COPPE UFRJ
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

REARRANJO DE GENOMAS: ALGORITMOS E COMPLEXIDADE

Luís Felipe Ignácio Cunha

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Rio de Janeiro
Março de 2017

REARRANJO DE GENOMAS: ALGORITMOS E COMPLEXIDADE

Luís Felipe Ignácio Cunha

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Profª. Celina Miraglia Herrera de Figueiredo, D.Sc.

Prof. Luis Antonio Brasil Kowada, D.Sc.

Prof. Franklin de Lima Marquezino, D.Sc.

Profª. Simone Dantas de Souza, D.Sc.

Prof. Fábio Protti, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2017

Cunha, Luís Felipe Ignácio

Rearranjo de Genomas: Algoritmos e Complexidade/Luís Felipe Ignácio Cunha. – Rio de Janeiro: UFRJ/COPPE, 2017.

XIII, 118 p.: il.; 29, 7cm.

Orientadores: Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 51 – 55.

1. Rearranjo de Genomas. 2. Distância de Transposição. 3. Diâmetro de Transposição. 4. Problemas de Centralidade. 5. Algoritmos aproximativos. 6. NP Completude. I. Figueiredo, Celina Miraglia Herrera de *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico a Eucinéa, Almir,
Almirzinho e Philippinho!

Agradecimentos

Agradeço a Deus pelo estado de graças que estou por ter chegado até aqui, pelo aprendizado e por ter colocado pessoas incríveis ao meu redor que me incentivam e me ajudam a seguir firme nesta trilha.

Dentre as diversas pessoas, primeiramente agradeço e dedico esta tese a minha mãe, meu pai, meu irmão e ao meu filho, Philippinho. Vocês fazem parte do alicerce do que sou hoje. Muito obrigado pelo carinho, toda a preocupação e conselhos. Meus pais sempre se esforçaram para nossa educação, minha mãe que tanto se sacrificou para nosso bem e também tanto se preocupa, meu pai que sempre apoiou ao lado da minha mãe no que foi necessário dentro de suas possibilidades, meu irmão sempre bastante atencioso com todos nós e Philippinho, meu filho querido, que é um grande amigo e conselheiro, mesmo com tão pouca idade. Difícil ser sucinto para expressar minha gratidão a vocês.

Agradeço demais aos meus amigos que, como dizem, são a família que nos permitiram escolher. Vocês são pessoas especiais para mim e sou grato pelos vários momentos únicos. Agradeço, em especial, a Fernanda e a Diana. Fernanda, sempre muito amiga, me ajuda e me ouve em tudo, muito obrigado! Na França, por exemplo, além de se preocupar, sempre me ajudava quando eu precisava falar mais do que um ça vá. Além disso, compartilhamos o gosto pelo pagode, erramos diversas letras de músicas e, no violão, é a pessoa que canta independente de eu estar tocando certo ou não. Com ela, meu dia fica bem melhor com todos os momentos bastante agradáveis. Diana é minha irmã mais velha, uma pessoa também bastante organizada e dedicada no que se propõe a fazer. Muito obrigado pelas várias conversas, pelo incentivo, torcida e paródias sobre mim. Ela é sempre muito animada, o que torna o trabalho muito agradável também. Aos outros amigos do LAC, muito obrigado. Não escreverei mais para não ultrapassar o tamanho das páginas da tese.

Durante o doutorado estive um período sanduíche na França, o que me proporcionou uma experiência muito boa. Tenho, portanto, muito a agradecer a todos que tornaram isso possível, em particular, à professora Simone Dantas pelo seu projeto e ao professor supervisor Anthony Labarre. Tenho a excelente oportunidade de poder trabalhar com várias pessoas e busco sempre absorver o melhor de cada um para desenvolver a minha própria personalidade em pesquisa. Agradeço em particular ao

Rodrigo Hausen, a pessoa que me apresentou o tema de pesquisa que sigo desde a iniciação científica. Além disso, continuamos a colaboração e espero que perdure.

Agradeço ao suporte financeiro para esse projeto que vem desde a graduação e continua até hoje. Contei e conto com auxílio dos órgãos de fomento FAPERJ, CAPES e CNPq, tanto por bolsas individuais quanto por projetos de pesquisa que, felizmente, posso integrar.

A pesquisa e as diversas oportunidades que tive, destaquei só algumas, foram possíveis graças aos meus orientadores professores Celina e Luis Antonio. Tenho muito a agradecer aos dois. Vocês confiaram e confiam em mim há alguns anos, se preocupam e se dedicam para minha evolução de pesquisador e de pessoa. Valeu muito a pena ter aceitado esse projeto e ter enfrentado todos os desafios propostos por vocês. Muitos outros estão por vir e isto é uma continuidade de tudo que passamos, enfrentamos e vencemos. Tive muita sorte então de ter sido (e ser) orientado por vocês: a professora Celina é sempre muito preocupada, dedicada e organizada com todos os seus filhos, e o professor Luis Antonio é praticamnete um amigo que trabalha junto com seus alunos. Muito obrigado.

Muito obrigado aos professores membros da banca de avaliação. Tenho profundo respeito a vocês e gratidão pelos comentários. Agradeço também a todos os professores que tive oportunidade de fazer disciplina e de acompanhar um pouco no cotidiano. Com certeza, a determinação e profissionalismo são qualidades que fazem o PESC ter o nível excelente de qualidade. Muito obrigado também, aos demais funcionários do PESC que são pessoas sempre prestativas, de bom humor e que nos proporcionam um bom ambiente de estudo e trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

REARRANJO DE GENOMAS: ALGORITMOS E COMPLEXIDADE

Luís Felipe Ignácio Cunha

Março/2017

Orientadores: Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Programa: Engenharia de Sistemas e Computação

Esta tese trata de rearranjo de genomas nos eventos de: transposição, pontos de quebra, movimento de blocos, movimento de blocos curtos, e de multi corte restritos. Abordamos os problemas de ordenação, permutação mais próxima, e de diâmetro. Apresentamos algoritmos aproximativos, NP-completudes e propriedades.

Sobre o problema de ordenação por transposições, provado ser NP-completo, alguns algoritmos aproximativos foram propostos baseados no grafo chamado diagrama de realidade e desejo. Através da análise dos ciclos deste grafo, propomos um novo algoritmo que atinge melhores resultados correntes, tanto de razão de aproximação de 1,375 quanto de complexidade de tempo de $O(n \log n)$.

Embora ordenação por transposições seja NP-completo, há outros problemas polinomiais ou em aberto. Nestes casos, surge o desafio de encontrar uma permutação que esteja a uma distância máxima limitada por algum valor em relação a um conjunto de permutações dadas de entrada. Este é o problema de encontrar a permutação mais próxima. Mostramos que, em relação às operações de pontos de quebra e de movimento de blocos, tais problemas são NP-completos.

Com o objetivo de obter propriedades sobre operações que restingem ou generalizam outras, tratamos da operação de movimento de blocos curtos e propomos a operação de multi corte restritos. Sobre movimento de blocos curtos, mostramos classes com distâncias exatas, propriedades sobre o grafo de permutação, e mostramos que o problema de permutação mais próxima é NP-completo. Sobre multi corte restritos, tratamos de duas variações: uma cujo número de blocos não reversíveis é limitado por constante, e outra cujo número de blocos não reversíveis é arbitrário. Mostramos limites justos de distância e de diâmetro para ambas as versões.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

GENOME REARRANGEMENTS: ALGORITHMS AND COMPLEXITY

Luís Felipe Ignácio Cunha

March/2017

Advisors: Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Department: Systems Engineering and Computer Science

This thesis discusses events of genome rearrangements problems: transposition, breakpoint, block interchange, short block move, and the restricted multi break. We consider problems of sorting, closest permutation, and the diameter. We develop approximation algorithms, NP-completeness and properties about these problems.

Regarding the sorting by transpositions, which is an NP-complete problem, several approximation algorithms were proposed based on the graph called the reality and desire diagram. Through a case analyses of the cycles of this graph, we propose a new one which achieves so far the best 1.375 ratio and $O(n \log n)$ running time complexity.

Although sorting by transpositions is NP-complete, there are several metrics whose sorting problems are polynomial or are open. In such cases, an interesting problem arises to find a permutation with maximum distance of an input permutation set at most some value, this is the closest permutation problem. We show that with respect to the polynomial distance problems of breakpoint and of block interchange, both problems are NP-complete.

In order to explore properties on operations that are restriction or generalization of others, we deal with the operation of short block move and we propose the operation of restricted multi break. Regarding the short block move, we show tractable classes of permutations, properties on the permutation graph, and we show that the closest permutation problem is NP-complete. Regarding the restricted multi break, we study two versions: one where the number of non reversible blocks is bounded by a constant, and another one whose number of non reversible blocks is arbitrary. We prove tight bounds on the distance and the diameter problems for both versions.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"Durante a evolução, moléculas de hereditariedade são costuradas, modificadas, cortadas, alongadas, encurtadas e revertidas."

(François Jacob, 1984)

Pelo princípio da evolução molecular, genomas dão origem a outros. Moléculas de DNA são responsáveis por toda informação genética dos seres vivos. Em proteínas, conteúdos de DNA são quase similares, porém suas organizações são bem diferentes, afetando assim suas funções. Um exemplo é o caso dos genomas do repolho e do nabo, que possuem genes quase idênticos [49]. Rearranjo de genomas é um conjunto de eventos mutacionais que afetam a organização do DNA. A distância de rearranjo é portanto, o número mínimo de eventos de rearranjo para transformar um genoma em outro.

Eventos de rearranjo envolvem modelos matemáticos que auxiliam a buscar o quão distante uma espécie está de outra, de modo que esta informação é usada para reconstrução de árvores filogenéticas. O problema de distância de rearranjo pode ser definido por determinar o menor número de operações necessárias para transformar uma permutação em outra (Figura 1.1), ou de maneira equivalente, determinar o menor número de operações para transformar uma permutação qualquer na permutação identidade, permutação onde todos os elementos estão em ordem crescente. Deste modo, genomas são definidos por permutações de inteiros distintos, onde cada inteiro representa um gene. Os livros de Fertin *et al.* [34] e de Setubal e Meidanis [52] abordam estes problemas, apresentando não somente os modelos de rearranjo de genomas com enfoques computacional e combinatório, como também suas implicações práticas.

Nas últimas décadas, houve um grande avanço no estudo de sequenciamento de DNA de espécies. Com isso, estudos foram propostos de modo a buscar novas técnicas computacionais que fossem capazes de analisar a grande quantidade de dados obtidos dessas sequências, um exemplo é a compreensão da comparação de

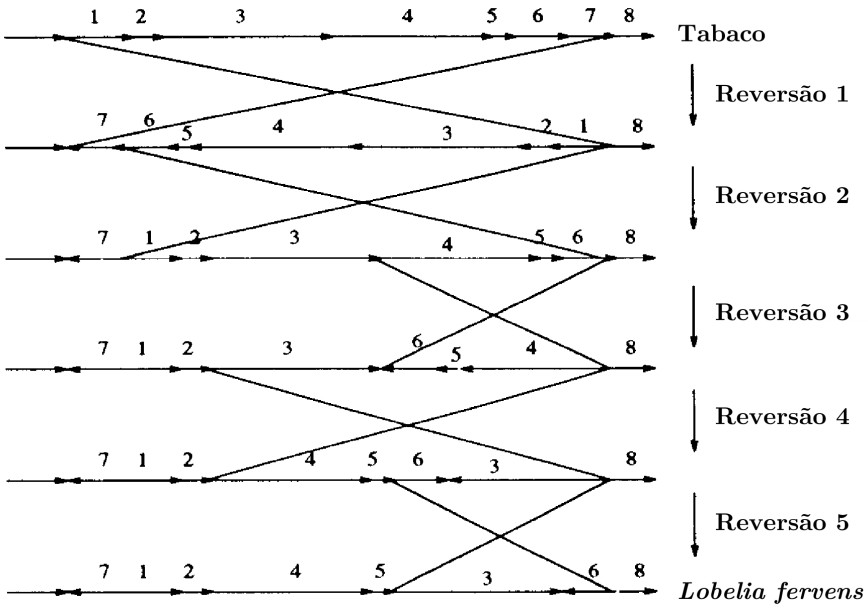

Figura 1.1: Evolução hipotética entre cloroplastos do Tabaco e *L. fervens* (adaptado de [4]).

genomas. Para que seja melhor entendida a filogenia dentro das hipóteses evolutivas, diversos modelos matemáticos foram elaborados motivadas por comparar distinções a partir de eventos mutacionais.

Nesta tese, tratamos de aspectos combinatórios e computacionais de alguns modelos de rearranjo em permutações. Destacaremos a seguir, um pouco da contextualização dos problemas abordados.

Um dos eventos mutacionais mais conhecidos e estudados pelo ponto de vista matemático é a *transposição*, que pode ser vista por uma operação que troca de posição dois blocos de elementos contíguos em uma permutação. O problema de *ordenação por transposições* consiste em determinar o menor número de transposições que transforma uma permutação na permutação identidade. Foi introduzido por Bafna e Pevzner [3] em 1998, e provado ser NP-completo somente em 2010, por Bulteau *et al.*, com publicação em 2012 [7]. Diversos trabalhos foram apresentados buscando limites para distâncias [3, 9, 30], algoritmos exatos para classes de permutações [17, 18, 29, 40, 43], algoritmos aproximativos, e até mesmo estruturas de dados foram propostas buscando melhores resultados de complexidade para algoritmos aproximativos [29, 33, 39]. Outro problema relacionado é determinar o *diâmetro*, em que busca-se encontrar os maiores valores de distância, foi considerado em [29, 30, 46, 48], e o melhor que se conhece são limites inferiores e superiores [18, 30].

Na dissertação de mestrado [11], tratamos de propriedades estruturais de clas-

ses de permutações, apresentamos limites justos e algoritmos exatos para classes de permutações, e abordamos também o diâmetro de transposição. Em relação ao diâmetro, elaboramos um estudo algébrico envolvendo união de permutações e obtivemos novos limites para distâncias, estabelecendo por consequência o atual limite inferior do diâmetro de transposição [14, 17, 18] e propusemos uma estratégia para melhorar este limite.

Nesta tese, tratamos de algoritmos aproximativos para o problema de ordenação por transposições. Hartman e Shamir [39] em 2006, propuseram um algoritmo de razão de aproximação de 1,5 e complexidade de tempo de $O(n^{\frac{3}{2}}\sqrt{\log n})$. Feng e Zhu [33] em 2007, elaboraram uma estrutura de dados para reduzir a complexidade de tempo do algoritmo de Hartman e Shamir para $O(n \log n)$. Elias e Hartman [29] em 2006, elaboraram um algoritmo de razão de aproximação de 1,375 e complexidade de tempo de $O(n^2)$. Firoz *et al.* [35] em 2011, afirmaram que a utilização da estrutura de dados de Feng e Zhu aplicada ao algoritmo de Elias e Hartman tornaria a complexidade de tempo deste algoritmo de $O(n \log n)$. Estudamos estas propostas e invalidamos a estratégia de Firoz *et al.*. Além disso, desenvolvemos um novo algoritmo que atinge a melhor razão de aproximação de 1,375 e a melhor complexidade de tempo de $O(n \log n)$.

Assim como o problema de ordenação por transposições, outros problemas de ordenação são também NP-completos, como: ordenação por reversões sem sinal [8], ordenação por reversões pré-fixadas [6], entre outros.

Temos porém alguns problemas polinomiais de distâncias, tais como: distância de pontos de quebra, ordenação por troca de blocos [9] (mesmo esta operação sendo uma generalização de transposição), e outros. Quando o problema de distância é polinomial, ou permanace em aberto, surge o novo desafio de determinar uma permutação que seja central em relação a um conjunto de outras permutações dadas. Este é o problema de determinar a *permutação mais próxima*, já sendo estudado considerando *cadeias de caracteres* ao invés de permutações, e em relação a distância de Hamming, onde foi provado ser NP-completo [45].

Nesta tese, tratamos deste problema de centralidade envolvendo permutações em relação as distâncias polinomiais de pontos de quebra e de movimentos de blocos. Consideramos também em relação a operação de movimentos de blocos curtos, cuja complexidade de ordenação permanece em aberto. Por outro lado, provamos que para cada uma dessas métricas, determinar a permutação mais próxima é NP-completo.

Tratamos também de outros problemas de ordenação que possuem relação com transposições. Propomos a operação de *multi corte restrito*, operação esta que generaliza reversões, transposições, movimentos de blocos, e ainda assim é uma restrição da operação proposta por Alekseyev e Pevzner [1] de multi corte.

Apresentamos propriedades estruturais e relações entre valores no número de blocos não-reversíveis por operação, e mostramos classes polinomiais de distâncias e limites para distâncias e para diâmetros.

A restrição mais natural de uma transposição é a operação em que dois elementos são afetados na permutação, i.e. troca de dois elementos consecutivos. Este problema de ordenação pode ser facilmente resolvido pelo algoritmo *bubble sort*. Existe uma outra restrição da operação de transposição, mais desafiadora, chamada de *movimentos de blocos curtos*, é uma transposição em que no máximo três elementos são afetados em cada operação. Este problema foi proposto por Heath e Vergara [41], e a complexidade computacional do problema de ordenação permanece em aberto.

Apresentamos classes de permutações não reduzidas, tratáveis no problema de transposições que continuam tratáveis para movimentos de blocos curtos. De forma que algumas classes possuem o mesmo valor de distância de transposição e de movimentos de blocos curtos. Identificamos classes de equivalências tais que permutações na mesma classe possuem a mesma distância. Mostramos uma condição suficiente para ordenar otimamente uma permutação, condição esta que não é válida para outros movimentos de blocos limitados por valores maiores do que 3. Além de mostrar que o problema de determinar a permutação mais próxima é NP-completo, mesmo sob esta operação.

Os modelos tratados nesta tese são baseados em genomas lineares em um único cromossomo. Outros modelos são considerados em diversos outros documentos sobre rearranjo de genomas, onde são tratados genomas tanto lineares quanto circulares, e também considerando genomas num único cromossomo ou não. Recomendamos Feijão [31] e Tannier [53] para detalhes sobre outros modelos.

1.1 Organização do trabalho

Apresentamos nesta tese os principais resultados desenvolvidos ao longo do doutorado. O Capítulo 2 com periódico correspondente no Anexo A, o Capítulo 3 com manuscrito no Anexo B, e o Capítulo 4 com manuscritos correspondentes nos Anexos C e D, respectivamente.

Esta tese está organizada da seguinte forma: no decorrer deste capítulo apresentamos as definições dos problemas, resultados preliminares sobre os problemas desenvolvidos na tese; no Capítulo 2 abordamos o problema de ordenação por transposições, onde apresentamos propriedades, estratégias conhecidas para algoritmos aproximativos, invalidamos estratégias propostas na literatura, e desenvolvemos novas propostas de forma a obter por fim um algoritmo aproximativo com melhores razão de aproximação e complexidade; no Capítulo 3 tratamos dos problemas de cadeias de caracteres e de permutações mais próximas, mostramos NP-completudes

do problema de permutações mais próximas em relação às métricas de distâncias de pontos de quebra, de movimentos de blocos e de movimentos de blocos curtos, e discutimos sua relação com o problema da mediana; no Capítulo 4 abordamos os problemas de ordenação por movimentos de blocos curtos e por multi corte restrito, descrevemos propriedades e demais resultados em relação a cada uma das operações, e também concluímos a tese apresentando caminhos futuros a serem investigados em relação a cada um dos problemas estudados.

Durante a dissertação de mestrado, obtivemos publicação no *SIAM Journal on Discrete Mathematics*, 2013 [18]. Além disso, apresentamos no *Brazilian Symposium on Bioinformatics*, 2012 [17].

Os resultados desenvolvidos nesta tese de doutorado possuem manuscritos anexados no fim deste trabalho. Referências anexadas:

- Anexo A "A faster 1.375-approximation algorithm for sorting by transpositions" [21], publicado no *Journal of Computational Biology*, 2015. Este trabalho obteve também publicações nas conferências:
  - Brazilian Symposium on Bioinformatics 2013 [19];
  - Workshop on Algorithms in Bioinformatics 2014 [20].
- Anexo B "On the computational complexity of closest string problems" [22], a ser submetido para o *Discrete Applied Mathematics*. Este trabalho obteve também publicações nas conferências:
  - Latin-Iberoamerican Conference on Operations Research 2016 [26];
  - Latin American Workshop on Cliques in Graphs 2016 [27].
- Anexo C "Sorting by short block-moves and the complexity of the short block-move closest permutation problem" [15], a ser submetido para o *SIAM Journal on Discrete Mathematics*. Este trabalho obteve também publicação na conferência:
  - International Colloquium on Graph Theory and Combinatorics 2014 [23].
- Anexo D "On sorting permutations by restricted multi-break rearrangements" [16], a ser submetido para o *SIAM Journal on Discrete Mathematics*. Este trabalho obteve também publicações nas conferências:
  - Latin American Workshop on Cliques in Graphs 2014 [25];
  - Cologne-Twente Workshop on Graphs & Combinatorial Optimization 2015 [24].

Em todos os capítulos, apresentamos os principais resultados dos assuntos, e algumas demonstrações são apresentadas como "Esquema de demonstração". Para detalhes completos sugerimos consultar referência anexada correspondente ao capítulo.

1.2 Preliminares

O modelo de rearranjo entre genes deve obedecer algumas hipóteses a respeito dos genomas:

1. É conhecida a ordem dos genes de cada genoma;
2. Todos os genomas compartilham o mesmo conjunto de genes;
3. Todos os genomas contém uma única cópia de cada gene;
4. Todos os genomas possuem um único cromossomo.

A partir destas hipóteses, podemos tratar genomas por permutações pertencentes ao grupo simétrico[1] S_n. Assim, temos as seguintes correspondências em relação às hipóteses anteriores:

1. Cada permutação é definida por uma função bijetiva;
2. Caso estejamos transformando uma permutação π em outra σ, então π e σ pertencem a S_n, ou seja, ambas possuem o mesmo número de elementos;
3. Toda permutação $\pi \in S_n$ contém n elementos distintos;
4. Se π contém n elementos distintos, então π é uma permutação do grupo simétrico S_n.

A partir disto, descrevemos formalmente uma permutação.

Definição 1.1 (Permutação) *Uma* permutação $\pi_{[n]} = [\boldsymbol{\pi_0}\, \pi_1\, \pi_2 \cdots \pi_n \boldsymbol{\pi_{n+1}}]$ *é uma função bijetiva com domínio no conjunto* $\{1, 2, \cdots, n\}$ *e imagem em* $\{1, 2, \cdots, n\}$*, tal que* $\pi_0 = 0$ *e* $\pi_{n+1} = n + 1$*. Ou seja,* $\pi(i) = \pi_i, \forall i = 1, 2, \cdots, n$ *e* $\pi(i) = \pi(j)$*, se e somente se,* $i = j$*.*

O *tamanho* de uma permutação $\pi_{[n]}$ é definido pelo número de elementos que existem em $\pi_{[n]}$, ou seja, $\pi_{[n]}$ possui tamanho n. Quando não causar ambiguidade, escreveremos π para uma permutação pertencente a S_n omitindo o índice $\pi_{[n]}$.

