Em *Proceedings of the International Symposium on Biocomputing*, páginas 23:1–23:8, New York, NY, USA, 2010. ACM. 34, 35, 36, 50

[36] Fisher, R. A. e F. Yates: *Statistical tables for biological, agricultural and medical research.* Hafner Publishing Company, 1953. 35

[37] `https://github.com/luizaugustogarcia/sbt1.333/`, 2022. 42, 45, 46, 49

[38] `http://sbt1333.s3-website.us-east-2.amazonaws.com/`, 2022. 42

[39] `https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html`, 2022. 46

[40] `https://www.rabbitmq.com/`, 2022. 47

[41] Silva, L. A. G., L. A. B. Kowada, N. R. Rocco e M. E. M. T. Walter: *A new 1.375-approximation algorithm for sorting by transpositions.* Algorithms for Molecular Biology, 17(1):1–17, 2022. 51

Anexo I

As 11 combinações problemáticas

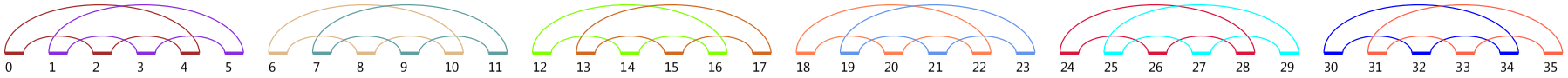

Figura I.1: Configuração problemática n° 1.

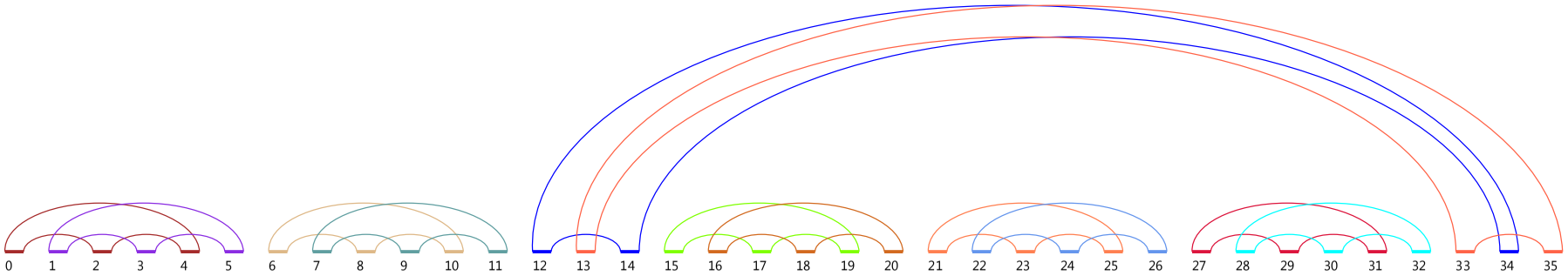

Figura I.2: Configuração problemática n° 2.

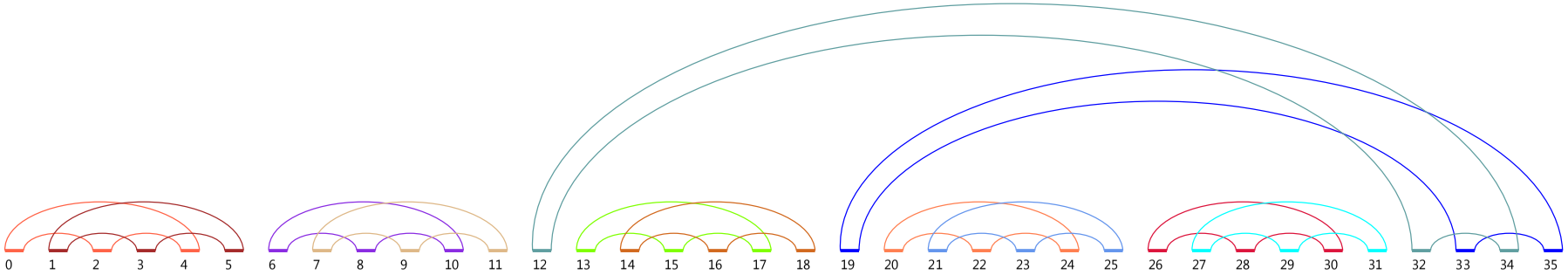

Figura I.3: Configuração problemática n^{o} 3.