[1] Para o leitor interessado em conceitos de Teoria dos Grupos e trabalhos que tratam permutações no contexto de estruturas algébricas abstratas, recomendamos livros de Gonçalves [36] e de MacLane e Birkhoff [47].

Algumas permutações recebem nomes e símbolos especiais, como: *permutação identidade*, $\iota_{[n]} = [\mathbf{0}\,1\,2\,\cdots\,n\,\boldsymbol{n}\mathbf{+1}]$; *permutação reversa*, $\rho_{[n]} = [\mathbf{0}\,n\,n{-}1\,\cdots\,1\,\boldsymbol{n}\mathbf{+1}]$. Outra que merece destaque é permutação inversa. Seja $\pi = [\pi_1\,\pi_2\,\cdots\,\pi_n]$, temos que $\pi^{-1} = [\mathbf{0}\,\pi_1^{-1}\,\pi_2^{-1}\,\cdots\,\pi_n^{-1}\,\boldsymbol{n}\mathbf{+1}]$ é a *permutação inversa* de π, onde $\pi(\pi^{-1}(i)) = i$.

Dada a permutação $\pi = [\mathbf{0}\ 8\ 5\ 2\ 10\ 7\ 4\ 1\ 9\ 6\ 3\,\mathbf{11}]$, temos que sua permutação inversa é $\pi^{-1} = [\mathbf{0}\ 7\ 3\ 10\ 6\ 2\ 9\ 5\ 1\ 8\ 4\ \mathbf{11}]$, isto devido $\pi(\pi^{-1}(1)) = \pi(7) = 1$, $\pi(\pi^{-1}(2)) = \pi(3) = 2$, e assim por diante.

Operações em permutações Modelos de rearranjo descrevem mutações em genomas, as mutações são vistas por operações a serem aplicadas em permutações. A seguir, descrevemos algumas destas operações trabalhadas ao longo desta tese, todas estas operações são elementos do conjunto gerador de S_n.

O *produto* entre duas permutações π e σ para as duas de tamanho n é a permutação $\pi\sigma$ obtida pela composição $\pi(\sigma(i))$, para todo $i \in \{1, 2, \ldots, n\}$.

Transposição, proposta em 1998 por Bafna e Pevzner [3], $t(i, j, k)$, onde $1 \leq i < j < k \leq n + 1$ é a permutação:

$$t(i, j, k) = [\mathbf{0}\,1\,2\,\cdots\,i{-}1\ j\ j{+}1\,\cdots\,k{-}1\,i\,\cdots\,j{-}1\,k\,\cdots\,n\,\boldsymbol{n}+\mathbf{1}].$$

O produto $\pi t(i, j, k)$ é visto como uma operação em π que troca o bloco entre as posições i e $j - 1$ com o bloco entre as posições j e $k - 1$, ou seja:

$$\pi t(i, j, k) = [\mathbf{0}\,\pi_1\,\pi_2\,\cdots\,\pi_{i-1}\boxed{\pi_j\,\cdots\,\pi_{k-1}}\boxed{\pi_i\,\cdots\,\pi_{j-1}}\pi_k\,\cdots\,\pi_n\,\boldsymbol{n}+\mathbf{1}].$$

Seja $\pi = [\mathbf{0}\,8\,5\,2\,10\,7\,4\,1\,9\,6\,3\,\mathbf{11}]$ uma permutação de S_{10}. Ao aplicarmos $t(3, 7, 9)$ em π obtemos: $\pi t(3, 7, 9) = [\mathbf{0}\,8\,5\,1\,9\,2\,10\,7\,4\,6\,3\,\mathbf{11}]$.

Movimento de blocos, proposto em 1999 por Christie [9], $b(i, j, k, m)$, onde $1 \leq i < j \leq k < m \leq n + 1$ é a permutação:

$$b(i, j, k, m) = [\mathbf{0}\,1\,2\,\cdots\,i-1\boxed{k\,\cdots\,m-1}\,j\,\cdots\,k-1\boxed{i\,\cdots\,j-1}\,m\,\cdots\,n\,\boldsymbol{n}+\mathbf{1}].$$

O produto $\pi b(i, j, k, m)$ é visto como uma operação em π que troca o bloco entre as posições i e $j - 1$ com o bloco entre as posições k e $m - 1$, ou seja:

$$\pi b(i, j, k, m) = [\mathbf{0}\,\pi_1\,\pi_2\,\cdots\,\pi_{i-1}\boxed{\pi_k\,\cdots\,\pi_{m-1}}\,\pi_j\,\cdots\,\pi_{k-1}\boxed{\pi_i\,\cdots\,\pi_{j-1}}\,\pi_m\,\cdots\,\pi_n\,\boldsymbol{n}\mathbf{+1}].$$

Note que uma transposição pode ser vista pelo movimento de blocos $b(i, j, j, m)$.

Movimento de blocos curtos, proposto em 1998 por Heath e Vergara [41], $s(i, j, k)$, onde $1 \leq i < j \leq k \leq n + 1$ é a transposição $t(i, j, k)$ tal que $k - i \leq 3$.

Se um movimento de blocos curtos é a transposição $t(i, i+1, i+2)$, então chama-

mos esta operação de *pequeno salto* (do inglês *skip*), se é a transposição $t(i, i{+}1, i{+}3)$, ou $t(i, i+2, i+3)$, então chamamos de *pulo* (do inglês *hop*).

Movimento de multi corte restrito, proposta nesta tese, é a operação $\varpi(a, b; c_1 \leftrightarrow d_1; c_2 \leftrightarrow d_2; \ldots; c_k \leftrightarrow d_k)$, onde $1 \leq a \leq c_1 \leq d_1 \leq c_2 \leq d_2 \leq \ldots \leq c_k \leq d_k \leq b \leq n$, que aplicada em π inverte o intervalo de π definido pelas posições a e b, exceto pelos elementos nos *blocos não reversíveis*, blocos definidos pelos pares (c_i, d_i) para $0 \leq i \leq k$, transformando a permutação π na permutação $\pi\varpi$:

$$[\mathbf{0}\pi_1 \cdots \pi_{a-1} \boxed{\pi_b \cdots \pi_{d_k+1} \underbrace{\boxed{\pi_{c_k} \cdots \pi_{d_k}} \pi_{c_k-1} \cdots \pi_{d_1+1} \boxed{\pi_{c_1} \cdots \pi_{d_1}}}_{k \text{ blocos não reversíveis}} \pi_{c_1-1} \cdots \pi_a} \pi_{b+1} \cdots \pi_n \boldsymbol{n+1}],$$

Note que um movimento de multi corte restrito é uma operação que generaliza todas as demais vistas anteriormente, e também a clássica operação de *reversão* (operação que inverte um bloco de elementos na permutação, proposta em 1993 por Bafna e Pevzner [4]).

Operações de transposição $t(i, j, k)$, movimento de blocos $b(i, j, k, t)$, movimento de blocos curtos $s(i, j, i+2)$ ou $s(i, j, i+3)$, e a reversão $r(i, j)$, onde $1 \leq i < j < k < t \leq n+1$, são multi corte restritos:

- *transposição*, troca de dois blocos de elementos consecutivos, $t(a, d{+}1, b{+}1) = \varpi(a, b; a \leftrightarrow d; d{+}1 \leftrightarrow b)$;
- *movimento de blocos*, troca de dois blocos de elementos (não necessariamente consecutivos), $b(a, d_1, c_2{+}1, b) = \varpi(a, b; a \leftrightarrow d_1; d_1{+}1 \leftrightarrow c_2; c_2{+}1 \leftrightarrow b)$;
- *movimento de blocos curtos*, transposição tal que o número de elementos nos dois blocos é no máximo 3, $s(a, b, a+2) = \varpi(a, a+1)$, ou $s(a, b, a+3) = \varpi(a, a+2; a \leftrightarrow b-1; b \leftrightarrow a+2)$;
- *reversão*, inverte um intervalo, $r(a, b) = \varpi(a, b)$ corresponde a $k = 0$.

Um *k-multi corte restrito* é uma operação que inverte um intervalo que contém no máximo k blocos de elementos não reversíveis. Denotamos ϖ para o caso em que k é arbitrário, ou seja $k = n$, e por $k\varpi$ para um k-multi corte restrito, um caso em que k é algum valor fixo.

Ordenação de permutações A partir de uma determinada operação, temos então o problema de calcular o número mínimo de operações que transforme uma permutação em outra.

Definição 1.2 (Distância) *Seja P uma operação entre permutações. A* distância *$d_P(\pi, \sigma)$ de uma permutação π a uma permutação σ, em que π e σ possuem o mesmo*

tamanho, é a quantidade mínima q de operações P da sequência P_1, P_2, $\cdots$, P_q, tal que $\pi P_1 P_2 \cdots P_q = \sigma$. Se $\pi = \sigma$, então $d_P(\pi, \sigma) = 0$.

É possível provar que dadas as permutações π, σ e γ, temos que $d_P(\pi, \sigma) = d_P(\gamma\pi, \gamma\sigma)$. Para isso, considere $\pi P_1 P_2 \cdots P_q = \sigma$ e o produto $\gamma\pi P_1 P_2 \cdots P_q$. Pela associatividade do produto entre elementos de S_n, é imediato verificar que $\gamma(\pi P_1 P_2 \cdots P_q) = \gamma\sigma$.

Sabendo então que $d_P(\pi, \sigma) = d_P(\gamma\pi, \gamma\sigma)$, ao considerar $\gamma = \sigma^{-1}$ teremos $d_P(\pi, \sigma) = d_P(\sigma^{-1}\pi, \iota)$. Assim, o problema de *Distância*, Definição 1.2, é equivalente a determinar a quantidade mínima q de operações P da sequência P_1, P_2, $\cdots$, P_q, tal que $\pi P_1 P_2 \cdots P_q = \iota$. Com isso, a distância de uma permutação π é computada em relação à ι, e escrevemos $d_P(\pi, \iota) = d(\pi)$. É por esta razão que o problema de distância é também conhecido como o problema de *ordenação*. Assim, nesta tese, quando nos referirmos a *ordenar* uma permutação estamos preocupados em obter o valor da *sequência mínima de operações* que ordena a permutação.

A partir da relação $d_P(\pi, \sigma) = d_P(\gamma\pi, \gamma\sigma)$, também é imediado verificarmos que $d_P(\pi) = d_P(\pi^{-1})$. Isto devido $d_P(\pi, \iota) = d_P(\pi^{-1}\pi, \pi^{-1}) = d_P(\iota, \pi^{-1})$.

Com isso, temos a formulação do problema de decisão de ordenação de permutações:

ORDENAÇÃO DE PERMUTAÇÕES PELA OPERAÇÃO P
ENTRADA: Uma operação de permutações P, uma permutação π de tamanho n e um inteiro d.
PERGUNTA: A distância da permutação π pela operação P é $d_P(\pi) \leq d$?

Complexidade dos problemas de ordenação Diversos artigos tratam dos problemas de ordenação mencionados neste tese. O interesse computacional é uma das maiores motivações para estes estudos. Temos que:

- Ordenação por transposições foi provado ser NP-completo por Bulteau *et al.* [7] em 2012;
- Ordenação por movimento de blocos foi provado ser polinomial por Christie [9] em 1999;
- Ordenação por movimentos de blocos curtos permanece em aberto. Note que esta operação é uma generalização natural da operação em que a soma dos blocos é exatamente 2. Este último problema de ordenação pode ser resolvido em tempo polinomial pelo *Bubble sort*. Observe que a complexidade dos problema de: ordenação pela operação em que a soma dos blocos é exatamente 2

é polinomial; a operação de transposição – que generaliza esta última – é NP-completo; e a operação de movimento de blocos – que generaliza transposição – é polinomial. Evidenciando então que não existe monotonicidade entre os problemas de ordenação;

- Ordenação por reversões foi provado ser NP-completo por Caprara [8] em 1997. Este problema pode ser tratado em relação a permutações com sinal, de forma que assim que uma reversão for aplicada, os elementos no intervalo alternam sinais entre + e -. O problema de ordenação de reversões com sinal foi provado polinomial por Hannenhalli e Pevzner [38];

- Ordenação por multi-corte restritos é um problema em aberto, mesmo sendo um problema em que uma operação generaliza as demais vistas previamente.

Outros modelos de rearranjo surgiram de forma a considerar genomas com mais de um cromossomo linear ou circular, e também outras operações associadas a mutações. Estes modelos implicam em cadeias de caracteres lineares ou circulares, e em operações a serem aplicadas em tais cadeias. Um estudo mais detalhado pode ser visto no livro de Fertin et al. [34] e em artigos, como Feijão e Meidanis [32], e Yancopoulos et al. [54].

1.3 Limites de distância e de diâmetro

Distância de transposição e distância de movimento de blocos Os resultados envolvendo algoritmos aproximativos para distância de transposição encontram-se no Capítulo 2, e manuscrito referente a este tema de pesquisa no Anexo A.

Limites não triviais para a distância de transposição foram obtidos a partir do diagrama de realidade e desejo [3], grafo este que representa as adjacências da permutação e as adjacências da identidade. Dada uma permutação π, o *diagrama de realidade e desejo* de π é $G(\pi) = (V, R \cup D)$. O conjunto de vértices é $V = \{0, -1, +1, -2, +2, \ldots, -n, +n, -(n+1)\}$, e o conjunto de arestas é particionado em dois subconjuntos, as *arestas de realidade* direcionadas $R = \{\overrightarrow{i} = (+\pi_i,\ -\pi_{i+1}) \mid i = 0, \ldots, n\}$ e as *arestas de desejo* não direcionadas $D = \{(+i,\ -(i+1)) \mid i = 0, \ldots, n\})$. A Figura 2.1 ilustra $G([\mathbf{0}\,10\,9\,8\,7\,1\,6\,11\,5\,4\,3\,2\,\mathbf{12}])$, onde as setas representam as arestas direcionadas de R e os arcos as arestas não direcionadas de D.

Como cada vértice de $G(\pi)$ possui grau 2, o grafo pode ser particionado em ciclos disjuntos. Quando não causar ambiguidade, nos referimos a um *ciclo de* π por um *ciclo de* $G(\pi)$. Um ciclo de π possui tamanho ℓ, dito um ℓ-*ciclo*, se possui exatamente ℓ arestas de realidade. Uma permutação π é uma *permutação simples* se todo ciclo de π possui tamanho no máximo 3. Conforme exemplo da Figura 2.1, $\pi = [\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}]$ é uma permutação simples.

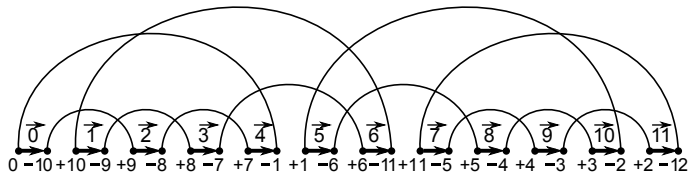

Figura 1.2: $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$, grafo com quatro 3-ciclos.

Bafna e Pevzner [3] mostraram que após aplicar uma transposição t, o número de ciclos de tamanhos ímpares de π, denotado $c_{odd}(\pi)$, é alterado de tal forma que $c_{odd}(\pi t) = c_{odd}(\pi) + x$, para $x \in \{-2, 0, 2\}$. Portanto, uma transposição t é classificada como um *x-movimento de π*. Como $c_{odd}(\iota) = n + 1$, temos o seguinte limite inferior da distância de transposição.

Teorema 1.3 *[3] Dada uma permutação π de tamanho n, $d_t(\pi) \geq \left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil$.*

A igualdade da distância ao limite inferior segue se, e somente se, for possível ordenar π somente por 2-movimentos.

Hannenhalli e Pevzner [38] provaram que toda permutação π pode ser transformada, com complexidade de tempo de $O(n)$, em uma permutação simples $\hat{\pi}$, pela inserção de novos elementos em posições apropriadas de π, de modo a preservar o limite inferior da distância, $\left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil = \left\lceil \frac{(m+1)-c_{odd}(\hat{\pi})}{2} \right\rceil$, onde m é tal que $\hat{\pi} = [\mathbf{0}\hat{\pi}_1 \ldots \hat{\pi}_m \boldsymbol{m+1}]$. Além disso, em qualquer sequência que ordena $\hat{\pi}$, cada transposição pode ser transformada em outra (com complexidade de tempo de $O(\log n)$ [33]), mantendo o mesmo número de transposições que também ordena π, implicando que $d_t(\pi) \leq d_t(\hat{\pi})$. Portanto, a estratégia de obter uma sequência que ordena π através de uma sequência que ordena $\hat{\pi}$ é comumente usada em algoritmos aproximativos de ordenação por transposições [29, 39].

Uma transposição $t(i, j, k)$ *afeta* um ciclo C se o ciclo contém uma das seguintes arestas de realidade: $\overrightarrow{i+1}$, ou $\overrightarrow{j+1}$, ou $\overrightarrow{k+1}$. Um ciclo é *orientado* se existe um 2-movimento que afeta este ciclo, caso contrário este ciclo é *não orientado*. Se π contém um ciclo orientado, então π é *orientada*, caso contrário π é *não orientada*.

Uma sequência de q transposições de exatamente r transposições 2-movimento é uma *(q, r)-sequência*. Uma $\frac{q}{r}$-sequência é uma (x, y)-sequência tal que $x \leq q$ e $\frac{x}{y} \leq \frac{q}{r}$.

Todas as definições vistas até agora relacionadas à transposição podem ser aplicadas a uma operação de movimento de blocos. Porém, uma operação de movimento de blocos é definida ser do tipo x-movimento se altera o número de ciclos, e não somente os ciclos ímpares, de tal forma que $x \in \{-2, 0, +2\}$. Christie [9] mostrou que sempre é possível encontrar um 2-movimento. Com isso, temos o valor da distância de movimentos de blocos de uma permutação.

Teorema 1.4 *[9] Dada uma permutação π de tamanho n, $d_b = \left\lceil \frac{(n+1)-c(\pi)}{2} \right\rceil$.*

Distância de movimento de blocos curtos Os resultados sobre distância de movimento de blocos curtos encontram-se no Capítulo 4, e manuscrito referente a este tema de pesquisa no Anexo C.

Heath e Vergara [41] encontraram limites para distância de movimentos de blocos curtos pelo uso do *grafo de permutação* $PG(\pi_{[n]}) = (V_\pi^p, E_\pi^p)$, tal que $V_\pi^p = \{1,\ 2,\ \ldots, n\}$ e $E_\pi^p = \{(i,j) \mid \pi_i > \pi_j,\ i < j\}$, cada aresta de PG é chamada de uma *inversão* em π. Denotamos *conexidade de uma permutação* π ao referir à conexidade do grafo de permutação de π.

Heath e Vergara provaram que sempre existe uma sequência ótima que ordena $\pi_{[n]}$, tal que cada movimento de blocos curtos diminui o número de inversões em pelo menos uma unidade, e no máximo duas unidades, portanto:

$$\left\lceil \frac{|E_\pi^p|}{2} \right\rceil \leq d_{sbm}(\pi_{[n]}) \leq |E_\pi^p|. \tag{1.1}$$

Nosso objetivo é, portanto, minimizar o número de operações que diminuem somente uma inversão de PG. Exemplos de permutações que possuem distância iguais aos limites inferior e superior são [**0** 2 4 3 5 1 **6**] e [**0** 2 1 4 3 6 5 **7**], respectivamente.

Heath e Vergara [42] definiram outro grafo para obter novos limites para a distância de movimentos de blocos curtos. O *grafo arco* $A_\pi^a = (V_\pi^a, E_\pi^a)$ de π é um grafo não direcionado com conjunto de vértices $V_\pi^a = E_\pi^p$, e dois arcos $(a,b),(c,d)$ são adjacentes se existe uma das duas opções:

i) $a = c$, e em π, entre b e d não existe nenhum elemento x tal que $b < x < d$; ou

ii) $b = d$, e em π, entre a e c não existe nenhum elemento x tal que $a < x < c$.

Segue da definição que A_π^a é um subgrafo do grafo linha de $PG(\pi_{[n]})$.

Heath e Vergara [42] observaram que um pulo é representado por uma aresta do grafo arco. Portanto, um emparelhamento máximo M do grafo arco gera um número mínimo de pulos a serem aplicados em π, enquanto os vértices que não são emparelhados U representam o número de pequenos saltos, com isso:

$$d_{sbm}(\pi_{[n]}) \geq |M| + |U|. \tag{1.2}$$

O limite obtido da Equação (1.2) é, de forma geral, maior que o limite da Equação (1.1). Se existe um emparelhamento perfeito em A_π^a, então os dois limites são iguais. Heath e Vergara [42] mostraram uma classe de permutações em que a Equação (1.2) gera uma sequência ótima de operações por movimentos de blocos curtos.

Exemplo 1.5 *A Figura 1.3 ilustra o grafo arco de* [**0** 2 4 6 8 1 3 5 7 **9**]. *Observe que não existe nenhuma aresta* $\{(8,1),(8,5)\}$, *já que o elemento* 3 *satisfaz* $1 < 3 < 5$ *e está entre* 1 *e* 5 *na permutação.*

Obtemos a sequência de movimentos de blocos curtos que ordena a permutação através das arestas "grossas" que formam o emparelhamento máximo $M = \{\{(6,1),(8,1)\},\{(6,3),(8,3)\},\{(6,5),(8,5)\},\{(2,1),(4,1)\}\}$*, o conjunto dos vértices que não são emparelhados é* $U = \{(4,3),(8,7)\}$*. A primeira operação é tomada pela busca do par mais à direita em* M *ou vértices em* U*, por exemplo tomamos* $(6,1),(8,1)$ *e aplicamos o pulo* $[\mathbf{0}\ 2\ 4\ \underline{6\ 8}\ \underline{1}\ 3\ 5\ 7\ \mathbf{9}] \rightarrow [\mathbf{0}\ 2\ 4\ 1\ 6\ 8\ 3\ 5\ 7\ \mathbf{9}]$*. A partir desta nova permutação, continuamos a busca em* M *e em* U*.*

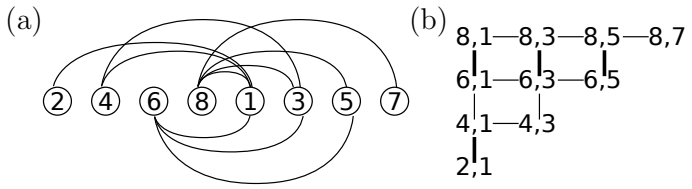

Figura 1.3: (a) $PG([\mathbf{0}\,2\,4\,6\,8\,1\,3\,5\,7\,\mathbf{9}])$ (b) $A^a_{[\mathbf{0}\,2\,4\,6\,8\,1\,3\,5\,7\,\mathbf{9}]}$.

Distância de multi corte restrito Os resultados sobre distância de multi corte restrito, além dos apresentados a seguir, encontram-se no Capítulo 4, e manuscrito referente a este tema de pesquisa no Anexo D.

Os limites mais naturais envolvendo problemas de distâncias são obtidos pelo uso do *ponto de quebra* de uma permutação. Uma *adjacência* (respectivamente uma *adjacência reversa*) numa permutação π é um par (π_i, π_{i+1}) para $0 \leq i \leq n$ tal que $\pi_{i+1} = \pi_i + 1$ (respectivamente $\pi_{i+1} = \pi_i - 1$). Se este par não é uma adjacência nem uma adjacência reversa, i.e. $|\pi_i - \pi_{i+1}| \neq 1$, então o par (π_i, π_{i+1}) é chamado de um *ponto de quebra*, e denotamos por $b(\pi)$ o número de pontos de quebra de π.

Chamamos de *tira crescente* de uma permutação π uma sequência maximal de elementos adjacentes. A *permutação reduzida* $gl(\pi)$ é a permutação obtida de π pela remoção da primeira tira crescente, se esta tira começa com o elemento 1, substituindo todas as outras tiras crescentes pelo menor valor da tira, e renumerando os s elementos restantes da permutação resultante π' de modo a obter uma permutação em relação ao conjunto $\{1, 2, \ldots, s\}$, em que os elementos são ordenados da mesma forma que em π' [9].

Por exemplo, se $\alpha = [\mathbf{0} \bullet 6\,5\,4 \bullet 1\,2\,3 \bullet 10\,9\,8\,7 \bullet \mathbf{11}]$, então $gl(\alpha) = [\mathbf{0} \bullet 4\,3\,2\,1 \bullet 8\,7\,6\,5 \bullet \mathbf{9}]$, onde $\bullet$ indica um ponto de quebra. Uma consequência imediata da definição é que $b(gl(\pi)) \leq b(\pi)$. Christie [9] mostrou que a operação de redução preserva a distância de movimentos de blocos e a distância de transposição. A mesma propriedade vale para a distância de ϖ.

Proposição 1.6 *Dada uma permutação* π*, temos que* $d_\varpi(\pi) = d_\varpi(gl(\pi))$*.*

Apesar desta igualdade valer para movimentos de blocos, transposição e multicorte restrito, o mesmo não vale para movimentos de blocos restritos. Outra

operação em que a igualdade de distâncias entre uma permutação e sua reduzida não é válida é para operação $k\varpi$, tal que k é algum inteiro fixo. Considere, por exemplo, a distância de reversão, $d_{0\varpi}([\mathbf{0}\,2\,3\,1\,\mathbf{4}]) = 2$, enquanto $d_{0\varpi}([\mathbf{0}\,2\,1\,\mathbf{3}]) = 1$.

Isto não contradiz a Proposição 1.6, de modo que a distância por multi corte restrito ϖ, em oposição à $k\varpi$, pode tomar qualquer quantidade de blocos não reversíveis por operação.

Apesar de não podermos considerar a Proposição 1.6 para $k\varpi$, obtemos o seguinte limite inferior numa distância de $k\varpi$.

Teorema 1.7 *Dada uma permutação π, temos que* $d_{k\varpi}(\pi) \geq \left\lceil \frac{b(\pi)}{2(k+1)} \right\rceil$.

Como visto anteriormente, limites de distâncias das operações de transposição e de movimentos de blocos são obtidos através do diagrama de realidade e desejo. Não mencionamos aqui, porém limites também foram obtidos para distância de reversão através de uma variação do diagrama de realidade e desejo, chamada de *grafo de pontos de quebra*. Este grafo também é útil no contexto da distância de $k\varpi$.

O *grafo de pontos de quebra* [4] de uma permutação $\pi = [\mathbf{0}\,\pi_1 \cdots \pi_n\,\boldsymbol{n+1}]$ é o grafo $PQ(\pi) = (V, E)$ com conjunto de vértices $V = \{0, 1, \ldots, n, n+1\}$ e com conjunto de arestas E particionado em: *arestas pretas*, que conecta pares de vértices que correspondem a um ponto de quebra em π; e *arestas cinzas*, que conectam pares de vértices que correspondem elementos que são consecutivos em ι porém não consecutivos em π.

Um *ciclo alternante* neste grafo é um ciclo cujas arestas no ciclo alternam entre pretas e cinzas. Existem diversas formas de decompor o grafo de ponto de quebras em ciclos alternantes disjuntos. Usamos a notação $c(PQ(\pi))$ para denotar a cardinalidade de uma decomposição máxima de ciclos alternantes disjuntos do grafo de pontos de quebra de π $PQ(\pi)$. A Figura 1.4 ilustra $PQ([\mathbf{0}\,3\,1\,5\,7\,4\,2\,6\,\mathbf{8}])$ e uma correspondente decomposição máxima de ciclos alternantes disjuntos.

Os grafos de pontos de quebra foram definidos por Bafna e Pevzner [4] no contexto da distância de reversão e limites de distâncias não triviais foram obtidos.

Dadas uma permutação π e uma reversão ρ em π, $\Delta c(PQ(\pi\rho)) = c(PQ(\pi\rho)) - c(PQ(\pi))$, e $\Delta b(\pi\rho) = b(\pi\rho) - b(\pi)$. Bafna e Pevzner [4] provaram que para toda permutação π e qualquer reversão ρ, temos $\Delta c(PQ(\pi\rho)) - \Delta b(\pi\rho) \leq 1$ e que a distância de reversão é pelo menos $b(\pi) - c(PQ(\pi))$.

Obtemos também um limite inferior na distância de $k\varpi$ pelo uso do grafo de pontos de quebra.

Lema 1.8 *Para toda permutação π e qualquer $k\varpi$,* $\Delta c(PQ(\pi\varpi)) - \Delta b(\pi\varpi) \leq k+1$.

Teorema 1.9 *Para toda permutação π,* $d_{k\varpi}(\pi) \geq \left\lceil \frac{b(\pi) - c(PQ(\pi))}{k+1} \right\rceil$.

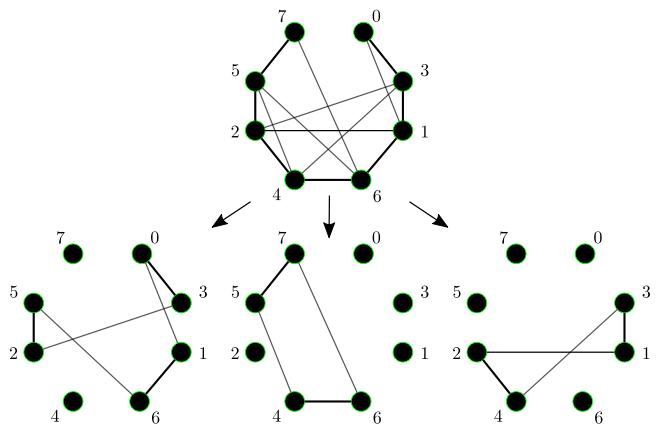

Figura 1.4: $PQ([\mathbf{0}\,3\,1\,6\,4\,2\,5\,\mathbf{7}])$, e uma correspondente decomposição máxima de ciclos alternantes disjuntos.

Diâmetro O estudo do diâmetro surge como uma maneira de entender, para uma dada operação, o quão longe as permutações podem estar da identidade.

Definição 1.10 (Diâmetro de uma operação P) *Dada uma operação P, o* diâmetro $D(n)$ *é o valor da maior distância da operação P de qualquer permutação com n elementos. Ou seja,* $D(n) = \max_{\pi \in S_n}\{d_P(\pi)\}$.

Note que isto é exatamente o diâmetro no *grafo de Cayley* associado, onde o conjunto de vértices é definido pelas $n!$ permutações, e dois vértices são vizinhos se suas correspondentes permutações estão à distância 1. O diâmetro num grafo qualquer pode ser obtido em tempo polinomial, simplesmente pela computação da distância entre todos os pares de vértices num grafo, de modo que a maior das distâncias será o diâmetro do grafo. Nosso problema surge devido a entrada do grafo já ser exponencial, $n!$ permutações.

Em relação as operações tratadas nesta tese, temos que:

- Diâmetro de transposição. Problema em aberto. Diversos artigos foram publicados propondo melhores limites [29, 30, 40, 46, 48]. Estabelecemos, durante a dissertação de mestrado em 2013, o melhor limite inferior corrente do diâmetro [17, 18] de $\lfloor \frac{n+1}{2} \rfloor + 1$. Valor que para n ímpar há permutações com distâncias maiores que a distância da permutação reversa, e para n par não se sabe se há permutações com distâncias mariores do que a reversa;

- Diâmetro de movimentos de blocos. Problema polinomial, Christie [9]. Como o problema de distância de uma permutação π de tamanho n é igual a $\lceil \frac{n+1-c(\pi)}{2} \rceil$, é possível obter uma permutação que minimiza o número de ciclos. Com isso, o diâmetro é igual a $\lceil \frac{n}{2} \rceil$. Um exemplo de permutação diametral é a reversa de tamanho par;

- Diâmetro de movimentos de blocos curtos. Problema polinomial, Heath e Vergara [41]. Apesar do problema de distância estar em aberto, Heath e Vergara apresentaram uma estratégia que ordena qualquer permutação com $\binom{n}{2}$ operações, além disso eles provaram que a distância da permutação reversa de tamanho n é $\binom{n}{2}$. Com isso, a reversa é diametral;

- Diâmetro de reversão. Problema polinomial, Bafna e Pevzner [4]. De maneira similar ao diâmetro de movimentos de blocos curtos, há uma estratégia que ordena qualquer permutação, neste caso com $n-1$ operações, e Bafna e Pevzner mostraram uma classe de permutações com distâncias iguais a $n-1$. É a classe das *permutações Gollan*, tratada no Capítulo 4;

- Diâmetro de multi-corte restritos. Problema em aberto. Limites justos são vistos no Capítulo 4, e no Anexo D, onde apresentamos também os desafios atuais a cerca deste problema.

A Tabela 1.1 descreve as complexidades dos problemas de distância e de diâmetro das operações discutidas.

	Transposição	Movimento de blocos	Movimento de blocos curtos	Reversão	Multi corte restrito
Distância	NP-completo [7]	Polinomial [9]	Em aberto	NP-completo [8]	Em aberto
Diâmetro	Em aberto	Polinomial [9]	Polinomial [42]	Polinomial [4]	Em aberto

Tabela 1.1: Complexidade computacional dos problemas de distância e de diâmetro vistos até aqui.

1.4 Permutação mais próxima

No Capítulo 3, e o correspondente Anexo B, tratamos do problema de determinar as permutações mais próximas. Quando o problema de ordenação de permutações é polinomial ou permanece em aberto, um problema mais geral é proposto.

Para um conjunto de permutações, uma métrica de distância e um inteiro, buscamos encontrar uma permutação que possua distância para cada uma das permutações da entrada no máximo o inteiro da entrada.

PERMUTAÇÃO MAIS PRÓXIMA PELA DISTÂNCIA M (M–PMP)
ENTRADA: Conjunto de permutações $\{p_1, p_2, \ldots, p_k\}$ em que cada permutação possui tamanho n, e um inteiro não negativo d.
PERGUNTA: Existe uma permutação π de tamanho n tal que $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$?

Ao considerar o conjunto de entrada sendo formado por cadeias de caracteres, o problema de determinar a cadeia de caracteres mais próxima pela *distância de Hamming* – definida pelo número de posições com elementos distintos entre duas cadeias – foi provado por Lanctot *et al.* [45] ser NP-completo mesmo no caso de alfabeto binário.

Permutações são cadeias de caracteres restritas, já que cada caracter do alfabeto aparece exatamente uma vez, portanto o problema de determinar a permutação mais próxima é uma restrição natural. Este problema já foi estudado considerando distância de Cayley por Popov [51], tendo provado que determinar a permutação mais próxima pela distância de Cayley é NP-completo.

Este problema não foi estudado ao considerar outras métricas polinomiais de distância. Se o problema de ordenação é NP-completo, então o de determinar a permutação mais próxima também será.

Consideramos nesta tese três métricas de distância, onde provamos NP-completudes, conforme visto na Tabela 1.2.

	Pontos de quebra	Movimento de blocos	Movimento de blocos curtos
Distância	Polinomial	Polinomial	Em aberto
Permutação mais próxima	NP-completo	NP-completo	NP-completo

Tabela 1.2: NP-completudes para permutação mais próxima provadas nesta tese em relação às distâncias conhecidas.

Capítulo 2

Algoritmo 1,375-aproximativo para ordenação por tranposições com complexidade de tempo $O(n \log n)$

Como visto no Capítulo 1, problemas de ordenação de permutações são interessantes não somente pela motivação biológica, mas também pelo seu desafio combinatório.

O problema de ordenação por transposições foi proposto por Bafna e Pevzner [3] em 1998, e provado ser NP-difícil por Bulteau *et al.* [7] somente em 2012. Assim, alguns algoritmos polinomiais aproximativos para este problema foram desenvolvidos. Hartman e Shamir [39] em 2006, desenvolveram um algoritmo 1,5-aproximativo com complexidade de tempo de $O(n^{\frac{3}{2}}\sqrt{\log n})$. Esta complexidade foi reduzida para $O(n \log n)$ por Feng e Zhu [33] em 2007, ao propor uma estrutura de dados, chamada árvore de permutação. Elias e Hartman [29] em 2006, desenvolveram um algoritmo 1,375-aproximativo, através de uma análise sistemática computacional dos ciclos do grafo de realidade e desejo, este algoritmo possui complexidade de tempo de $O(n^2)$. Firoz *et al.* em 2011, afirmaram ter melhorado a complexidade do algoritmo de Elias e Hartman de $O(n^2)$ para $O(n \log n)$ pela utilização da árvore de permutação.

Apresentamos contraexemplos em relação a corretude do algoritmo de Elias e Hartman pela estratégia de Firoz *et al.*, mostramos que não é possível atingir extensões suficientes para famílias infinitas de permutações. Além disso, desenvolvemos um algoritmo 1,375-aproximativo tal que pela utilização da árvore de permutação, atingimos complexidade de tempo $O(n \log n)$. A corretude de nosso algoritmo é obtida por uma análise computacional de força bruta que encontra uma $\frac{11}{8}$-sequência para toda combinação de configurações pequenas ruins.

Os resultados deste capítulo foram apresentados nas seguintes conferências:

- Brazilian Symposium on Bioinformatics 2013 [19];
- Workshop on Algorithms in Bioinformatics 2014 [20].

Além disso, publicamos no periódico *Journal of Computational Biology* 2015 [21], manuscrito encontrado no Anexo A.

Este capítulo está organizado da seguinte forma: na Seção 2.1 apresentamos definições e propriedades do diagrama de realidade e desejo; na Seção 2.2 apresentamos o algoritmo de Elias e Hartman; na Seção 2.3 apresentamos a árvore de permutação; na Seção 2.4 tratamos sobre a estratégia de Firoz *et al.* no uso da árvore de permutação no algoritmo de Elias e Hartman, onde mostramos exemplos que invalidam esta estratégia; na Seção 2.5 mostramos como determinar em tempo linear uma sequência de duas transposições em que cada uma é um 2-movimento; e na Seção 2.6 descrevemos nosso algoritmo 1,375-aproximativo com complexidade de tempo $O(n \log n)$.

2.1 Interações entre ciclos do diagrama de realidade e desejo

Como visto na Seção 1.3, através do diagrama de realidade e desejo, limites para a distância de transposição foram obtidos. Em particular, pelo número de ciclos podemos mensurar limites inferiores e superiores para a distância de transposição.

Um ciclo de uma permutação π é identificado unicamente por suas arestas de realidade, listando pela ordem que as arestas aparecem da esquerda para a direita. Um ℓ-ciclo C é portanto representado por $C = \langle x_1 x_2 \ldots x_\ell \rangle$, de forma que $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, $\ldots$, $\overrightarrow{x_\ell}$ são arestas de realidade, e $x_1 = \min\{x_1, x_2, \ldots, x_\ell\}$. O ciclo *mais à esquerda* é o ciclo que contém a aresta $\overrightarrow{0}$.

Sejam $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$, arestas de realidade de um ciclo C tais que $x < y < z$, e $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$, arestas de realidade de um outro ciclo C' tais que $a < b < c$. O par de arestas $\overrightarrow{x}, \overrightarrow{y}$ *intersecta* o par $\overrightarrow{a}, \overrightarrow{b}$ se essas quatro arestas aparecem em ordem alternada no diagrama de realidade e desejo, ou seja, ou $x < a < y < b$ ou $a < x < b < y$. Dizemos que os ciclos C e C' se *intersectam*. Similarmente, uma tripla $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$ *intercala* a tripla $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$, se essas arestas aparecem em ordem alternada: $x < a < y < b < z < c$ ou $a < x < b < y < c < z$. Um 3-ciclo C e C' se *intercalam* se suas respectivas triplas de arestas se intercalam. A Figura 2.1 ilustra esses conceitos.

Uma *configuração* de uma permutação π é um subconjunto de ciclos em $G(\pi)$. Uma configuração $\mathcal{C}$ é *conexa* se existe uma sequência de ciclos $C_1, ..., C_k$ em $\mathcal{C}$ tal que $C_1 = C$, $C_k = C'$ e para cada $i \in \{1, 2, ..., k-1\}$, os ciclos C_i e C_{i+1} se intersectam. Uma *componente* é uma configuração conexa maximal em relação a propriedade de conexidade. Toda permutação admite uma única decomposição em componentes distintas. Por exemplo na Figura 2.1, a configuração $\{C_1, C_2, C_3, C_4\}$ é uma componente, ao passo que a configuração $\{C_1, C_2, C_3\}$ não é uma componente,

apesar de ser conexa.

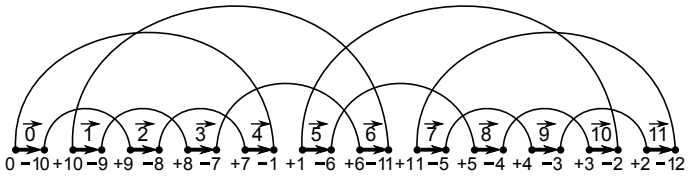

Figura 2.1: $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$. O conjunto de ciclos é $\{C_1 = \langle 0\,2\,4\rangle, C_2 = \langle 1\,3\,6\rangle, C_3 = \langle 5\,8\,10\rangle, C_4 = \langle 7\,9\,11\rangle\}$. Os ciclos C_2 e C_3 se intersectam, mas não se intercalam; os ciclos C_1 e C_2 se intercalam, e da mesma forma os ciclos C_3 e C_4. O ciclo C_1 é o ciclo mais à esquerda.

Seja C um 3-ciclo de uma configuração $\mathcal{C}$. Uma *porta aberta* é um par de arestas de realidade de C que não intersecta nenhum outro par de arestas de realidade de um ciclo em $\mathcal{C}$. Se uma configuração $\mathcal{C}$ não possui nenhuma porta aberta, então $\mathcal{C}$ é uma *configuração completa*.

Nem todas configurações correspondem a diagramas de realidade e desejo de alguma permuntação. Um configuração completa corresponde a alguma permutação se, e somente se, sua *configuração complementar* é Hamiltoniana. [29], como ilustrado na Figura 2.2b. A Figura 2.2a mostra uma configuração completa $F = \{\langle 0\,7\,9\rangle$, $\langle 1\,3\,6\rangle$, $\langle 2\,4\,11\rangle$, $\langle 5\,8\,10\rangle\}$, que não corresponde a nenhuma permutação. Esta configuração é importante para a análise de nosso algoritmo, estudada na Seção 2.6.

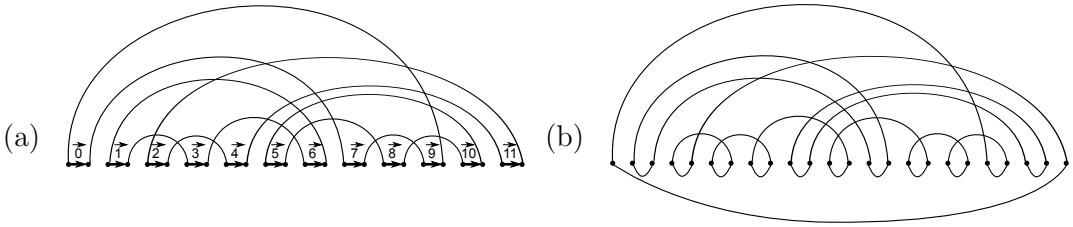

Figura 2.2: (a) Configuração completa $F = \{\langle 0\,7\,9\rangle$, $\langle 1\,3\,6\rangle$, $\langle 2\,4\,11\rangle$, $\langle 5\,8\,10\rangle\}$, que não corresponde a permutação. (b) Configuração complementar F, obtida pela troca das arestas de realidade pelas as arestas $-\pi_i\pi_i$ e $0 - \pi_{n+1}$.

2.2 Algoritmo 1,375-aproximativo de Elias e Hartman com complexidade de tempo $O(n \log n)$

Elias e Hartman [29] enumeraram todos os componentes de no máximo nove ciclos, tal que cada ciclo possui tamanho no máximo 3. Sua estratégia inicia-se por um único 3-ciclo, a partir daí componentes são obtidas pela série de *extensões suficientes*, conforme descritas a seguir. Uma extensão suficiente é uma inclusão de um novo ciclo na configuração de forma que:

1. estende à uma configuração de tal forma que fecha uma porta aberta;

2. estende uma configuração completa à uma configuração de tal forma que cria no máximo uma porta aberta.

Uma configuração obtida por uma série de extensões suficientes é uma *configuração suficiente* se uma (x, y)-, ou uma $\frac{x}{y}$-sequência pode ser aplicada aos seus ciclos.

Elias e Hartman [29] mostraram, através de um algoritmo de força bruta, uma $\frac{11}{8}$-sequência para toda configuração suficiente com nove ciclos.

Lema 2.1 *[29] Toda configuração suficiente não-orientada com nove ciclos possui uma $\frac{11}{8}$-sequência.*

Componentes com menos que nove ciclos são chamadas de *componentes pequenas*. Componentes pequenas que possuem $\frac{11}{8}$-sequência são chamadas de *componentes pequenas boas*, e componentes pequenas que não possuem $\frac{11}{8}$-sequência são chamadas de *componentes pequenas ruins*. Elias e Hartman mostraram, também por força bruta, que dentre todas as componentes pequenas, somente cinco delas são componentes pequenas ruins.