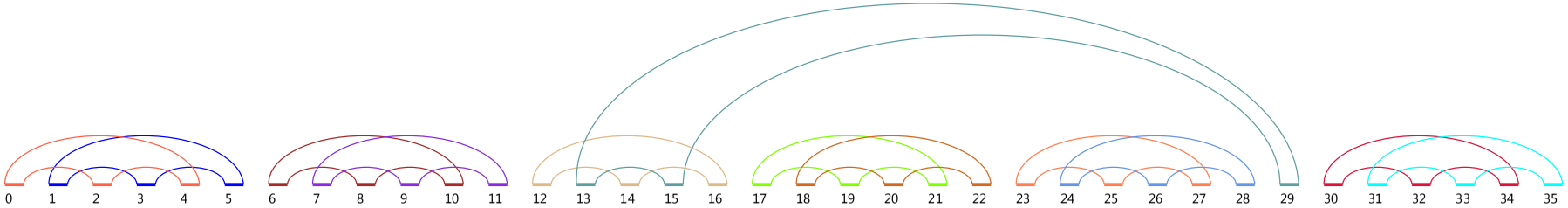

Figura I.4: Configuração problemática n^{o} 4.

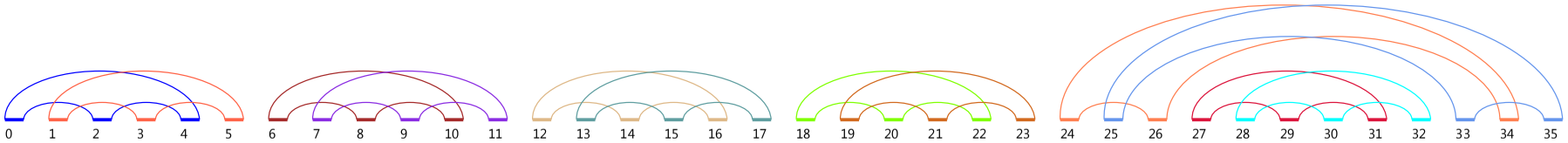

Figura I.5: Configuração problemática n° 5.

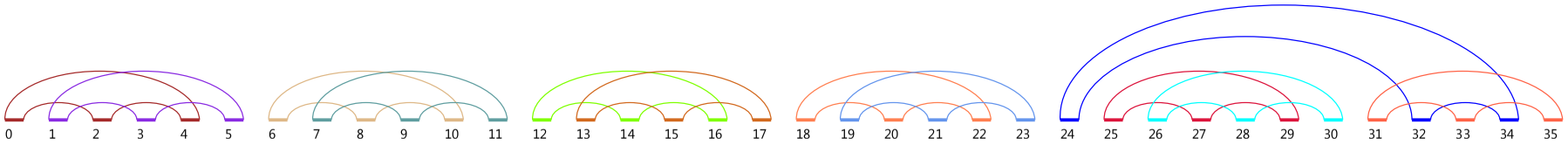

Figura I.6: Configuração problemática n° 6.

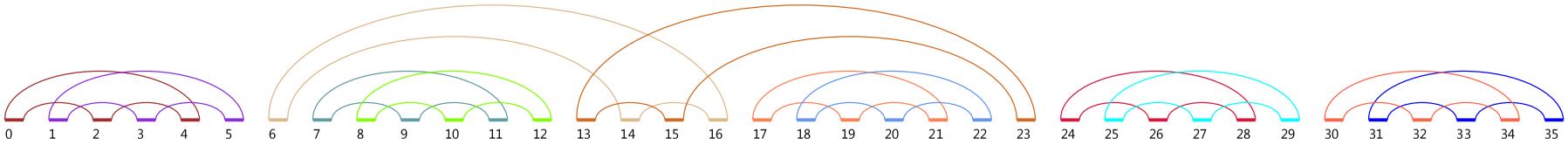

Figura I.7: Configuração problemática n° 7.