Lema 2.2 *[29] As componentes pequenas ruins são:*

- $A = \{\langle 0\,2\,4\rangle,\ \langle 1\,3\,5\rangle\}$;
- $B = \{\langle 0\,2\,10\rangle,\ \langle 1\,3\,5\rangle,\ \langle 4\,6\,8\rangle,\ \langle 7\,9\,11\rangle\}$;
- $C = \{\langle 0\,5\,7\rangle,\ \langle 1\,9\,11\rangle,\ \langle 2\,4\,6\rangle,\ \langle 3\,8\,10\rangle\}$;
- $D = \{\langle 0\,2\,4\rangle,\ \langle 1\,12\,14\rangle,\ \langle 3\,5\,7\rangle,\ \langle 6\,8\,10\rangle,\ \langle 9\,11\,13\rangle\}$; *e*
- $E = \{\langle 0\,2\,16\rangle,\ \langle 1\,3\,5\rangle,\ \langle 4\,6\,8\rangle,\ \langle 7\,9\,11\rangle,\ \langle 10\,12\,14\rangle,\ \langle 13\,15\,17\rangle\}$.

Apesar de não ser possível aplicar uma $\frac{11}{8}$-sequência numa componente pequena ruim, Elias e Hartman provaram ser possível aplicar $(11, 8)$-sequência a uniões destas componentes. Conforme enunciado no Lema 2.3.

Lema 2.3 *[29] Seja π uma permutação com pelo menos oito ciclos e possuindo somente componentes pequenas ruins. Assim, π possui $(11, 8)$-sequência.*

Corolário 2.4 *[29] Se todo ciclo em $G(\pi)$ é um 3-ciclo, e existem pelo menos oito ciclos, então π posui uma $\frac{11}{8}$-sequência.*

Os Lemas 2.1 e 2.3, e o Corolário 2.4 formam a procedimento básico do algoritmo $\frac{11}{8} = 1,375$-aproximativo de Elias e Hartman, descrito no Algoritmo 1, cuja idéia geral é: Obtenha extensões, se uma configuração com nove ciclos ou se uma componente pequena boa é obtida, então uma $\frac{11}{8}$-sequência é aplicada (Lema 2.1); se

uma componente pequena ruim é obtida, então nenhuma sequência de transposições é aplicada. Após todas as configurações de tamanho 9 ou todas as componentes pequenas boas terem sido ordenadas, a permutação resultante possui somente componentes pequenas ruins, e assim o Lema 2.3 garante a existência de $(11, 8)$-sequência a serem aplicadas.

Algoritmo 1: Elias e Hartman Ordena(π)

1 Transforme π numa permutação simples $\hat{\pi}$.
2 Cheque se existe uma $(2, 2)$-sequência. Se existir, aplique.
3 Enquanto $G(\hat{\pi})$ possuir um 2-ciclo, aplique um 2-movimento.
4 $\hat{\pi}$ contém 3-ciclos. Marque todos 3-ciclos em $G(\hat{\pi})$.
5 **enquanto** $G(\hat{\pi})$ *conter um* 3*-ciclo marcado* C **faça**
6 **se** C *é orientado* **então**
7   Aplique um 2-movimento.
8 **senão**
9   Tente estender suficiente C oito vezes (para obter uma configuração com no máximo nove ciclos).
10   **se** *configuração suficiente é obtida* **então**
11   Aplique uma $\frac{11}{8}$-sequência.
12   **senão** É uma componente pequena
13   **se** *é uma componente boa* **então**
14   Aplique uma $\frac{11}{8}$-sequência.
15   **senão**
16   Desmarque todos os ciclos da componente.
17 (Agora $G(\hat{\pi})$ possui apenas componentes pequenas ruins.)
18 **enquanto** $G(\hat{\pi})$ *contém pelo menos oito ciclos* **faça**
19 Aplique uma $(11, 8)$-sequência
20 Enquanto $G(\hat{\pi})$ contém um 3-ciclo, aplique uma $(3, 2)$-sequência.
21 Obtenha a sequência que ordena π obtida pela sequência que ordena $\hat{\pi}$.

Este algoritmo possui complexidade de tempo quadrática [29], dado que no Passo 2 é feita uma simples busca por um par de ciclos que satisfaça a propriedade de existência de $(2, 2)$-sequência, e no Passo 9 uma extensão é feita pela busca a algum outro ciclo que intersecte uma dada configuração.

Teorema 2.5 *[29] O Algoritmo 1 possui complexidade de tempo quadrática.*

2.3 Árvore de permutação de Feng e Zhu

Feng e Zhu [33] desenvolveram a *árvore de permutação*, uma árvore binária balanceada que representa uma permutação, e além disso, uma operação de transposição pode ser realizada com complexidade de tempo logarítmico.

Seja $\pi = [\boldsymbol{\pi_0}\pi_1\pi_2\cdots\pi_n\boldsymbol{\pi_{n+1}}]$ uma permutação. Sua correspondente árvore de permutação possui n folhas, que são os elementos $\pi_1, \pi_2, \cdots, \pi_n$; cada nó interno da árvore representa um intervalo dos elementos consecutivos $\pi_i, \pi_{i+1}, \cdots, \pi_{k-1}$, com $i < k$, e este nó interno é rotulado pelo valor máximo dos elementos no intervalo. Assim, o nó raíz da árvore é rotulado pelo valor n. O filho esquerdo de um nó representa o intervalo $\pi_i, \cdots, \pi_{j-1}$ e o filho direito representa o intervalo $\pi_j, \cdots, \pi_k$, onde $i < j < k$. Feng e Zhu propuseram algoritmos: *construção* da árvore de permutação com complexidade de tempo de $O(n)$; *junção* de duas árvores de permutação em uma com complexidade de tempo de $O(h)$, onde h é a diferença das alturas das árvores; e *separação* de uma árvore de permutação em duas com complexidade de tempo de $O(\log n)$.

A Figura 2.3a ilustra o diagrama de realidade e desejo de uma permutação e sua árvore de permutação na Figura 2.3b. Note que as cores das folhas representam os ciclos do diagrama de realidade e desejo correspondente.

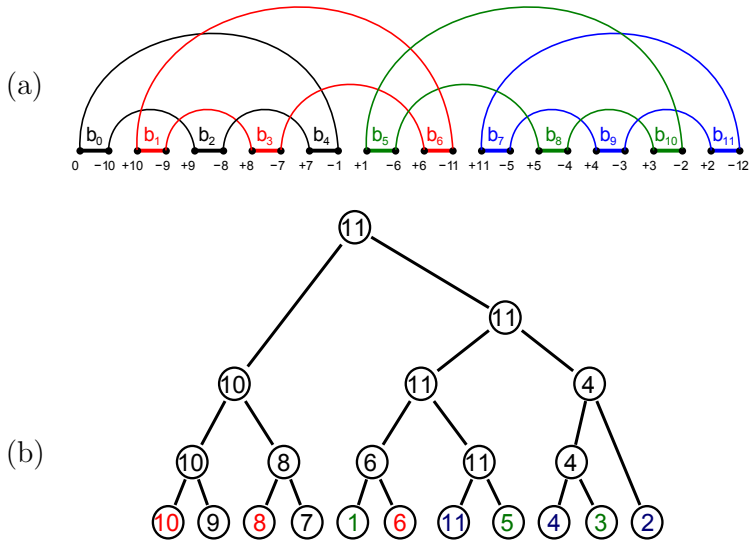

Figura 2.3: (a) Diagrama de realidade e desejo de $\pi = [10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2]$. (b) Árvore de permutação de π.

As operações de *separação* e *junção* nos permitem aplicar uma transposição em uma permutação com a atualização da árvore com complexidade de tempo de $O(\log n)$. Feng e Zhu propuseram também o procedimento de *acessar*, que possibilita encontrar um par de arestas de realidade que intersecta um outro par de arestas de realidade do diagrama de realidade e desejo. O procedimento *acessar* é descrito pelo Algoritmo 2, baseado no Lema 2.6.

Lema 2.6 *[3] Sejam $\overrightarrow{i}$ e $\overrightarrow{j}$ duas arestas de realidade num ciclo não-orientado C, $i < j$. Seja $\pi_k = \max_{i<m\leq j} \pi_m$, $\pi_\ell = \pi_k + 1$. Assim, as arestas de realidade $\overrightarrow{k}$ e $\overrightarrow{\ell - 1}$ pertencem ao mesmo ciclo, e o par $\overrightarrow{k}, \overrightarrow{\ell - 1}$ intersecta o par $\overrightarrow{i}, \overrightarrow{j}$.*

Algoritmo 2: $Acessar(\pi, i, j)$

Entrada: permutação π, inteiros i e j

1 Seja T uma árvore de permutação de π

2 Aplique a separação de T, em três árvores de permutação, T_1, T_2 e T_3, correspondente a $[\boldsymbol{\pi_0}, \pi_1, \cdots, \pi_i]$, $[\pi_{i+1}, \cdots, \pi_j]$, e $[\pi_{j+1}, \cdots, \pi_n, \boldsymbol{\pi_{n+1}}]$, respectivamente.

3 $\pi_k = raiz(T_2)$. (o maior elemento no intervalo $\pi_{i+1}, \cdots, \pi_j$)

4 $\pi_\ell = \pi_k + 1$

5 Retorne o par k, $\ell - 1$ (pelo Lema 2.6, $\overrightarrow{k}, \overrightarrow{l-1}$ intersecta $\overrightarrow{i}, \overrightarrow{j}$)

A partir dos procedimentos possíveis de serem aplicados em uma permutação pela utilização da árvore de permutação, Firoz *et al.* sugeriram o uso desta estrutura de dados no Algoritmo 1 de Elias e Hartman com o objetivo de reduzir a complexidade de $O(n^2)$ para $O(n \log n)$. A seguir, na Seção 2.4 mostramos que esta estratégia falha ao estender alguns 3-ciclos em configurações completas com nove ciclos.

2.4 Uso da árvore de permutação por Firoz *et al.*

Firoz *et al.* afirmaram que extensões suficientes (Passo 9 do Algoritmo 1) poderiam ser feitas em $O(\log n)$. Para isso, novos ciclos seriam adicionados a uma configuração A pelo uso do procedimento *acessar*, este procedimento seria portanto efetuado em chamadas dos tipos 1 e 2. Extensão do *tipo 1* é aquela que fecha uma porta aberta de A; e extensão do *tipo 2* é aquela que estende uma configuração completa A pela adição de um novo ciclo C tal que $A \cup \{C\}$ possui no máximo uma porta aberta.

Uma extensão do tipo 1 pode de fato ser executada por $Acessar(\pi, i, j)$, de tal forma que $\overrightarrow{i}, \overrightarrow{j}$ formam uma porta aberta. Porém, extensões do tipo 2 podem não encontrar novos ciclos intersectando pares de arestas de realidade de uma configuração pela utilização de $Acessar(\pi, i, j)$.

Considere a permutação σ abaixo e seu diagrama de realidade e desejo ilustrado na Figura 2.4, que possui uma 11/8-sequência por conter uma configuração de nove ciclos (Lema 2.1).

$$\sigma = [\mathbf{0}\ 25\ 24\ 23\ 22\ 1\ 21\ 26\ 20\ 19\ 18\ 2\ 17\ 27\ 16\ 15\ 14\ 3\ 13\ 28\ 12\ 11\ 10\ 4\ 9\ 29\ 8\ 7\ 6\ 5\ \mathbf{30}].$$

Segundo a estratégia de Firoz *et al.*, temos os seguinites passos: começamos por uma configuração possuindo um único ciclo, após a primeira chamada ao procedimento *acessar* obtemos corretamente um novo ciclo que intersecta o primeiro (esta nova configuração pode ser vista na Figura 2.5); a partir desta configuração

possuindo dois ciclos intercalados, qualquer outra chamada ao procedimento *acessar* retorna um dos outros ciclos já pertencentes à configuração (esta estratégia não estende a configuração); a configuração resultante (Figura 2.5) é uma componente pequena ruim, portanto pelo algoritmo de Elias e Hartman os ciclos são desmarcados; se um ciclo desmarcado continua sem ter sido ordenado, este é selecionado para iniciar uma configuração, sendo assim não é possível aplicar uma extensão suficiente para mais que dois ciclos. O procedimento termina após desmarcar todos os dez ciclos incorretamente presumindo que o diagrama de realidade e desejo possui somente componentes pequenas ruins com dois ciclos, cada.

Note que de acordo com o Passo 18 do Algoritmo 1, se uma permutação contém somente componentes pequenas ruins, então uma 11/8-sequência pode ser aplicada. A permutação γ, com diagrama de realidade e desejo ilustrado na Figura 2.6 é uma dessas permutações:

$$\gamma = [\mathbf{0}\ 5\ 4\ 3\ 2\ 1\ 6\ 11\ 10\ 9\ 8\ 7\ 12\ 17\ 16\ 15\ 14\ 13\ 18\ 23\ 22\ 21\ 20\ 19\ 24\ 29\ 28\ 27\ 26\ 25\ \mathbf{30}].$$

Temos assim que os diagramas das Figuras 2.5 e 2.6 são distintos, mas de acordo pela estratégia de Firoz *et al.*, esses diagramas são indistinguíveis. Na Figura 2.6, as componentes pequenas ruins podem ser separadamente tratadas, que não é o caso das configurações da Figura 2.5 com ciclos intercalados. Portanto, no Algoritmo 1 não há uma regra para lidar com este último caso.

Uma família infinita de permutações cuja estratégia de Firoz *et al.* falha pode ser obtida como segue: seja k um inteiro maior ou igual a 2, e seja $f(i)$ uma sequência de seis inteiros

$$i\ \ 5k-4i\ \ 5k+i\ \ 5k-4i-1\ \ 5k-4i-2\ \ 5k-4i-3. \tag{2.1}$$

Seja σ^k uma permutação de $6k-1$ elementos definida usando a Equação (2.1):

$$[\mathbf{0}\ \ 5k\ \ 5k-1\ \ 5k-2\ \ 5k-3\ \ f(1)\ \ f(2)\ldots\ \ f(k-1)\ \ k\ \ \mathbf{6k}], \tag{2.2}$$

que possui diagrama de realidade e desejo similar ao da Figura 2.1 (onde na Equação (2.2) $k = 2$) e 2.4.

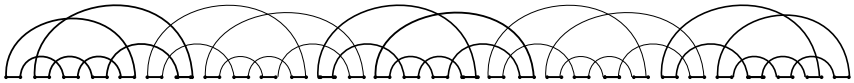

Figura 2.4: Diagrama de realidade e desejo de σ em que a estratégia de Firoz *et al.* falha. Note que σ possui dez ciclos, e que σ é obtido ao assumir $k = 5$ na Equação (2.2).

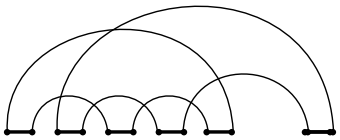

Figura 2.5: Uma configuração que não é maximal retornada pelo procedimento *acessar* em σ.

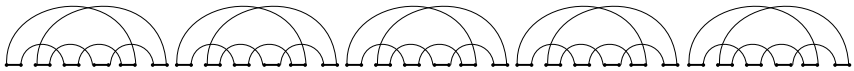

Figura 2.6: Diagrama de realidade e desejo de γ.

Algumas outras configurações podem ser obtidas pelo uso do procedimento *acessar*, por exemplo a configuração completa ilustrada na Figura 2.2. Esta é uma configuração pequena ruim que não corresponde ao diagrama de realidade e desejo de nenhuma permutação [29], porém esta configuração pode aparecer durante a ordenação de alguma permutação maior.

2.5 Encontrar e aplicar uma $(2,2)$-sequência em tempo linear

Elias e Hartman [29] mostraram que no Passo 2 do Algoritmo 1 uma $(2,2)$-sequência pode ser obtida em tempo quadrático.

Firoz *et al.* [35] descreveram uma maneira de encontrar e aplicar uma $(2,2)$-sequência com complexidade de tempo de $O(n \log n)$. Sua estratégia basea-se para cada 3-ciclo orientado (que podem haver $O(n)$ ciclos), aplicamos um 2-movimento, e após isto buscamos em tempo $O(n)$ por um outro ciclo orientado, este procedimento é porém quadrático no pior caso. Um exemplo pode ser visto na Figura 2.7, onde os ciclos com arestas contínuas são intercalados – um ciclo orientado, e outro ciclo não-orientado – e pelo Caso 3 do Lema 2.7 existe uma $(2,2)$-sequência que afeta os dois ciclos. Porém, se o primeiro 2-movimento é aplicado a qualquer ciclo com arestas pontilhadas, o diagrama de realidade e desejo possuirá nenhum ciclo orientado. Portanto a transposição é desfeita e outro ciclo é selecionado. Assim, a busca de um novo ciclo orientado é refeita até que o ciclo contínuo orientado seja escolhido.

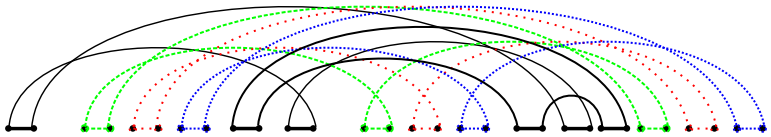

Figura 2.7: Diagrama de realidade e desejo cuja estratégia de Firoz et al. para encontrar $(2,2)$-sequência toma complexidade de tempo de $\Omega(n^2)$ no pior caso.

Algoritmo 3: Busca $(2,2)$-sequência de K_1.

1 **para** $i = \min K_1{+}1, \ldots, \operatorname{mid} K_1{-}1$ **hacer**
2   **se** $\overrightarrow{i}$ *pertence a um ciclo orientado* K_j **então**
3   **se** $\operatorname{mid} K_j < \operatorname{mid} K_1$ **então**
4   **retorna** $(2,2)$-sequência que afeta K_1 e K_j.
5   **senão se** $\max K_j < \max K_1$ **então**
6   **retorna** $(2,2)$-sequência que afeta K_1 e K_j.
7   **senão se** $\max K_1 < \operatorname{mid} K_j$ **então**
8   **retorna** $(2,2)$-sequência que afeta K_1 e K_j.
9   **se** $\overrightarrow{i}$ *pertence a um ciclo não-orientado* L_j **então**
10   **se** $\operatorname{mid} K_1 < \operatorname{mid} L_j < \max K_1 < \max L_j$ **então**
11   **retorna** $(2,2)$-sequência que afeta K_1 e L_j.
12   **senão se** $\min L_j < \min K_1 < \operatorname{mid} L_j < \operatorname{mid} K_1 < \max L_j < \max K_1$ **então**
13   **retorna** $(2,2)$-sequência que afeta K_1 e L_j.
14 **para** $i = \operatorname{mid} K_1{+}1, \ldots, \max K_1{-}1$ **hacer**
15   **se** $\overrightarrow{i}$ *pertence a um ciclo orientado* K_j **então**
16   **se** $\operatorname{mid} K_1 < \min K_j$ **então**
17   **retorna** $(2,2)$-sequência que afeta K_1 e K_j.
18 **para** $i = \max K_1{+}1, \ldots, n{-}1$ **hacer**
19   **se** $\overrightarrow{i}$ *pertence a um ciclo orientado* K_j **então**
20   **se** $\max K_1 < \min K_j$ **então**
21   **retorna** $(2,2)$-sequência que afeta K_1 e K_j.

O Algoritmo 5 resume nossa proposta de encontrar e aplicar uma $(2,2)$-sequência com complexidade de tempo de $O(n)$. É uma aplicação dos Algoritmos 3 e 4 aos casos listados no Lema 2.7.

Lema 2.7 *[3, 9, 29] Dado o diagrama de realidade e desejo de uma permutação simples, existe uma* $(2,2)$*-sequência se uma das seguinites condições satisfeita:*

1. *ou existem quatro* 2*-ciclos, ou dois* 2*-ciclos que se intersectam, ou dois* 2*-ciclos que não se intersectam, e o grafo resultante contém um ciclo orientado após a primeira transposição*

2. *existem dois* 3*-ciclos orientados que não se intercalam;*

3. *existe um ciclo orientado que intercala um ciclo não-orientado.*

Primeiramente, checamos o primeiro caso do Lema 2.7, confome descrito das linhas 1 até 4 no Algoritmo 5. Não é necessário verificar todos os pares de ciclos nas

condições 2 e 3 do Lema 2.7. Isto difere dos métodos anteriores [29, 35], fixamos o primeiro ciclo orientado da esqueda para a direita do diagrama de realidade e desejo, o qual chamamos de K_1. Após isto enumeramos todos os ciclos em tempo linear. O tamanho de cada ciclo e sua orientação são determinados em tempo constante, dado que os ciclos são simples.

Christie [9] provou que toda permutação possui um número par de ciclos pares (possivelmente zero), ele também mostrou que dada uma permutação simples, quando o número de ciclos pares não é zero, então existe uma $(2,2)$-sequência que afeta esses ciclos se, e somente se, ou existem quatro 2-ciclos, ou existem dois ciclos pares que se intersectam. Nestes casos, uma $(2,2)$-sequência pode ser aplicada com complexidade de tempo de $O(n \log n)$ pela utilização da árvore de permutação. Se existe um único par de 2-ciclos não intersectantes, resta checar se existe um 3-ciclo que intersecta os dois ciclos pares: i) se o 3-ciclo é orientado, primeiramente aplicamos o 2-movimento ao 3-ciclo, e o segundo 2-movimento é aplicado aos 2-ciclos; ii) se o 3-ciclo é não-orientado, primeiramente aplicamos o 2-movimento aos 2-ciclos, e o segundo 2-movimento é aplicado ao 3-ciclo, que se torna orientado após a primeira transposição. Também existe uma $(2,2)$-sequência se existe um ciclo orientado que intesecta no máximo um ciclo par. Porém, se não existe ciclo par, mas existe um 3-ciclo orientado, devemos examinar a existência de uma $(2,2)$-sequência, conforme Casos 2 e 3 necessários pelo Lema 2.7.

Para analisar em tempo linear os Casos 2 e 3 do Lema 2.7, considere os ciclos orientados do diagrama de realidade e desejo, na ordem $K_1 = \langle a_1\, b_1\, c_1 \rangle, K_2 = \langle a_2\, b_2\, c_2 \rangle, K_3 = \langle a_3\, b_3\, c_3 \rangle, \ldots$ tal que $a_1 < a_2 < a_3 < \ldots$, e os ciclos não-orientados, na ordem $L_1 = \langle x_1\, y_1\, z_1 \rangle, L_2 = \langle x_2\, y_2\, z_2 \rangle, L_3 = \langle x_3\, y_3\, z_3 \rangle, \ldots$ tal que $x_1 < x_2 < x_3 < \ldots$. Dado um 3-ciclo $C = \langle a\, b\, c \rangle$, sejam $\min C = a$, $\operatorname{mid} C = \min\{b, c\}$ e $\max C = \max\{b, c\}$, i.e. se C é não-orientado, então $\min C = a$, $\operatorname{mid} C = b$, $\max C = c$, e se C é orientado, então $\min C = a$, $\operatorname{mid} C = c$, $\max C = b$. A idéia principal é:

1. Verifique a existência de um ciclo orientado K_j que não-intercala K_1 ou um ciclo não-orientado L_j que intercala K_1. Algoritmo 3 busca por um ciclo orientado K_i não-intercalado com K_1 ou um ciclo não-orientado L_i que intercala K_1. A busca é feita entre $\min K_1$ e $\operatorname{mid} K_1$, entre $\operatorname{mid} K_1$ e $\max K_1$, e à direita de $\max K_1$.

2. Se o Algoritmo 3 não retornar nenhum ciclo orientado que não-intercala K_1, então todo ciclo orientado intercala K_1 porém nenhum ciclo não-orientado intercala K_1. Portanto, devemos checar a existência de dois ciclos orientados K_i, K_j que se intersectam porém não sejam intercalados. Note que se K_i, K_j fossem orientados não-intersectantes, o Algoritmo 3 teria encontrado (veja

Algoritmo 4: Encontrando ciclos orientados que intercalam K_1.

s_1 = sequência de arestas pertencentes aos ciclos orientados da esquerda para a direita entre $\min K_1$ and $\operatorname{mid} K_1$.
s_2 = sequência de arestas pertencentes aos ciclos orientados da esquerda para a direita entre $\operatorname{mid} K_1$ and $\max K_1$.
se *as sequências de ciclos correspondentes a* s_1 *e* s_2 *são diferentes* **então**
  Existe um par de ciclos que se intersectam, existe uma $(2,2)$-sequência.
senão
  Todos os ciclos são mutuamente intercalados.

Figura 2.8), dado que ou K_i ou K_j não seria intercalado com K_1. Algoritmo 4 descreve como verificar a existência de dois ciclos orientados intersectantes que também são intercalados com K_1.

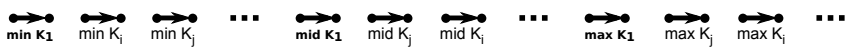

Figura 2.8: Ciclos orientados representados pelas arestas de realidade. Todos ciclos orientados intercalam K_1, mas existem i e j tais que K_i e K_j são não-intercalados.

Algoritmo 5: Busque e aplique $(2,2)$-sequência

se *existem quatro* 2*-ciclos* **então**
  Aplique uma $(2,2)$-sequência.
senão se *existe um par de* 2*-ciclos que se intersectam* **então**
  Aplique uma $(2,2)$-sequência.
senão se *existe um* 3*-ciclo que intersecta um par de* 2*-ciclos* **então**
  Aplique uma $(2,2)$-sequência.
senão se *existe um par de* 2*-ciclos* **e** *e um* 3*-ciclo orientado que intersecta no máximo um desses* 2*-ciclos* **então**
  Aplique uma $(2,2)$-sequência.
senão se *Busca* $(2,2)$*-sequência de* K_1 *retorna uma sequência* **então**
  Aplique uma $(2,2)$-sequência.
senão se *Encontrando ciclos orientados que intercalam* K_1*.* **então**
  Aplique uma $(2,2)$-sequência.
senão
  Não há $(2,2)$-sequências a serem aplicadas.

2.6 Novo algoritmo 1,375-aproximativo em tempo $O(n \log n)$

Na Seção 2.4, discutimos o uso da árvore de permutação no algoritmo de Elias e Hartman, de modo que este algoritmo não trata das configurações com menos de nove ciclos que não são componentes, já que sucessivas chamadas do procedimento *acessar* podem resultar em configurações completas com menos de nove ciclos que não são componentes pequenas. Nossa estratégia generaliza a definição de componentes pequenas para *configuraçõoes pequenas*, que são configurações com menos de nove ciclos. Uma configuração pequena é *completa* se não possui portas abertas. Classificamos também configurações pequenas como boas ou ruins, caso tenham ou não uma $\frac{11}{8}$-sequência, respectivamente.

Nossa estratégia trata configurações completas ruins – que podem ser ou não ser componentes pequenas – durante extensões por sucessivas chamadas ao procedimento *acessar*. As possíveis configurações pequenas completas ruins são as componentes A, B, C, D e E, do Lema 2.2, e a configuração completa ruim $F = \{\langle 0\,7\,9\rangle, \langle 1\,3\,6\rangle, \langle 2\,4\,11\rangle, \langle 5\,8\,10\rangle\}$, esta é a única que não é uma componente [29].

Nossa estratégia (Algoritmo 6) é similar a estratégia de Elias e Hartman (Algoritmo 1): aplique $\frac{11}{8}$-sequência a cada configuração suficiente não-orientada com nove ciclos e também a cada configuração pequena completa boa; a principal diferença é, sempre que uma combinação de configurações completas pequenas ruins é encontrada, uma decisão de aplicar uma $\frac{11}{8}$-sequência é realizada de acordo com os Lemas 2.8 e 2.9.

Desenvolvemos uma ferramenta [12] que encontra $\frac{11}{8}$-sequências para uma dada configuração usando uma técnica de branch-and-bound, em que o *branch* é obtido pela aplicação de um 2-movimento ou um 0-movimento, e o processo de *bound* é a razão entre o número total de transposições pelo número de 2-movimentos aplicados, de modo que não pode ser maior do que 1,375. O algoritmo ou encontra uma $\frac{11}{8}$-sequência, se existe, ou não retorna nada após tentar todas as possibilidades.

Lema 2.8 *Toda combinação de F com uma ou mais cópias de B, C, D ou E possui uma $\frac{11}{8}$-sequência.*

Observe que o Lema 2.8 não considera combinações de F com F, nem combinações de F com A. Encontramos $\frac{11}{8}$-sequências para quase todas as combinações de F com F, conforme enunciado no Lema 2.9. Seja F_iF^j a configuração obtida pela inserção da circularização $F + j$ entre as arestas $\overrightarrow{i}$ e $\overrightarrow{i+1}$ de F.

Lema 2.9 *Existe uma $\frac{11}{8}$-sequência para F_iF^j, se:*

- $i \in \{0, 4\}$ *e* $j \in \{0, 1, 2, 3, 4, 5\}$*; ou*

- $i \in \{1, 2, 3\}$ *e* $j \in \{1, 2, 3, 4, 5\}$*; ou*
- $i = 5$ *e* $j \in \{1, 5\}$.

Somente sete combinações de F com F não possuem uma $\frac{11}{8}$-sequência: F_1F^0, F_2F^0, F_3F^0, F_5F^0, F_5F^2, F_5F^3 e F_5F^4. Veremos o que fazer com elas a seguir.

Todas combinações de cópias de F e uma cópia de A possuem menos que oito ciclos. Resta-nos analisar combinações de F e duas cópias de A, combinação denotada por F–A–A. As *combinações boas* F–A–A são as combinações F–A–A que possuem $\frac{11}{8}$-sequência. Das 57 combinações de F–A–A, 31 são boas. A lista completa das combinações encontra-se em [12].

Combinações de F com A, B, C, D, E, e F que possuem $\frac{11}{8}$-sequências são chamadas de *combinações bem comportadas*: As combinações dos Lemas 2.8, 2.9, e as combinações boas F–A–A. As combinações restantes que possuem F são chamadas de *malcriadas*: as sete combinações de $F - F$ possuem nenhuma $\frac{11}{8}$-sequência, e as 57 combinações de F–A–A.

Ao obter extensões que geram uma configuração completa ruim, adicionamos estes ciclos ao conjunto $\mathcal{S}$ (linha 18) no Algoritmo 6. Após isto, se uma combinação bem comportada é encontrada entre os ciclos de $\mathcal{S}$, uma $\frac{11}{8}$-sequência é aplicada (linha 21) e o conjunto é esvaziado. Se todas as combinações de $\mathcal{S}$ são malcriadas, outra configuração pequena ruim é obtida e adicionada na próxima iteração (linha 6).

Mostramos que toda combinação de três cópias de F é bem comportada, mesmo se cada par de $F - F$ for malcriada [12]; o mesmo acontece com toda combinação de F com três cópias de A, mesmo se cada tripla de F–A–A for malcriada. Portanto, no máximo 12 ciclos estão em $\mathcal{S}$, de modo que pode conter no máximo três cópias de F, ou uma cópia de F e três cópias de A, no pior caso. Para cada um desses casos, existe uma $\frac{11}{8}$-sequência [12].

O algoritmo O Algoritmo 6 é uma aplicação dos resultados desta seção. Em linhas gerais, configurações são obtidas usando o procedimento *acessar* e aplicamos $\frac{11}{8}$-sequências à configurações de tamanho no máximo 9. O Algoritmo 6 difere do Algoritmo 1 não somente pelo uso da árvore de permutação, mas também devido ao passo principal lidarmos com configurações pequenas ruins, diferente do algoritmo de Elias e Hartman que trata delas somente no fim.

Teorema 2.10 *O Algoritmo 6 possui complexidade de tempo de* $O(n \log n)$.

Esquema da demonstração. Entre as Linhas 1 e 5 podem ser implementadas com complexidade de tempo linear ([29, 33] e na Seção 2.5). A Linha 11 pode ser implementada em tempo $O(\log n)$ pelo uso da árvore de permutação. A comparação das Linhas 12, 15, e 20 são feitas em $O(1)$ já que a quantidade de elementos é limitada

Algoritmo 6: Algoritmo proposto baseado no algoritmo de Elias e Hartman.

```
1  Transforme π numa permutação simples π̂.
2  Busque e aplique (2,2)-sequência (Algoritmo 5).
3  Enquanto G(π̂) possuir um 2-ciclo, aplique um 2-movimento.
4  π̂ contém 3-ciclos. Marque todos 3-ciclos em G(π̂).
5  Seja S um conjunto vazio.
6  enquanto G(π̂) contém pelo menos oito 3-ciclos faça
7      Comece uma configuração C com um 3-ciclo marcado.
8      se o ciclo em C é orientado então
9          Aplique um 2-movimento.
10     senão
11         Tente fazer uma extensão suficiente C oito vezes usando o
           procedimento acessar.
12         se C é uma configuração suficiente com nove ciclos então
13             Aplique uma 11/8-sequência.
14         senão
               C é uma configuração completa pequena
15             se C é uma configuração pequena ruim então
16                 Aplique uma 11/8-sequência.
17             senão
                   C é uma configuração pequena ruim
18                 Adicione todo ciclo C a S.
19                 Desmarque todos ciclos de C.
20                 se S contém combinações bem comportadas então
21                     Aplique uma 11/8-sequência.
22                     Marque os 3-cycles restantes em S.
23                     Remova todos ciclos de S.
24 Enquanto G(π̂) contém um 3-ciclo, aplique uma 4/3-sequência ou uma
   3/2-sequência.
25 Obtenha a sequência que ordena π obtida pela sequência que ordena π̂.
```

por uma constante. A atualização do conjunto $\mathcal{S}$ também gasta tempo constante, já que possui no máximo 12 ciclos (caso onde $\mathcal{S}$ contém $F-F-F$). Toda sequência de transposições consome tempo $O(\log n)$ pelo uso da árvore de permutações. Passos entre Linhas 6 e 23 também é implementada em $O(n \log n)$, já que o número de 3-ciclos é linear, e o número de ciclos diminui, no pior caso, em cada terceita iteração. Na Linha 24, a busca por uma 4/3- ou uma 3/2-sequência é feita em tempo constante, já que o número de ciclos é limitado por uma constante. As Linhas 24 e 25 podem também ser implementadas em $O(n \log n)$, de acordo com [33]. □

Capítulo 3

Permutação mais próxima por movimento de blocos e por pontos de quebra são NP-completos

Um problema mais geral do que o de ordenação é o de encontrar um objeto que esteja a uma distância limitada por um valor determinado de outros objetos dados de entrada. O problema de encontrar o objeto mais próximo foi estudado inicialmente considerando *cadeias de caracteres* (*strings*). Determinar a cadeia de caracteres mais próxima pela distância de Hamming (Hamming–CMP) foi provado por Lanctot *et al.* [45] ser NP-completo, mesmo no caso de alfabeto binário.

Permutações são cadeias de caracteres restritas, já que cada caracter do alfabeto aparece exatamente uma vez. Portanto, o problema de determinar a permutação mais próxima é uma restrição natural. Este problema já foi estudado considerando distância de Cayley por Popov [51], tendo provado que determinar a permutação mais próxima pela distância de Cayley (Cayley–PMP) é NP-completo.

O problema de determinar a permutação mais próxima não foi estudado ao considerar outras métricas de distância que sejam polinomiais, como por exemplo movimentos de blocos ou por pontos de quebra.

Neste capítulo, consideramos portanto o problema de determinar a permutação mais próxima em relação a estas duas métricas de distância, e provamos que Movimento de blocos–PMP e Pontos de quebra–PMP são NP-completos.

Resultados deste capítulo foram apresentados na seguinte conferência:

- 18th Latin-Iberoamerican Conference on Operations Research 2016 [26];

O manuscrito referente a este assunto, a ser submetido ao *Annals of Operations Research* [22], está no Anexo B.

Este capítulo está organizado da seguinte forma: na Seção 3.1 definimos os problemas de determinar a cadeia de caracteres mais próxima e de determinar a

permutação mais próxima, além de revermos as métricas de distância que tratamos neste capítulo; nas seções subsequentes provamos NP-completudes, onde na Seção 3.2 provamos que permutação mais próxima por trocas de blocos é NP-completo e na Seção 3.3 provamos que permutação mais próxima por pontos de quebra é NP-completo; após isto, na Seção 3.4 discutimos sobre demais problemas relacionados.

3.1 Permutações mais próximas

Um *alfabeto* Σ é um conjunto não vazio de *caracteres*, e uma *cadeia de caracteres* de Σ é uma sequência de caracteres de Σ. A *distância de Hamming de duas cadeias de caracteres* s e σ, denotada por $d_H(s, \sigma)$, é definida pelo número de posições com elementos distintos entre s e σ. Dizemos que a *distância de Hamming de uma cadeia* s, denotada por $d_H(s)$, é a distância de Hamming de s e $\iota = 0^m$.

O problema de determinar a existência da cadeia de caracteres mais próxima pela distância de Hamming é definido como segue:

CADEIA MAIS PRÓXIMA PELA DISTÂNCIA DE HAMMING (H–CMP) ENTRADA: Conjunto de cadeias de caracteres $\{s_1, s_2, \ldots, s_\ell\}$ de tamanho m cada cadeia do alfabeto Σ, e um inteiro não negativo f. PERGUNTA: Existe uma cadeia de caracteres σ de tamanho m tal que $\max\limits_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$?

No caso de resposta positiva para H–CMP, dizemos que uma *solução de H–CMP* é alguma cadeia σ que satisfaça $\max\limits_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$. Lanctot *et al.* [45] mostraram a complexidade computacional deste problema.

Teorema 3.1 *[45] H–CMP é* NP*-completo para cadeias binário.*

Uma permutação de tamanho n é uma cadeia de caracteres particular, já que é uma bijeção do conjunto $\{1, 2, \ldots, n\}$ neste mesmo conjunto.

A *união* de duas permutações α e β de tamanhos n e m, respectivamente, é a permutação π construída pela justaposição de α e β, $\pi = [\mathbf{0}\ \alpha(1)\ \alpha(2)\ \ldots \alpha(n)\ \beta(1)+n\ \beta(2)+n\ \ldots\ \beta(m)+n]$. Por exemplo, a $[\mathbf{0}\ 1\ 2\ 3\ 4\ 7\ 6\ 5\ 8\ \mathbf{13}]$ é a união de $[\mathbf{0}\ 1\ 2\ 3\ 4\ \mathbf{5}]$ e $[\mathbf{0}\ 3\ 2\ 1\ 4\ \mathbf{5}]$.

Dadas uma métrica M e $d_M(p, \pi)$ a distância entre p e π pela operação M, a *distância da permutação* π de tamanho m denotada por $d_M(\pi)$ é a distância de π e a permutação identidade $\iota = [\mathbf{0}\ 1\ 2\ \cdots\ n\ \boldsymbol{n}\ +\ \mathbf{1}]$.

A Permutação mais Próxima é definida por:

PERMUTAÇÃO MAIS PRÓXIMA PELA DISTÂNCIA M (M–PMP)
ENTRADA: Conjunto de permutações $\{p_1, p_2, \ldots, p_k\}$ em que cada permutação possui tamanho n, e um inteiro não negativo d.
PERGUNTA: Existe uma permutação π de tamanho n tal que $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$?

No caso de resposta positiva para M–PMP, dizemos que uma *solução de M–PMP* é alguma permutação π que satisfaça $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$.

Dado um conjunto de permutações, o problema de encontrar a permutação mais próxima objetiva encontrar a solução que minimize a maior das distâncias da permutação solução e cada uma das permutações da entrada. A métrica de distâncias depende do contexto do problema.

Se o problema de ordenação de uma determinada operação for NP-completo, então o problema de determinar a permutação mais próxima para a mesma operação é também NP-completo, de modo que ao considerar um conjunto unitário de entrada de PMP, temos o problema de ordenação de uma permutação. Neste caso, é interessante considerarmos a PMP relacionada aos problemas de ordenação com complexidade computacional polinomial ou em aberto.

Uma condição necessária para que exista uma solução é dada pela desigualdade triangular. Se existe uma permutação (ou cadeia de caracteres) solução com distância no máximo d (ou f) para cada permutação (ou cadeia de caracteres) da entrada, então a distância entre cada par de permutações (ou cadeia de caracteres) da entrada é no máximo $2d$ (ou $2f$).

3.2 Movimento de blocos–PMP é NP-completo

A operação de *movimento de blocos* transforma uma permutação em outra pela troca de dois blocos de elementos, esta operação generaliza a transposição, já que uma transposição troca dois blocos consecutivos, e generaliza também a operação de Cayley, devido uma operação de Cayley trocar dois blocos (não necessariamente consecutivos) em que cada bloco possui um único elemento.

Como visto anteriomente na Seção 1.2 (página 9), operações que generalizam outras não implicam os correspondentes problemas de ordenação serem NP-completos caso os problemas de ordenação de operações particulares sejam NP-completos. Exemplos são as operações de movimento de blocos, cujo problema de ordenação é polinomial [9], enquanto ordenação por transposições é NP-completo [7].

A prova de NP-completude de Movimento de Blocos–PMP baseia-se pela redução de Hamming–CMP. Primeiramente, aplicamos o Algoritmo 7 que trans-

forma uma cadeia de caracteres binária arbitrária s de tamanho m em uma permutação particular λ_s de tamanho $2m$.

Algoritmo 7: $Permut_{MB}(s)$

Entrada: Cadeia binária s de tamanho m
Saída: Permutação λ_s

1 cada ocorrência do bit 0 na posição i corresponde aos elementos $2i-1$ e $2i$ nas posições $2i-1$ e $2i$, respectivamente.
2 cada ocorrência do bit 1 na posição i corresponde aos elementos $2i-1$ e $2i$ nas posições $2i$ e $2i-1$, respectivamente.

A Figura 3.1 ilustra a construção de uma permutação a partir de uma cadeia de caracteres binária em relação ao Algoritmo 7, com diagrama de realidade e desejo ilustrado na Figura 3.2.

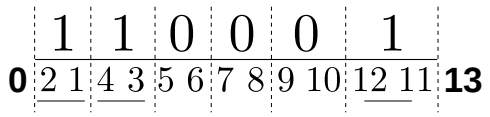

Figura 3.1: Permutação $\lambda_s = [\mathbf{0}\,2\,1\,4\,3\,5\,6\,7\,8\,9\,10\,12\,11\,\mathbf{13}]$ obtida pelo Algoritmo 7, onde $s = 110001$.

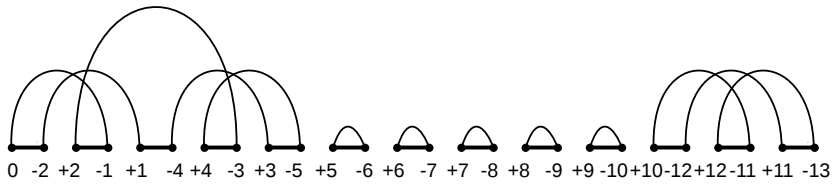

Figura 3.2: Diagrama de realidade e desejo de $[\mathbf{0}\,2\,1\,4\,3\,5\,6\,7\,8\,9\,10\,12\,11\,\mathbf{13}]$.

Lema 3.2 *Dadas uma cadeia de caracteres s de tamanho m e a permutação λ_s de tamanho $2m$ obtida pelo Algoritmo 7, então a permutação reduzida $gl(\lambda_s)$ possui tamanho n', tal que $2d_H(s) \leq n' \leq 3d_H(s) - 1$.*

Demonstração. Se a cadeia s é $s = 1^{d_H(s)}0^{m-d_H(s)}$ (ou $s = 0^{m-d_H(s)}1^{d_H(s)}$), então a permutação reduzida obtida possui menos $2(m - d_H(s))$ elementos. Assim, a permutação associada possui tamanho $n' = 2m - 2(m - d_H(s)) = 2d_H(s)$. Se s é $s = (01)^{\frac{m}{2}}$ (ou $s = (10)^{\frac{m}{2}}$), então cada adjacência $2i-1, 2i$ é removida na permutação reduzida, exceto a primeira adjacência $1, 2$ (ou a última adjacência $2m-1, 2m$), já que os dois elementos que formam a adjacência são removidas. Assim, o tamanho da permutação reduzida é $n' = 2d_H(s) + d_H(s) - 1$. □

Lema 3.3 *Se λ_s é uma permutação obtida pelo Algoritmo 7 e $gl(\lambda_s)$ é sua permutação reduzida de tamanho $2d_H(s) + x$, para $0 \leq x \leq d_H(s) - 1$, então $C(G(gl(\lambda_s))) = x + 1$.*

Demonstração. Existem x sequências contíguas de bits 0 em s, e uma sequência de bits 0 está entre duas sequências contíguas de bits 1. Isto implica em um ciclo no diagrama de realidade e desejo para cada sequência contígua de bits 1. □

A seguir, estabelecemos a igualdade entre a distância de Hamming de uma cadeia de caracteres da entrada de Hamming–CMP e a distância de movimentos de blocos da permutação correspondente obtida no Algoritmo 7.

Lema 3.4 *Dada a permutação λ_s obtida de uma cadeia de caracteres s pelo Algoritmo 7, a distância de movimento de blocos de λ_s é igual a distância de Hamming da cadeia s, $d_b(\lambda_s) = d_H(s)$.*

Demonstração. Sabendo que $d_b(\lambda_s) = d_b(gl(\lambda_s))$, temos pelo Lema 3.3 que $d_b(gl(\lambda_s)) = \frac{2d_H(s)+x+1-(x+1)}{2}$, implicando $d_b(\lambda_s) = d_H(s)$. □

Agora, mostraremos como uma solução para Hamming–CMP implica numa solução para Movimento de Blocos–PMP, e vice e versa.