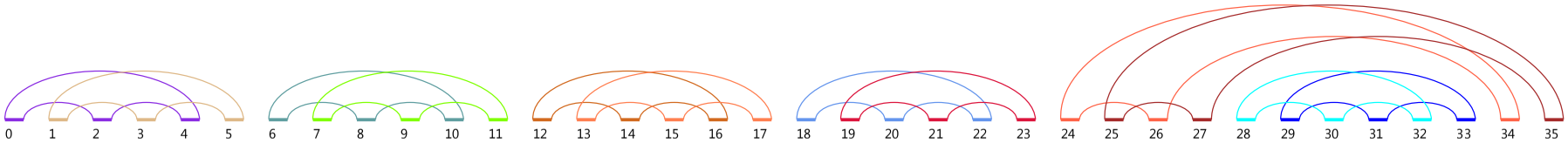

Figura I.8: Configuração problemática n° 8.

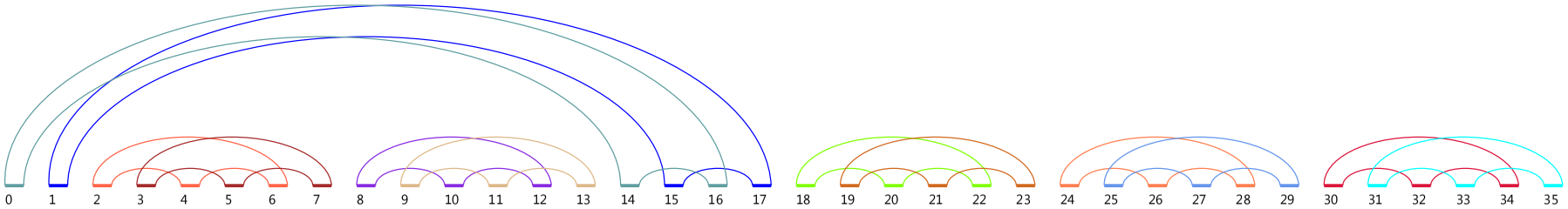

Figura I.9: Configuração problemática n^{o} 9.

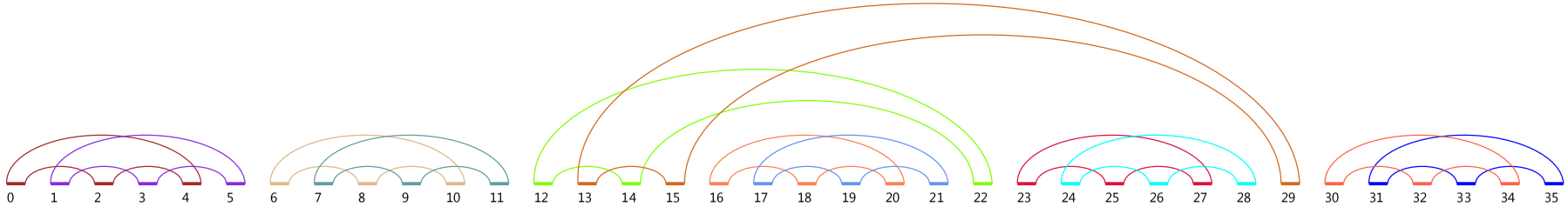

Figura I.10: Configuração problemática n^{o} 10.

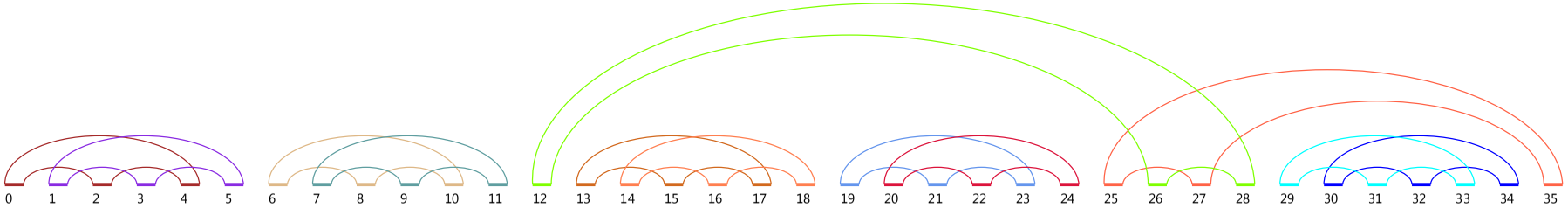

Figura I.11: Configuração problemática n^{o} 11.