Lema 3.5 *Dado um conjunto de k permutações obtidas pelo Algoritmo 7, existe uma permutação solução para* Movimento de Blocos–PMP *com distância de no máximo d se, e somente se, existe uma cadeia de caracteres solução para* Hamming – CMP *com distância de no máximo d.*

Esquema da demonstração. ($\Rightarrow$) "de permutação para cadeia de caracteres". Se λ' pode ser associada a alguma s' pelo Algoritmo 7, então, pelo Lema 3.4, s' é a cadeia mais próxima.

Caso contrário, buscamos na permutação solução da esquerda para a direita até encontrar a primeira posição onde o elemento correspondente é diferente do que poderia ser pelo algoritmo, este elemento pode estar em posições ímpares ou pares. Para cada uma, transformamos em uma nova permutação com maior prefixo de acordo com alguma saída do algoritmo, de forma a não aumentar a distância para nenhuma permutação da entrada.

Para garantir que a distância não aumenta, aplicamos transposições e mostramos que em cada caso, a pior operação possível é um 0-movimento em relação a qualquer permutação da entrada, isto devido a transposição aplicada afetar elementos de um mesmo ciclo no diagrama de realidade e desejo.

Ao repetir este processo, uma cadeia de caracteres de acordo com o algoritmo é encontrada e pelo Lema 3.4, uma cadeia com distância máxima igual a d é construída.

($\Leftarrow$) "de cadeia de caracteres para permutação". Dada uma cadeia de caracteres solução s, obtemos a permutação associada λ_s pelo Algoritmo 7. Pelo Lema 3.4

temos que a solução s em relação ao HAMMING–CMP corresponde a permutação λ_s com valor máximo de distância igual a d. □

Teorema 3.6 MOVIMENTO DE BLOCOS–PMP *é* NP*-completo.*

3.3 PONTOS DE QUEBRA–PMP é NP-completo

A *distância de pontos de quebra* é o número de elementos consecutivos em uma permutação que não são consecutivos em outra. Observe que nesta métrica nenhuma operação é aplicada de modo a transformar uma permutação em outra.

Similar a Seção 3.2, a prova de NP-completude de PONTOS DE QUEBRA–PMP baseia-se pela redução de HAMMING–CMP. Com isso, primeiramente aplicamos o Algoritmo 8 que transforma uma cadeia de caracteres binária arbitrária s de tamanho m em uma permutação particular β_s de tamanho $4m$.

Algoritmo 8: $Permut_{BP}(s)$

Entrada: Cadeia binária s de tamanho m.
Saída: Permutação β_s

1 cada ocorrência do bit 0 na posição i corresponde aos elementos $4i-3$, $4i-2$, $4i-1$ e $4i$ nas posições $4i-3$, $4i-2$, $4i-1$, $4i$, respectivamente.
2 cada ocorrência do bit 1 na posição i corresponde aos elementos $4i-3$, $4i-2$, $4i-1$ e $4i$ nas posições $4i-2$, $4i-3$, $4i-1$, $4i$, respectivamente.

A Figura 3.3 ilustra a construção de uma permutação a partir de uma cadeia de caracteres binária em relação ao Algoritmo 8.

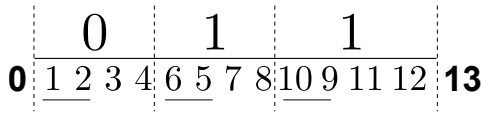

Figura 3.3: Permutação $\beta_s = [\mathbf{0}\ 1\ 2\ 3\ 4\ 6\ 5\ 7\ 8\ 10\ 9\ 11\ 12\ \mathbf{13}]$ obtida pelo Algoritmo 8, onde $s = 011$.

A seguir, estabelecemos a relação entre a distância de Hamming de uma cadeia de caracteres da entrada de H–CMP e a distância de pontos de quebra permutação correspondente obtida no Algoritmo 8.

Lema 3.7 *Dada a permutação β_s obtida de uma cadeia de caracteres s pelo Algoritmo 8, a distância de pontos de quebra de β_s é $d_{BP}(\beta_s) = 2d_H(s)$.*

Demonstração. Cada elemento 1 da cadeia de caracteres binária gera uma troca entre dois elementos consecutivos, deste modo criamos exatamente dois pontos de quebra. □

Agora, mostraremos como uma solução para Hamming–CMP implica numa solução para Pontos de Quebra–PMP, e vice e versa.

Lema 3.8 *Dado um conjunto de k permutações obtidas pelo Algoritmo 8, existe uma permutação solução para* Pontos de Quebra–PMP *com distância de no máximo $2d$ se, e somente se, existe uma cadeia de caracteres solução para* Hamming–CMP *com distância de no máximo d.*

Esquema da demonstração. ($\Rightarrow$) "de permutação para cadeia de caracteres". Se β' pode ser associada a alguma s' pelo Algoritmo 8, então, pelo Lema 3.7, s' é a cadeia mais próxima.

Caso contrário, buscamos na permutação solução da esquerda para a direita até encontrar a primeira posição onde o elemento correspondente é diferente do que poderia ser pelo algoritmo. Para cada uma, transformamos em uma nova permutação com maior prefixo de acordo com alguma saída do algoritmo, de forma a não aumentar a distância para nenhuma permutação. Ao repetir este processo, uma cadeia de caracteres de acordo com o algoritmo é encontrada e pelo Lema 3.7, uma cadeia com distância máxima igual a d é construída.

($\Leftarrow$) "de cadeia de caracteres para permutação". Dada uma cadeia de caracteres solução s, obtemos a permutação associada β_s pelo Algoritmo 8. Pelo Lema 3.7, temos que a solução s em relação ao Hamming–CMP corresponde a permutação β_s com valor máximo de distância igual a $2d$. $\square$

Teorema 3.9 Pontos de quebra–PMP *é* NP*-completo.*

3.4 Outros problemas relacionados

Além de termos provado NP-completudes das métricas de movimento de blocos e de pontos de quebra, veremos no capítulo a seguir que PMP é também NP-completo para movimento de blocos curtos. Para isso, veremos com mais detalhes propriedades dessa métrica em relação ao grafo de permutação.

Como visto no início deste capítulo (página 35), uma condição necessária para haver solução é obtida da desigualdade triangular. Assim, caso essa condição seja satisfeita, qualquer permutação da entrada é uma 2-aproximação para a solução ótima. Portanto, uma pergunta interessante é saber se é possível desenvolver um algoritmo polinomial com melhor razão de aproximação.

Um problema natural relacionado ao da centralidade é o da mediana. Neste problema, temos de entrada um conjunto de objetos (cadeias de caracteres ou permutações), uma métrica e um inteiro. Perguntamos se há algum objeto cuja soma das distâncias deste objeto a cada um dos objetos da entrada seja no máximo o inteiro.

O problema da mediana em relação a distância de Hamming em cadeias de caracteres é polinomial. Um algoritmo guloso para se obter a mediana se dá após alinhar todas as cadeia da entrada, escolher para cada posição da solução o bit mais frequente dentre os da entrada.

Apesar do problema da mediana ser simples de se resolver quando consideramos cadeias de caracteres em relação a distância de Hamming, o mesmo não parece ser ao tratar de permutações. Este problema permanece em aberto em relação a distância de Cayley, mesmo quando a entrada é composta por três permutações. Quando consideramos a distância de pontos de quebra, Pe'er e Shamir [50] provaram que este problema é NP-completo, o mesmo vale para transposição [2]. A mediana em relação a transposição foi estudada antes de temos que seu problema de ordenação fosse NP-completo. Uma pergunta natural que surge é: qual a complexidade computacional dos demais problemas ainda não estudados?

Capítulo 4

Outros problemas de ordenação e trabalhos futuros

Vimos no Capítulo 1 relações entre operações em permutações, algumas podem ser vistas por generalizações de outras. Neste capítulo destacamos operações de *movimento de blocos curtos* e de *multi corte restrito*, que são vistas por restrição e por generalização da operação de transposição, respectivamente. Além de tratarmos destes dois problemas de forma mais concisa (de modo que indicamos os Anexos C e D para maiores detalhes e mais informações), concluímos esta tese com as questões em aberto tanto para os problemas deste capítulo quanto para os demais apresentados anteriormente.

Resultados sobre estes assuntos foram apresentados nas conferências:

- Movimento de blocos curtos:
  - International Colloquium on Graph Theory and Combinatorics 2014 [23];
  - Latin American Workshop on Cliques in Graphs 2016 [27].

  O manuscrito referente a este assunto está no Anexo C.

- Multi corte restrito:
  - Latin American Workshop on Cliques in Graphs 2014 [25];
  - Cologne-Twente Workshop on Graphs & Combinatorial Optimization 2015 [24].

O manuscrito referente a este assunto está no Anexo D.

Este capítulo está organizado da seguinte forma: Na Seção 4.1, sobre o movimento de blocos curtos, destacamos alguns resultados obtidos a cerca dos problemas de ordenação e de permutação mais próxima. Sobre o problema de ordenação, identificamos classes tratáveis de permutações já estudadas no problema de ordenação

por transposição, mesmo ao considerar suas permutações não-reduzidas; identificamos relações de equivalência para permutações, tais que permutações equivalentes possuem mesma distância de movimento de blocos curtos; mostramos que para qualquer permutação, existe uma sequência ótima de movimentos de blocos curtos que é obtida pela ordenação de cada componente conexa separadamente do grafo de permutação. Sobre o problema de permutação mais próxima, provamos que em relação à distância de movimento de blocos curtos, este problema é NP-completo. Na Seção 4.2, sobre o movimento de multi corte restrito, estudamos duas variações, uma onde o k é fixo dado de entrada, e outra onde k é arbitrário. Além dos resultados sobre limite de distância apresentados na Seção 1.3 (páginas 13, 14 e 14), tratamos do problema para $k = 1$, onde propomos um limite superior em função do número de *ciclos algébricos*, este limite é justo para algumas classes de permutações; estudamos ainda limites superiores, onde conseguimos valores de $3n/4$, $2n/3$, e deixamos a conjectura do limite superior de $n/2$; quando k é arbitrário, consideramos também o problema do diâmetro, onde mostramos o limite inferior de $\Omega(\log n)$, e o limite superior de $O(\log^2 n)$. Na Seção 4.3 concluímos esta tese, apresentando também os problemas em aberto e os desafios correntes para cada um dos assuntos tratados nesta tese.

4.1 Movimento de blocos curtos

Vista a dificuldade de alguns problemas de ordenação, outros estudos surgem por formas restritas de operações. Uma das restrições da operação de transposição é o problema de determinar a distância de uma dada permutação de tal forma que a soma de elementos nos blocos numa operação seja limitada por alguma constante.

Quando limitamos a soma dos elementos nos blocos por alguma função de n não constante, seu correspondente problema de ordenação é NP-completo, havendo uma redução do problema de ordenação por transposições [42]. Porém, quando limitamos por alguma constante, o problema de ordenação permanece em aberto, conhecido polinomial apenas quando a soma dos elementos é igual a 2, onde o *bubble sort* pode ser aplicado para ordenar a permutação.

Heath e Vergara [42] propuseram a limitação dos blocos por 3, esta é a operação de *movimento de blocos curtos*.

4.1.1 Permutações não-reduzidas

Christie [9] mostrou que $d_t(\pi) = d_t(gl(\pi))$, porém esta propriedade não vale para distância de movimento de blocos curtos. Considere por exemplo $\pi = [2\,1] = gl(\sigma = [3\,4\,1\,2])$: $d_{sbm}(\pi) = 1$, enquanto $d_{sbm}(\sigma) = 2$. A seguir, consideramos permutações

cujas reduzidas são uma permutação reversa, ou uma α-permutação, duas classes bastante estudadas no contexto de transposições.

Não-reduzida da reversa Heath e Vergara [41] computaram a distância da permutação reversa, $\rho_{[n]} = [n\, n-1\, \cdots\, 2\, 1]$. Uma *permutação não-reduzida da reversa* é a permutação π onde $gl(\pi) = \rho_{[n]}$. Se π é um permutação não-reduzida da reversa, então $PG(\pi)$ é um *grafo k-partido completo.*

Heath e Vergara [41] mostraram que o limite inferior da Equação (1.2) (página 12) é justo para permutações que possuem grafos de permutação bipartidos, além disso uma sequência ótima para ordenar pode ser obtida dos vértices emparelhados e não emparelhados. Mostramos que isto também vale para permutações não-reduzidas da reversa.

Teorema 4.1 *Se $\pi_{[n]}$ é uma permutação não-reduzida da reversa, então*

$$d_{sbm}(\pi_{[n]}) = |M| + |U|,$$

onde M é um emparelhamento máximo e U é o conjunto dos vértices não emparelhados de $A^a_{\pi_{[n]}}$.

Esquema da demonstração. Analisamos todos os casos de paridade dos tamanhos dos blocos de adjacência. Para cada um, obtemos uma sequência que ordena a permutação com quantidade igual ao limite inferior da Equação (1.2). □

α^k-permutação Labarre [44] propôs as α-permutações no contexto de transposições. Uma permutação π de tamanho n é uma α-permutação se todo os elementos pares estiverem em posições corretas e todos os outros elementos ímpares formam um único ciclo ou crescente ou decrescente no grafo $\Gamma(\pi) = (V, E)$, que é um grafo direcionadao, onde $V = \{1, \ldots, n\}$, e $E = \{(i, \pi_i) | i = 1, \ldots, n\}$.

A seguir, propomos uma classe que generaliza a classe das α-permutações.

Definição 4.2 *Uma permutação π de tamanho n é uma* α^k-permutação *se: todos os elementos pares estiverem em posições corretas; existem k elementos ímpares em posições corretas; e outros elementos ímpares formam um único ciclo ou crescente ou decrescente em $\Gamma(\pi)$.*

Se $k = 0$, então α-permutação e α^k-permutação coincidem. Uma α^k-permutação, para $k > 0$, é uma permutação não-reduzida de uma α-permutação, porém existem permutações não-reduzidas de α-permutações que não são α^k-permutações, por exemplo, $\pi = [3\ 2\ 5\ 6\ 4\ 1]$, que é uma não-reduzida da α-permutação $gl(\pi) = [3\ 2\ 5\ 4\ 1]$, porém π não é uma α^k-permutação.

Teorema 4.3 *Para cada α^k-permutação $\pi_{[n]}$, temos $d_{sbm}(\pi_{[n]}) = \left\lceil \frac{|E^p_{\pi_{[n]}}|}{2} \right\rceil$, em que $|E^p_{\pi_{[n]}}|$ é o número de inversões de $\pi_{[n]}$.*

Esquema da demonstração. Analisamos todos os casos de paridade dos tamanhos dos blocos de adjacência, e obtemos pulos em cada um dos casos. □

4.1.2 Relações de equivalência em operações por movimento de blocos limitados

Encontramos algumas relações de equivalência em que ao considerarmos transposições, duas permutações equivalentes possuem mesma distância. Já para operações de movimento de blocos limitados, ao considerar a relação tórica isso nem sempre acontece, porém ao considerar a equivalência por complemento reverso, duas permutações equivalentes possuem mesma distância de movimento de blocos limitados.

O estudo de relações de equivalência neste contexto é interessante pelo seguinte fato: sejam duas permutações π e σ equivalentes por complemento reverso, porém π e σ não são toricamente equivalentes. Isto implica que ao englobar as classes de equivalência tórica correspondentes a π e a σ, todas as permutações nestas classes possuirão mesma distância.

Relação tórica A *circularização* de uma permutação π é a permutação circular $\pi^\circ = (0\ \pi_1 \cdots \pi_n)$. O *q-passo cíclico* é a permutação circular $q + \pi^\circ = (\overline{q}\ \overline{q+\pi_1} \cdots \overline{q+\pi_n})$, em que $\overline{x}$ é o resto de divisão de x por $n+1$. Assim, duas permutações π e σ são *toricamente equivalentes* se $\sigma^\circ \equiv q + \pi^\circ$ para algum inteiro q. Por exemplo, a permutação [3 4 1 2], é toricamente equivalente a, [2 3 1 4], [1 3 4 2], [3 1 2 4] e [1 4 2 3].

Eriksson *et al.* [30] introduziram equivalência tórica no contexto de transposições. Se π e σ são toricamente equivalentes, então $d_t(\pi) = d_t(\sigma)$. Já ao considerar movimento de blocos curtos, isto não vale. Por exemplo, $\pi = [1\,2\,4\,3]$ e $\sigma = [4\,1\,2\,3]$, temos que $d_{sbm}(\pi) = 1$, porém $d_{sbm}(\sigma) = 2$.

Apesar da equivalência tórica não preservar a distância de movimento de blocos curtos, quando restringimos a distâncias de movimento de blocos limitados, de modo que um *movimento de blocos limitados por p* é uma transposição $t(i, j, k)$, tal que $k - i \leq p$, temos que:

Proposição 4.4 *Dados π e σ duas permutações toricamente equivalentes. Se $p > \frac{n+1}{2}$, então $d_{pbbm}(\pi) = d_{pbbm}(\sigma)$, onde d_{pbbm} é a distância por movimento de blocos limitados por p.*

Esquema da demonstração. Dada uma permutação π, podemos transformar uma operação de movimento de blocos limitados por p em uma outra operação toricamente equivalente a σ. Mostramos se $p > \frac{n+1}{2}$, isto é sempre possível de ser obtido. $\square$

Complemento reverso O *complemento reverso* de $\pi_{[n]}$ é a permutação $[n+1-\pi_n \quad n+1-\pi_{n-1} \quad \cdots \quad n+1-\pi_1]$, que é o produto $\pi^\rho = \rho\pi\rho^{-1} = \rho\pi\rho$, onde $\rho_{[n]} = [n \; n{-}1 \cdots 1]$ é a permutação reversa.

Dentre os resultados sobre equivalência de complemento reverso, destacamos:

Proposição 4.5 *Dada uma permutação π, temos que $d_{pbbm}(\pi) = d_{pbbm}(\pi^\rho)$.*

Esquema da demonstração. Mostramos, assim como na Proposição 4.4, que dada uma permutação π, podemos transformar uma operação de movimento de blocos limitados por p em uma outra para a permutação π^ρ. $\square$

Uma pergunta natural é se o produto $\sigma\pi\sigma^{-1}$, tal que $\sigma \neq \rho$, possui mesma distância que π. A seguir, temos que isto vale somente para $\sigma \in \{\iota, \rho\}$.

Proposição 4.6 *Sejam $p \geq 2 \in \mathbb{N}$ e $\sigma \in S_n$, tal que $\pi \in S_n$. Se $d_{pbbm}(\pi) = d_{pbbm}(\pi^\sigma)$, então $\sigma \in \{\iota, \rho\}$.*

Esquema da demonstração. O produto $\sigma\pi\sigma^{-1}$, para $\sigma \in \{\iota, \rho\}$, satisfaz a distância. Mostramos para qualquer outro σ, existe um movimento de bloco limitado por p, tal que sua operação inversa não é um movimento de blocos limitados por p. $\square$

4.1.3 Componentes conexas no grafo de permutação e Movimento de blocos curtos–PMP

Mostramos que em relação à operação de movimento de blocos curtos há uma propriedade de modo que em outras operações de movimento de blocos limitados esta propriedade não é preservada.

Teorema 4.7 *Dada uma permutação π, existe uma ordenação ótima por movimento de blocos curtos tal que cada componente conexa de π pode ser ordenada separadamente.*

É interessante observar que esta propridade não vale para movimento de blocos limitados por $p > 3$. Por exemplo, para $p = 4$, seja $[3\ 2\ 1\ 6\ 5\ 4]$ cuja ordenação de cada componente separadamente nos leva a 4 operações, porém algo melhor pode ser obtido: $[3\ 2\ \underline{1\ 6}\ \underline{5\ 4}] \to [3\ \underline{2\ 5}\ \underline{4\ 1}\ 6] \to [\underline{3\ 4}\ \underline{1\ 2}\ 5\ 6] \to \iota$.

Em relação a operação de transposição, juntar componentes pode nos levar a melhores valores de distância, conforme mostramos em [11, 18].

Apesar da complexidade computacional do problema de ordenação por movimento de blocos curtos continuar em aberto, uma questão interessante é investigar como se comporta o problema mais geral de PMP. De modo que concluímos no Teorema 4.10 que Movimento de blocos curtos–PMP é NP-completo.

Primeiramente, aplicamos o Algoritmo 9 que transforma uma cadeia de caracteres binária s de tamanho m em uma permutação particular λ_s de tamanho $2m$.

Algoritmo 9: $Permut_{SBM}(s)$

Entrada: Cadeia binária s de tamanho m
Saída: Permutação λ_s

1 cada ocorrência do bit 0 na posição i corresponde aos elementos $2i-1$ e $2i$ nas posições $2i-1$ e $2i$, respectivamente.
2 cada ocorrência do bit 1 na posição i corresponde aos elementos $2i-1$ e $2i$ nas posições $2i$ e $2i-1$, respectivamente.

Lema 4.8 *Dada uma cadeia s de tamanho m e a permutação λ_s de tamanho $2m$ obtida no Algoritmo 7, temos que $d_{sbm}(\lambda_s) = d_H(s)$.*

Demonstração. Pelo Teorema 4.7, cada componente conexa pode ser ordenada separadamente, e cada bit 1 corresponde a uma inversão. □

Lema 4.9 *Dado um conjunto de k permutações obtidas pelo Algoritmo 9, existe uma permutação solução para* Movimento de blocos curtos–PMP *com distância de no máximo d se, e somente se, existe uma cadeia de caracteres solução para* Hamming–CMP *com distância de no máximo d.*

Esquema da demonstração. ($\Rightarrow$) "de permutação para cadeia de caracteres". Se λ' pode ser associada a alguma s' pelo Algoritmo 9, então, pelo Lema 4.8, s' é a cadeia mais próxima.

Caso contrário, buscamos na permutação solução da esquerda para a direita até encontrar a primeira posição onde o elemento correspondente é diferente do que poderia ser pelo algoritmo. Cada um dos elementos a partir desta posição até onde está o elemento que deveria pelo algoritmo forma uma inversão. Assim, transformamos em uma nova solução de modo que a distância entre a nova solução e cada uma das permutações da entrada não aumenta.

($\Leftarrow$) "de cadeia de caracteres para permutação". Dada uma cadeia de caracteres solução s, obtemos a permutação associada λ_s pelo Algoritmo 9. Pelo Lema 4.8, temos que a solução s em relação ao Hamming–CMP corresponde a permutação β_s com valor máximo de distância igual a d. □

Teorema 4.10 Movimento de blocos curtos–PMP *é* NP*-completo.*

4.2 Multi corte restrito

Na Seção 4.1 vimos um estudo de uma operação mais restrita comparada ao problema de transposição, temos agora um estudo de uma operação mais geral. Existem diversas formas de generalizações, algumas das quais tornam o problema de ordenação polinomial, alguns exemplos são operações de *double-cut-and-join* [54] e de *single-cut-or-join* [32].

Alekseyev e Pevzner [1] propuseram uma generalização da operação double-cut-and-join, chamando a operação de *multi corte*, onde muito pouco se sabe sobre esta operação. Propomos uma versão mais restrita desta última operação de Alekseyev e Pevzner, a qual chamamos de *multi quebra restrita*. Apesar desta operação ser uma restrição do *multi corte*, ainda é uma generalização da operação de movimento de blocos, onde o problema de ordenação é polinomial [10], e das operações de transposição e de reversão, problemas estes que são NP-completos [7, 8].

Uma operação de k *multi quebra restrita* pode ser vista como uma reversão num bloco de elementos tal que há no máximo k blocos internos de elementos não-reversíveis. Estudamos duas variações deste problema: uma onde o k é fixo dado de entrada, e outra onde k é arbitrário. Além dos limites para distâncias que mencionamos na Seção 1.3, apresentamos a seguir os demais resultados obtidos sobre esta operação.

4.2.1 Limites superiores, $k = 1$

Quando tratamos do problema onde $k = 1$, propomos um limite superior em função do número de *ciclos algébricos*, este limite é justo para algumas classes de permutações.

Permutações podem ser representadas por cada elemento seguido por sua imagem. Por exemplo, $\{1, 2, 3\}$, $(1\,2\,3)$ mapeia 1 em 2, 2 em 3, e 3 em 1, correspondendo a permutação $[\mathbf{0}\,2\,3\,1\,\mathbf{4}]$. Esta representação não é única; $(2\,3\,1)$ e $(3\,1\,2)$ são equivalente. Permutações são compostas por um ou mais ciclos algébricos. Por exemplo, $\pi = [\mathbf{0}\,8\,5\,1\,3\,2\,7\,6\,4\,\mathbf{9}] = (1\,8\,4\,3)(2\,5)(6\,7)$ possui 3 ciclos algébricos. Denotamos $pc(\pi)$ pelo número de ciclos algébricos de π.

Lema 4.11 *Dada uma permutação π com n elementos e $pc(\pi)$ seu número de ciclos algébricos, $d_{1rmb}(\pi) \leq n - pc(\pi)$.*

Mostramos que este limite é justo para a classe das *permutações de involuções estrelas*. Uma *permutação involução* é tal que todos os ciclos algébricos possuem tamanho até 2. Uma permutação é *involução estrela* se: cada elemento ímpar $2i+1$ está na posição correta, para $1 \leq i \leq \lfloor n/2 \rfloor$, $\pi_{2i+1} = 2i+1$; e todo os outros

elementos formam ciclos de tamanho 2, $(a\ a')$ tal que $a' > a + 2$. Por exemplo $[\mathbf{0}\,9\,6\,3\,8\,5\,2\,7\,4\,1\,\mathbf{10}]$ é uma permutação involução estrela.

Teorema 4.12 *Dada uma permutação involução estrela π, temos que $d_{1rmb}(\pi) = \frac{b(\pi)}{4}$.*

Esquema da demonstração. Já que $\frac{b(\pi)}{4}$ é um limite inferior (Teorema 1.7, página 14), mostramos que $\frac{b(\pi)}{4} = n - pc(\pi)$. □

Outro estudo que damos atenção é o de determinar o diâmetro. Ao considerar a operação de reversão, Hannenhalli e Pevzner [38] provaram que as *permutações Gollan* são diametrais, cujo valor é $n - 1$. Isto devido ao limite inferior de $n - 1$ da operação de reversão das permutações Gollan ser igual ao limite superior de $n - 1$ para ordenar qualquer permutação por reversões.

Definição 4.13 *[38] Uma* permutação Gollan *de tamanho n é:*

- $[\mathbf{0}\ 3\,1\,5\,2\,7\,4 \cdots n{-}3\,n{-}5\,n{-}1\,n{-}4\,n\,n{-}2\,\boldsymbol{n{+}1}]$, *se n é par;*
- $[\mathbf{0}\ 3\,1\,5\,2\,7\,4 \cdots n{-}6\,n{-}2\,n{-}5\,n\,n{-}3\,n{-}1\,\boldsymbol{n{+}1}]$, *se n é ímpar.*

Teorema 4.14 *Dada uma permutação Gollan π de tamanho n, temos que $d_{1\varpi}(\pi) = \lfloor \frac{n}{2} \rfloor$.*

Esquema da demonstração. Obtemos uma sequência ordenante cujo número de operações é igual ao limite inferior do Teorema 1.9. □

Teorema 4.15 *O diâmetro de 1ϖ é pelo menos $\lfloor \frac{n}{2} \rfloor$.*

Apresentamos também limites superiores somente em função de n.

Lema 4.16 *Dada uma permutação π com n elementos, $d_{1rmb}(\pi) \leq \frac{3n}{4}$.*

Esquema da demonstração. Mostramos estratégia recursiva no número de elementos da permutação. Dada uma permutação, conseguimos reduzir 3 elementos da permutação numa sequência de 4 operações. □

Lema 4.17 *Dada uma permutação π com n elementos, $d_{1rmb}(\pi) \leq \frac{2n}{3}$.*

Esquema da demonstração. Similar ao Lema 4.17, mostramos estratégia recursiva no número de elementos da permutação. Dada uma permutação, conseguimos reduzir 2 elementos da permutação numa sequência de 3 operações. □

Além dos limites superiores apresentados nos Lemas 4.16 e 4.17, implementamos um algoritmo exato de força bruta [13] para o cálculo de distância para todas as $n!$ permutações com n elementos. Verificamos que $\lfloor n/2 \rfloor$ é o valor do diâmetro para todo $n \leq 11$, conforme visto na Tabela 4.1.

Caso este limite de $\lfloor n/2 \rfloor$ seja verdadeiro, teremos que as permutações diametrais para 1ϖ são as *permutações Gollan*, as mesmas diametrais para o problema do diâmetro de reversão.

n	2	3	4	5	6	7	8	9	10	11
$\mathbf{D_0(n)}$	**1**	2	**3**	4	**5**	6	**7**	8	**9**	10
$\mathbf{D_1(n)}$	**1**	**1**	2	2	**3**	**3**	4	4	**5**	**5**
$\mathbf{D_2(n)}$	**1**	**1**	2	2	**3**	**3**	**3**	**3**	4	4

Tabela 4.1: Valores do diâmetro $D_k(n)$ para $k = 0, 1$ e 2, e $n \leq 11$.

4.2.2 Limites superiores, k arbitrário

Note que o número de ϖ operações possíveis é exponencial, a seguir vemos que este valor é limitado por $O(2^n)$.

Lema 4.18 *Toda permutação de tamanho n, o número de ϖ operações possíveis é $O(2^n)$.*

Esquema da demonstração. Obtemos a seguinte fórmula recorrente no número de possíveis operações para uma permutação de tamanho n: $T(n) = 2T(n-1) - T(n-2) + 2^n - 2$, onde $T(1) = 0$ e $T(2) = 1$. □

A seguir, temos o seguinte limite inferior logaritmico para o diâmetro de ϖ.

Teorema 4.19 *O diâmetro de ϖ é $D_\varpi(n) = \Omega(\log n)$.*

Demonstração. Dada uma permutação, pelo Lemma 4.18, existem $O(2^n)$ permutações com distâncias iguais a 1. O número de permutações de tamanho n é $n! \approx 2^{n \log n}$. Portanto, de uma permutação, o número de permutações alcançáveis por d operações é no máximo 2nd, portanto o diâmetro é $d \geq \log n$. □

A seguir, temos um limite superior para o diâmetro de ϖ.

Teorema 4.20 *O diâmetro de ϖ é $D_\varpi(n) = O(\log^2 n)$.*

Esquema da demonstração. Desenvolvemos estratégia recursiva que ordena uma permutação qualquer com $O(\log^2 n)$ operações de ϖ. □

4.3 Trabalhos futuros

Em cada um dos assuntos tratados nesta tese, temos algumas questões interessantes a serem investigadas.

- Sobre a operação de transposições:
  - Desenvolvemos um algoritmo com melhor razão de aproximação e melhor complexidade, porém ainda não temos uma prova de APX-dificuldade para o problema de ordenação por transposições. Portanto, podemos buscar por tal redução. Sabemos que o problema de ordenação por reversão sem sinal é APX-difícil [5];

- Ainda em relação a operação de transposição, temos uma estratégia que tornará possível aumentar o diâmetro de transposição, contudo hoje necessitamos de viabilidade computacional para encontrar permutações candidatas que satisfaçam algumas propriedades. Uma questão interessante é obter outras propriedades algébricas que envolvam a operação de união e assim consigamos tais candidatas.

- Sobre permutações mais próximas:
  - Buscamos desenvolver algoritmos FPT (*fixed-parameter tractable*) [28] para estes problemas, assim como é conhecido para *strings* considerando a distância de Hamming [37];
  - Buscamos estudar o problema relacionado de encontrar a permutação *mediana* de um conjunto de entrada. Uma permutação é a mediana quando a soma das distâncias da solução para cada uma da entrada é limitada superiormente por algum inteiro. Este problema já foi estudado considerando algumas outras métricas [38].

- Sobre a operação de troca de blocos curtos:
  - Buscamos explorar a complexidade computacional do problema de ordenação. Este problema, apesar de ser uma generalização natural da ordenação pelo *bubble sort*, parece ser tão difícil quanto o problema de transposição. Pensamos em investigar classes de permutações cujos grafos de permutação são 2-conexos em arestas.

- Sobre a operação de multi corte restrito:
  - Temos como objetivo principal determinar o diâmetro para 1ϖ. Temos a conjectura de que o diâmetro é $n/2$, isto devido aos experimentos computacionais e também as candidatas a serem diametrais serem as permutações Gollan, as mesmas diametrais para o problema de reversão.

Referências Bibliográficas

[1] Alekseyev, M. A., Pevzner, P. A., 2008, "Multi-break rearrangements and chromosomal evolution", *Theoretical Computer Science*, v. 395, n. 2, pp. 193–202.

[2] Bader, M., 2011, "The transposition median problem is NP-complete", *Theoretical Computer Science*, v. 412, n. 12-14, pp. 1099–1110.

[3] Bafna, V., Pevzner, P. A., 1998, "Sorting by transpositions", *SIAM J. Discrete Math.*, v. 11, n. 2, pp. 224–240.

[4] Bafna, V., Pevzner, P. A., 1993, "Genome Rearrangements and Sorting by Reversals". In: *Proceedings of 34th Annual Symposium on Foundations of Computer Science*, FOCS, pp. 148–157.

[5] Berman, P., Karpinski, M., 1999, "On some tighter inapproximability results". In: *International Colloquium on Automata, Languages, and Programming*, pp. 200–209. Springer.

[6] Bulteau, L., Fertin, G., Rusu, I., 2011, "Pancake flipping is hard", *CoRR*, v. abs/1111.0434.

[7] Bulteau, L., Fertin, G., Rusu, I., 2012, "Sorting by transpositions is difficult", *SIAM J. Discrete Math.*, v. 26, n. 3, pp. 1148–1180.

[8] Caprara, A., 1997, "Sorting by reversals is difficult". In: *Proceedings of the first annual international conference on computational molecular biology*, RECOMB, pp. 75–83. ACM.

[9] Christie, D. A., 1999, *Genome Rearrangement Problems*. Phd thesis, University of Glasgow, UK.

[10] Christie, D. A., 1996, "Sorting permutations by block-interchanges", *Information Processing Letters*, v. 60, n. 4, pp. 165–169.

[11] Cunha, L. F. I., 2013, *Limites Inferiores e Superiores para Rearranjo de Genomas por Transposições*. Dissertação de mestrado, UFRJ, Brasil.

[12] Cunha, L. F. I., Kowada, L. A. B., de A. Hausen, R., et al., 2014. `http://compscinet.org/research/sbt1375`, .

[13] Cunha, L., Kowada, L. A. B., de A. Hausen, R., et al., 2016. `http://www.cos.ufrj.br/~lfignacio/rmb`, .

[14] Cunha, L. F. I., Kowada, L. A. B., 2010, "Upper bounds and exact values on transposition distance of permutations". In: *4th Latin-American Workshop on Cliques in Graphs*, Matemática Contemporânea, pp. 77–84. SBM.

[15] Cunha, L. F. I., dos Santos, V. F., Kowada, L. A. B., et al., , "Sorting by short block-moves and the complexity of the short block-move closest permutation problem", Manuscrito a ser submetido no SIAM J. Discrete Math., .

[16] Cunha, L. F. I., Kowada, L. A. B., de A. Hausen, R., et al., , "On sorting permutations by restricted multi-break rearrangements", Manuscrito a ser submetido no SIAM J. Discrete Math., .

[17] Cunha, L. F. I., Kowada, L. A. B., de A. Hausen, R., et al., 2012, "Transposition diameter and lonely permutations". In: *Advances in Bioinformatics and Computational Biology*, pp. 1–12. LNBI, Springer-Verlag.

[18] Cunha, L. F. I., Kowada, L. A. B., de A. Hausen, R., et al., 2013, "Advancing the transposition distance and diameter through lonely permutations", *SIAM J. Discrete Math.*, v. 27, n. 4, pp. 1682–1709.

[19] Cunha, L. F. I., Kowada, L. A. B., Hausen, R. d. A., et al., 2013, "On the 1.375-Approximation Algorithm for Sorting by Transpositions in $O(n \log n)$ Time". In: *Brazilian Symposium on Bioinformatics*, pp. 126–135. Springer, .

[20] Cunha, L. F. I., Kowada, L. A. B., Hausen, R. d. A., et al., 2014, "A faster 1.375-approximation algorithm for sorting by transpositions". In: *Workshop Algorithms on Bioinformatics*, pp. 26–37. Springer, .

[21] Cunha, L. F. I., Kowada, L. A. B., Hausen, R. d. A., et al., 2015, "A Faster 1.375-Approximation Algorithm for Sorting by Transpositions*", *Journal of Computational Biology*, v. 22, n. 11, pp. 1044–1056.

[22] Cunha, L. F. I., dos Santos, V. F., Kowada, L. A. B., et al., 2017, "On the computational complexity of closest string problems", Manuscrito a ser submetido no Discrete Applied Mathematics.

[23] Cunha, L. F. I., Kowada, L. A. B., de Alencar Hausen, R., et al., 2014, "An update on sorting permutations by short block-moves". In: *International Colloquium on Graph Theory and Combinatorics*, .

[24] Cunha, L. F. I., Kowada, L. A. B., de Figueiredo, C. M. H., 2015, "Sorting separable permutations by multi-break rearrangements". In: *Cologne-Twente Workshop on Graphs & Combinatorial Optimization*, pp. 26–28, .

[25] Cunha, L. F. I., Kowada, L. A. B., de Figueiredo, C. M. H., 2015, "Sorting separable permutations by restricted multi-break rearrangements", *Matemática Contemporânea*, v. 44, pp. 1–10.

[26] Cunha, L. F. I., dos Santos, V. F., Kowada, L. A. B., et al., 2016, "The block-interchange and the breakpoint closest permutation problems are NP-Complete". In: *Latin-Iberoamerican Conference on Operations Research*, pp. 239–246, .

[27] Cunha, L. F. I., dos Santos, V. F., Kowada, L. A. B., et al., 2016, "The short block-move closest permutation problem is NP-complete". In: *Latin American Workshop on Cliques in Graphs*, p. 15, .

[28] Cygan, M., Fomin, F. V., Kowalik, L., et al., *Parameterized algorithms*, v. 4. Springer.

[29] Elias, I., Hartman, T., 2006, "A 1.375-approximation algorithm for sorting by transpositions", *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, v. 3, n. 4, pp. 369–379.

[30] Eriksson, H., Eriksson, K., Karlander, J., et al., 2001, "Sorting a bridge hand", *Discrete Math.*, v. 241, n. 1-3, pp. 289–300.

[31] Feião, P., 2012, *On Genome Rearrangement Models*. Tese de doutorado, UNICAMP, Brasil.

[32] Feijao, P., Meidanis, J., 2011, "SCJ: a breakpoint-like distance that simplifies several rearrangement problems", *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, v. 8, n. 5, pp. 1318–1329.

[33] Feng, J., Zhu, D., 2007, "Faster algorithms for sorting by transpositions and sorting by block interchanges", *ACM Transactions on Algorithms (TALG)*, v. 3, n. 3, pp. 25.

[34] Fertin, G., Labarre, A., Rusu, I., et al., 2009, *Combinatorics of Genome Rearrangements*. The MIT Press.

[35] Firoz, J. S., Hasan, M., Khan, A. Z., et al., 2011, "The 1.375 approximation algorithm for sorting by transpositions can run in o (n log n) time", *Journal of Computational Biology*, v. 18, n. 8, pp. 1007–1011.

[36] Gonçalves, A., 2012, *Introdução à Álgebra*. Coleção Projeto Euclides, IMPA.

[37] Gramm, J., Niedermeier, R., Rossmanith, P., 2003, "Fixed-parameter algorithms for closest string and related problems", *Algorithmica*, v. 37, n. 1, pp. 25–42.

[38] Hannenhalli, S., Pevzner, P., 1995, "Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals". In: *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC'95, pp. 178–189, New York, NY, USA. ACM.

[39] Hartman, T., Shamir, R., 2006, "A simpler and faster 1.5-approximation algorithm for sorting by transpositions", *Information and Computation*, v. 204, n. 2, pp. 275–290.

[40] Hausen, R. A., Faria, L., de Figueiredo, C. M. H., et al., 2010, "Unitary toric classes, the reality and desire diagram, and sorting by transpositions", *SIAM J. Discrete Math.*, v. 24, n. 3, pp. 792–807.

[41] Heath, L. S., Vergara, J. P. C., 1998, "Sorting by bounded block-moves", *Discrete Applied Mathematics*, v. 88, n. 1, pp. 181–206.

[42] Heath, L. S., Vergara, J. P. C., 2000, "Sorting by short block-moves", *Algorithmica*, v. 28, n. 3, pp. 323–352.

[43] Kowada, L. A. B., de A. Hausen, R., de Figueiredo, C. M. H., 2010, "Bounds on the transposition distance for lonely permutations". In: *Advances in Bioinformatics and Computational Biology*, pp. 35–46. LNBI, Springer-Verlag.

[44] Labarre, A., 2008, *Combinatorial Aspects of Genome Rearrangements and Haplotype Networks*. Tese de Doutorado, Universite Libre de Bruxelles.

[45] Lanctot, J. K., Li, M., Ma, B., et al., 2003, "Distinguishing string selection problems", *Information and Computation*, v. 185, n. 1, pp. 41–55.

[46] Lu, L., Yang, Y., 2010, "A lower bound on the transposition diameter", *SIAM J. Discrete Math.*, v. 24, n. 4, pp. 1242–1249.

[47] MacLane, S., Birkhoff, G., 1971, *Algebra*. The Macmillan Company.

[48] Meidanis, J., Walter, M. E. M. T., Dias, Z., 1997, "Transposition distance between a permutation and its reverse". In: *Proceedings of the 4th South American Workshop on String Processing*, pp. 70–79.

[49] Palmer, J. D., Herbon, L. A., 1998, "Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence", *Journal of Molecular Evolution*, v. 27, pp. 87–97.

[50] Pe'er, I., Shamir, R., 1998, "The median problems for breakpoints are NP-complete". In: *Elec. Colloq. on Comput. Complexity*, v. 71.

[51] Popov, V. Y., 2007, "Multiple genome rearrangement by swaps and by element duplications", *Theoretical Computer Science*, v. 385, n. 1, pp. 115–126.

[52] Setubal, J. C., Meidanis, J., 1997, *Introduction to Computational Molecular Biology*. PWS Publishing Company.

[53] Tannier, E., Zheng, C., Sankoff, D., 2009, "Multichromosomal median and halving problems under different genomic distances", *BMC bioinformatics*, v. 10, n. 1, pp. 120.

[54] Yancopoulos, S., Attie, O., Friedberg, R., 2005, "Efficient sorting of genomic permutations by translocation, inversion and block interchange", *Bioinformatics*, v. 21, n. 16, pp. 3340–3346.

Apêndice A

Anexo: Manuscrito “A faster 1.375-approximation algorithm for sorting by transpositions”

JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 22, Number 11, 2015

Pp. 1044–1056
DOI: 10.1089/cmb.2014.0298

A Faster 1.375-Approximation Algorithm for Sorting by Transpositions*

LUÍS FELIPE I. CUNHA,[1] LUIS ANTONIO B. KOWADA,[2]
RODRIGO DE A. HAUSEN,[3] CELINA M.H. DE FIGUEIREDO[1]

ABSTRACT

Sorting by Transpositions is an NP-hard problem for which several polynomial-time approximation algorithms have been developed. Hartman and Shamir (2006) developed a 1.5-approximation $O(n^{\frac{3}{2}}\sqrt{\log n})$ algorithm, whose running time was improved to $O(n\log n)$ by Feng and Zhu (2007) with a data structure they defined, the permutation tree. Elias and Hartman (2006) developed a 1.375-approximation $O(n^2)$ algorithm, and Firoz et al. (2011) claimed an improvement to the running time, from $O(n^2)$ to $O(n\log n)$, by using the permutation tree. We provide counter-examples to the correctness of Firoz et al.'s strategy, showing that it is not possible to reach a component by sufficient extensions using the method proposed by them. In addition, we propose a 1.375-approximation algorithm, modifying Elias and Hartman's approach with the use of permutation trees and achieving $O(n\log n)$ time.

Key words: approximation algorithms, genome rearrangement, sorting by transpositions.

1 INTRODUCTION

In the study of genome rearrangements, chromosomes are commonly modeled by permutations (Fertin et al., 2009). In the Sorting by Transpositions (SBT) problem, the aim is to find the minimum number of contiguous block interchanges that transforms a given permutation of n elements into the identity permutation; this minimum is called the *transposition distance* (Bafna and Pevzner, 1998). SBT is an NP-hard problem (Bulteau et al., 2012), and tight bounds on the transposition distance are known (Bafna and Pevzner, 1998; Labarre, 2006), but exact values for the transposition distance are known only for a few classes of permutations (Cunha et al., 2013a; Labarre, 2006). Several approaches to handling the SBT problem have been considered. Our focus is to explore approximation algorithms for estimating the transposition distance between permutations, providing better practical results or lowering time complexities.

Bafna and Pevzner (1998) designed a 1.5-approximation $O(n^2)$ algorithm, where n is the length of the input permutation, based on the cycle structure of the *breakpoint graph*. Hartman and Shamir (2006) later

Extended abstracts appeared in Cunha et al. (2013b, 2014b).

[1]COPPE – Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil.
[2]Instituto de Computação, Universidade Federal Fluminense, Rio de Janeiro, Brazil.
[3]Centro de Matemática, Computação e Cognição, Universidade Federal do ABC, São Paulo, Brazil.

proposed an easier 1.5-approximation algorithm that exploits a balanced tree data structure to decrease the running time to $O(n^{\frac{3}{2}}\sqrt{\log n})$. Feng and Zhu (2007) developed another balanced tree data structure—the *permutation tree*—and further decreased the complexity of Hartman and Shamir's 1.5-approximation algorithm to $O(n\log n)$. Elias and Hartman (2006) obtained, by a thorough computational case analysis of cycles of the breakpoint graph, a 1.375-approximation algorithm that runs in $O(n^2)$ time. Firoz et al. (2011) claimed that this 1.375-approximation algorithm could be easily adapted to run in $O(n\log n)$ time if a permutation tree was used.

In the present article, we show that Firoz et al.'s usage of the so-called ''Query'' procedure to extend a full configuration into a component fails in some situations. We provide an infinite family of permutations for which Firoz et al.'s approach does not find an 11/8-sequence, proving that the immediate use of a permutation tree is not enough to lower the running time of the 1.375-approximation algorithm to $O(n\log n)$. We rectify the use of the permutation tree, proposing a new algorithm that generalizes the bad small component strategy of Elias and Hartman toward *bad small full configurations*, achieving both the 1.375 approximation ratio and the $O(n\log n)$ time complexity. We thus achieve the best approximation ratio and time complexity for the SBT problem, known so far. The correctness of our algorithm is asserted via a branch-and-bound analysis that finds an 11/8-sequence for every combination of bad small full configurations.

The present article is organized as follows: section 2 contains basic definitions, some background on Elias and Hartman's algorithm and on the permutation tree data structure; section 3 discusses Firoz et al.'s approach on the use of a permutation tree to speed up the 1.375-approximation algorithm and provides counterexamples that establish the incorrectness of their approach; section 4 presents a strategy to determine the existence and to find a sequence of two transpositions, in which both are 2-moves, in linear time; section 5 describes our proposed 1.375-approximation $O(n\log n)$ algorithm for SBT, proving its correctness and its worst-case running time; and section 6 contains our final remarks.

2 BACKGROUND

For our purposes, a gene is represented by a unique integer and a chromosome with n genes is a *permutation* $\pi = [\boldsymbol{\pi_0}\pi_1\pi_2 \ldots \pi_n\boldsymbol{\pi}_{n+1}]$, where $\boldsymbol{\pi_0} = 0$, $\boldsymbol{\pi}_{n+1} = n+1$ and each π_i, where $1 \le i \le n$, is a unique integer in the range $1, \ldots, n$. The *transposition* $t(i, j, k)$ applied to π, where $1 \le i < j < k \le n+1$, is the permutation $\pi \cdot t(i, j, k)$ in which the two contiguous blocks $\pi_i\, \pi_{i+1} \ldots \pi_{j-1}$ and $\pi_j\, \pi_{j+1} \ldots \pi_{k-1}$ are interchanged. A sequence of q transpositions $t_1, t_2, \ldots, t_q$ *sorts* a permutation π if $\pi \cdot t_1 \cdot t_2 \ldots\ldots t_q = \iota$, where ι is the identity permutation $[\mathbf{0}\ 1\ 2 \ldots n\ \boldsymbol{n+1}]$. The *transposition distance* of π, denoted $d(\pi)$, is the length of a minimum sequence of transpositions that sorts π.

The breakpoint graph Nontrivial bounds on the transposition distance were obtained by using the breakpoint graph (Bafna and Pevzner, 1998). Given a permutation π, the *breakpoint graph* of π is $G(\pi) = (V, R \cup D)$. The set of vertices is $V = \{0, -1, +1, -2, +2, \ldots, -n, +n, -(n+1)\}$, and the set of edges is partitioned into two subsets, the directed *reality edges* $R = \{\overrightarrow{i} = (+\pi_i, -\pi_{i+1}) | i = 0, \ldots, n\}$ and the undirected *desire edges* $D = \{(+i, -(i+1)) \mid i = 0, \ldots, n\}$). Figure 1 shows $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$, where the arrows represent the directed edges in R and the arcs represent the undirected edges in D.

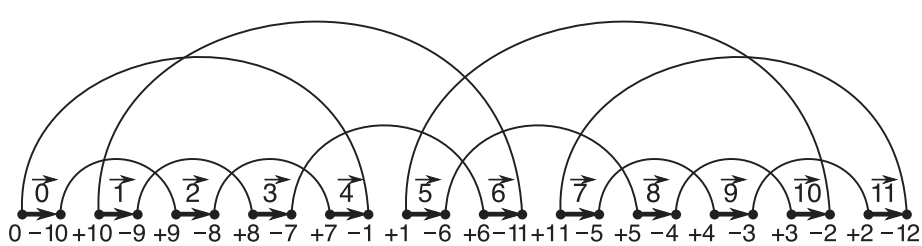

FIG. 1. $G([\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}])$. The set of cycles is $\{C_1 = \langle 024\rangle, C_2 = \langle 136\rangle, C_3 = \langle 5810\rangle, C_4 = \langle 7911\rangle\}$. The cycles C_2 and C_3 intersect, but are not interleaving; the cycles C_1 and C_2 are interleaving, and so are C_3 and C_4. The cycle C_1 is the leftmost cycle.

Since every vertex in $G(\pi)$ has degree 2, the graph can be partitioned into disjoint cycles. We shall use the terms *a cycle in* π and *a cycle in* $G(\pi)$ interchangeably to denote the latter. A cycle in π has length ℓ (or it is an ℓ*-cycle*), if it has exactly ℓ reality edges. A permutation π is a *simple permutation* if every cycle in π has length at most 3, as the example in Figure 1.

After applying a transposition t, the number of cycles of odd length in $G(\pi)$, denoted $c_{odd}(\pi)$, changes in such a way that $c_{odd}(\pi \cdot t) = c_{odd}(\pi) + x$, where $x \in \{-2, 0, 2\}$; the transposition t is thus classified as an *x-move for* π. Since $c_{odd}(\iota) = n+1$, we have the lower bound $d(\pi) \geq \left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil$, where the equality holds if, and only if, π can be sorted solely with 2-moves.

Hannenhalli and Pevzner (1999) proved that every permutation π can be transformed, in $O(n)$ time, into a simple one $\hat{\pi}$, by inserting new elements in appropriate positions of π, preserving the lower bound for the distance, $\left\lceil \frac{(n+1)-c_{odd}(\pi)}{2} \right\rceil = \left\lceil \frac{(m+1)-c_{odd}(\hat{\pi})}{2} \right\rceil$, where m is such that $\hat{\pi} = [\mathbf{0}\hat{\pi}_1 \ldots \hat{\pi}_m \mathbf{m+1}]$. Additionally, in a sequence that sorts $\hat{\pi}$, every transposition can be transformed, in $O(\log n)$ time (Feng and Zhu, 2007), into a sequence with the same number of transpositions that sorts π, which implies that $d(\pi) \leq d(\hat{\pi})$. The approach of finding a sorting sequence for π via a simple permutation $\hat{\pi}$ is commonly used in approximation algorithms for SBT (Elias and Hartman, 2006; Hartman and Shamir, 2006).

A transposition $t\ (i, j, k)$ *affects* a cycle C if it contains one of the following reality edges: $\overrightarrow{i+1}$, or $\overrightarrow{j+1}$, or $\overrightarrow{k+1}$. A cycle is *oriented* if there is a 2-move that affects it, otherwise it is *unoriented* (these names come from the order of such triplet of reality edges in the breakpoint graph). If π contains an oriented cycle then π is *oriented*, otherwise π is *unoriented*.

A sequence of q transpositions of which exactly r transpositions are 2-moves is a (q,r)-sequence. A q/r-sequence is an (x,y)-sequence such that $x \leq q$ and $x/y \leq q/r$.

Interactions between cycles A cycle in π can be uniquely identified by its reality edges, in the order that they appear, starting from the leftmost edge. The notation $C = \langle x_1 x_2 \ldots x_\ell \rangle$, where $\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_\ell}$ are reality edges, and $x_1 = \min\{x_1, x_2, \ldots, x_\ell\}$, characterizes an ℓ-cycle. The *leftmost cycle* is the cycle that contains the edge $\overrightarrow{0}$.

Let $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$, where $x < y < z$, be reality edges in a cycle C, and $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$, where $a < b < c$ be reality edges in a different cycle C'. The pair of reality edges $\overrightarrow{x}, \overrightarrow{y}$, *intersects* the pair $\overrightarrow{a}, \overrightarrow{b}$, if these four edges occur in an alternating order in the breakpoint graph, that is, either $x < a < y < b$ or $a < x < b < y$, and we say C and C' *intersect*. Similarly, a triplet of reality edges $\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{z}$ *interleaves* a triplet $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$ if these six edges occur in an alternating order: $x < a < y < b < z < c$ or $a < x < b < y < c < z$. Two 3-cycles *interleave* if their respective triplets of reality edges interleave. Figure 1 also illustrates these concepts.

A *configuration* of π is a subset of the cycles in $G(\pi)$. A configuration $\mathscr{C}$ is *connected* if there is a sequence of $C_1, \ldots, C_k$ in $\mathscr{C}$ such that $C_1 = C$, $C_k = C'$ and for each $i \in \{1, 2, \ldots, k-1\}$, the cycles C_i and C_{i+1} are intersecting. If the configuration $\mathscr{C}$ is connected and maximal, then $\mathscr{C}$ is a *component*. Every permutation admits a unique decomposition into disjoint components. For instance, in Figure 1, the configuration $\{C_1, C_2, C_3, C_4\}$ is a component, but the configuration $\{C_1, C_2, C_3\}$ is connected but not a component.

Let C be a 3-cycle in configuration $\mathscr{C}$. An *open gate* is a pair of reality edges in C that does not intersect any other pair of reality edges in $\mathscr{C}$. If a configuration $\mathscr{C}$ has only 3-cycles and no open gates, then $\mathscr{C}$ is a *full configuration*. Some full configurations do not correspond to the breakpoint graph of any permutation. A full configuration corresponds to a permutation if, and only if, the *complement configuration* is Hamiltonian (Elias and Hartman, 2006), as illustrated in Figure 2b. Figure 2a shows the full configuration $F = \{\langle 0\,7\,9 \rangle, \langle 1\,3\,6 \rangle, \langle 2\,4\,11 \rangle, \langle 5\,8\,11 \rangle\}$, which does not correspond to any permutation but is important in the analysis of our algorithm and will be studied in detail in section 5.

A configuration $\mathscr{C}$ that has k edges is in the *cromulent form*[4] if all edges $\overrightarrow{0}, \overrightarrow{1}, \ldots, \overrightarrow{k-1}$ are in $\mathscr{C}$. Given a configuration $\mathscr{C}$ having k edges, a *cromulent relabeling* of $\mathscr{C}$ is a configuration $\mathscr{C}$ such that $\mathscr{C}'$ is in the cromulent form and there is a function σ satisfying that, for every pair of edges $\overrightarrow{i}, \overrightarrow{j}$, in $\mathscr{C}$ such that $i < j$,

[4]*Cromulent* is neologism coined by David X. Cohen, meaning "normal" or "acceptable."

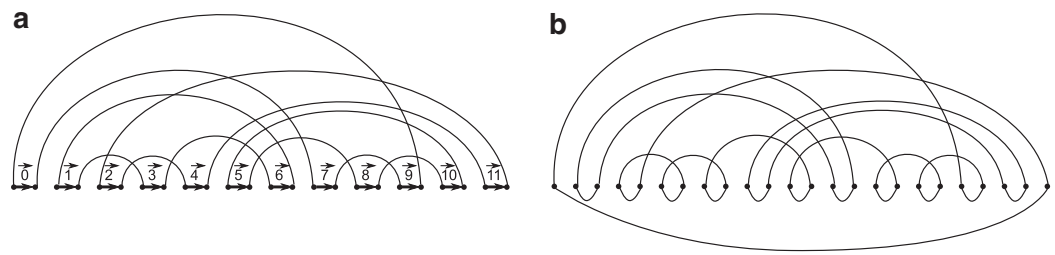

FIG. 2. **(a)** Full configuration $F = \{\langle 079\rangle, \langle 136\rangle, \langle 2411\rangle, \langle 5810\rangle\}$, which does not correspond to a permutation. **(b)** Complement of F, obtained by replacing the reality edges with the edges $\text{-}\pi_i\pi_i$ and $0\text{-}\pi_{n+1}$.

we have that $\overrightarrow{\sigma(i)}, \overrightarrow{\sigma(j)}$, are in $\mathscr{C}'$ and $\sigma(i) < \sigma(j)$. For instance, in Figure 2a the cromulent relabeling of the configuration $\{C_1 = \langle 0, 7, 9\rangle, C_2 = \langle 1, 3, 6\rangle\}$ is $\{C_1 = \langle 0, 4, 5\rangle, C_2 = \langle 1, 2, 3\rangle\}$.

Given an integer x, a *circular shift* of a configuration $\mathscr{C}$, which is in the cromulent form and has k edges, is a configuration denoted $\mathscr{C} + x$ such that every edge $\overrightarrow{i}$ in $\mathscr{C}$ corresponds to $\overrightarrow{i + x (\text{mod } k)}$ in $\mathscr{C} + x$. Two configurations $\mathscr{C}$ and $\mathscr{K}$ are *equivalent* if there is an integer x such that $\mathscr{C}' + x = \mathscr{K}'$, where $\mathscr{C}'$ and $\mathscr{K}'$ are their respective cromulent relabelings.

Elias and Hartman's algorithm Elias and Hartman (2006) performed a systematic enumeration of all components with nine cycles or less, in which all cycles have length 3. Starting from single 3-cycles, components were obtained by applying a series of *sufficient extensions*, as described next. An *extension* of a configuration $\mathscr{C}$ is a connected configuration $\mathscr{C} \cup \{C\}$, where $C \notin \mathscr{C}$. A *sufficient extension* is an extension that either: 1) closes an open gate; or 2) extends a full configuration such that the extension has at most one open gate. A configuration obtained by a series of sufficient extensions is a *sufficient configuration* if an *(x,y)*-, or an *x/y*-sequence can be applied to its cycles.

Lemma 1 *(Elias and Hartman, 2006) Every unoriented sufficient configuration of nine cycles has an* 11/8-*sequence.*

Components with less than nine cycles are called *small components*. Elias and Hartman have shown that, of all small components, only five types of them do not have an 11/8-sequence; these components are called *bad small components*. Small components that have an 11/8-sequence are called *good small components*.

Lemma 2 *(Elias and Hartman, 2006) The bad small components are:*

- $A = \{\langle 0\,2\,4\rangle, \langle 1\,3\,5\rangle\}$;
- $B = \{\langle 0\,2\,10\rangle, \langle 1\,3\,5\rangle, \langle 4\,6\,8\rangle, \langle 7\,9\,11\rangle\}$;
- $C = \{\langle 0\,5\,7\rangle, \langle 1\,9\,11\rangle, \langle 2\,4\,6\rangle, \langle 3\,8\,10\rangle\}$;
- $D = \{\langle 0\,2\,4\rangle, \langle 1\,12\,14\rangle, \langle 3\,5\,7\rangle, \langle 6\,8\,10\rangle, \langle 9\,11\,13\rangle\}$; *and*
- $E = \{\langle 0\,2\,16\rangle, \langle 1\,3\,5\rangle, \langle 4\,6\,8\rangle, \langle 7\,9\,11\rangle, \langle 10\,12\,14\rangle, \langle 13\,15\,17\rangle\}$.

If a permutation has bad small components, it is still possible to find an (11,8)-sequence, as Lemma 3 states.

Lemma 3 *(Elias and Hartman, 2006) Let π be a permutation with at least eight cycles and containing only bad small components. Then π has an* (11,8)-*sequence.*

Corollary 1 *(Elias and Hartman, 2006) If every cycle in $G(\pi)$ is a* 3-*cycle, and there are at least eight cycles, then π has an* 11/8-*sequence.*

Lemmas 1 and 3 and Corollary 1 form the theoretical basis for Elias and Hartman's $11/8 = 1.375$-approximation algorithm for SBT, Algorithm 1. The main procedure of Algorithm 1 is: obtain extensions, if a configuration with nine cycles or if a small good component is reached, then an 11/8-sequence is applied (Lemma 1); if a bad small component is reached then no sequence is applied. After all configurations with nine cycles or small good components have been sorted, the permutation just contains small bad components; Lemma 3 states the existence of an (11,8)-sequence.

Algorithm 1: Elias and Hartman's Sort(π)

```
Transform permutation π into a simple permutation π̂.
Check if there is a (2,2)-sequence. If so, apply it.
While G(π̂) contains a 2-cycle, apply a 2-move.
π̂ consists of 3-cycles. Mark all 3-cycles in G(π̂).
while G(π̂) contains a marked 3-cycle C do
    if C is oriented then
        Apply a 2-move.
    else
        Try to sufficiently extend C eight times (to obtain a configuration with at most nine cycles).
        if sufficient configuration with nine cycles has been achieved then
            Apply an 11/8-sequence.
        else It is a small component
            if it is a good component then
                Apply an 11/8-sequence.
            else
                Unmark all cycles of the component.
(Now G(π̂) has only bad small components.)
while G(π̂) contains at least eight cycles do
    Apply an (11,8)-sequence
While G(π̂) contains a 3-cycle, apply a (3,2)-sequence.
Mimic the sorting of π using the sorting of π̂.
```

Feng and Zhu's permutation tree Feng and Zhu (2007) introduced the *permutation tree*, a binary balanced tree that represents a permutation. In logarithmic time the operation of applying a transposition could be done, and also the *Query* procedure of finding a pair of reality edges that intersects another given pair of reality edges, as well. The *Query* procedure is the method used in Hartman and Shamir's (2006) 1.5-approximation algorithm to find a (3,2)-sequence that affects a pair of intersecting or interleaving cycles. Besides that, Firoz et al. (2011) claimed the *Query* procedure could be used to sufficiently extend a configuration in Algorithm 1.

Let $\pi = [\boldsymbol{\pi_0}\pi_1\pi_2 \ldots \pi_n\boldsymbol{\pi_{n+1}}]$ be a permutation. The corresponding permutation tree has n leaves, labeled $\pi_1, \pi_2, \ldots, \pi_n$; every internal node represents an interval of consecutive elements $\pi_i, \pi_{i+1}, \ldots, \pi_{k-1}$, with $i<k$, and is labeled by the maximum number in the interval. Therefore, the root of the tree is labeled with n. Furthermore, the left child of a node represents the interval $\pi_i, \ldots, \pi_{j-1}$, and the right child represents $\pi_j, \ldots, \pi_k$, with $i<j<k$. Feng and Zhu provided algorithms: to *build* a permutation tree in $O(n)$ time; to *join* two permutation trees into one in $O(h)$ time, where h is the height difference between the trees; and to *split* a permutation tree into two in $O(\log n)$ time.

The operations *split* and *join* allow us to apply a transposition to a permutation π, updating the tree, in time $O(\log n)$. Based on Lemma 4, the *Query* procedure (Algorithm 2) solves the problem of finding a pair of reality edges intersecting another given pair of reality edges.

Lemma 4 *(Bafna and Pevzner, 1998) Let $\overrightarrow{i}$ and $\overrightarrow{j}$ be two reality edges in an unoriented cycle C, $i<j$. Let $\pi_k = max_{i<m\leq j}\pi_m$, $\pi_\ell = \pi_k + 1$. Then, the reality edges $\overrightarrow{k}$ and $\overrightarrow{\ell-1}$ belong to the same cycle, and the pair $\overrightarrow{k}, \overrightarrow{\ell-1}$ intersects pair $\overrightarrow{i}, \overrightarrow{j}$.*

Algorithm 2: *Query(π,i,j)*

```
input: permutation π, integers i and j
Let T be the permutation tree of π
Split T, into three permutation trees, T1, T2 and T3, corresponding to [π0, π1, ..., πi], [πi+1, ..., πj],
    and [πj+1, ..., πn, πn+1], respectively.
Let πk = root (T2). (the largest element in the interval πi+1, ..., πJ)
Let πℓ = πk + 1
Return the pair k, ℓ − 1 (by Lemma 4, k→, l−1→ intersects, i→, J→)
```

Firoz et al. (2011) suggested the use of the permutation tree data structure to reduce the running time of Algorithm 1 to $O(n\log n)$, but in section 3 we show that the strategy, in the manner proposed by Firoz et al., fails to extend some 3-cycles into a full configuration with nine cycles.

3. THE USE OF THE PERMUTATION TREE BY FIROZ ET AL.

Firoz et al. (2011) stated that Step 9 in Algorithm 1 could be done in $O(\log n)$ time. To do so, they categorized the sufficient extensions of a configuration A obtained by a *Query* call into *type 1 extensions*—those that add a cycle that closes an open gate—and *type 2 extensions*—those that extend a full configuration by adding a cycle C such that $A \cup \{C\}$ has at most one open gate.

A type 1 extension can be performed in logarithmic time with *Query*(π, i, j), where $\overrightarrow{i}, \overrightarrow{j}$ form an open gate. For a type 2 extension, since there are no open gates, Firoz et al. claimed that it would be sufficient to perform queries with every pair of reality edges that belonged to the same cycle in the configuration that is being extended. Example 1 shows that this strategy is flawed.

Example 1 *Consider the permutation* $\pi = [\mathbf{0}\ 10\ 9\ 8\ 7\ 1\ 6\ 11\ 5\ 4\ 3\ 2\ \mathbf{12}]$, *whose breakpoint graph is depicted in Figure 1. It is a simple permutation having only unoriented 3-cycles. Mark all the cycles* $C_1 = \langle 0,2,4\rangle$, $C_2 = \langle 1, 3, 6\rangle$, $C_3 = \langle 5,8,10\rangle$, *and* $C_4 = \langle 7,9,11\rangle$. *Let* $A = \{C_1\}$ *be the configuration to be sufficiently extended (step* 9 *in Algorithm 1). Using the method proposed by Firoz et al., we have that:*

1. *Configuration A has three open gates* $\overrightarrow{0}, \overrightarrow{2}; \overrightarrow{2}, \overrightarrow{4}; \overrightarrow{4}, \overrightarrow{0}$. *Execute Query*$(\pi, 0, 2)$, *which returns the pair* $\overrightarrow{1}, \overrightarrow{6}$, *in cycle* $C_2 = \langle 1, 3, 6\rangle$ *(or, alternatively Query*$(\pi, 2, 4)$, *which yields the same result). Therefore,* C_2 *is added to the configuration A, which becomes* $A = \{C_1, C_2\}$.
2. *Configuration A has no more open gates. We must execute Query*(π,i,j) *for every pair of elements* $\overrightarrow{i}, \overrightarrow{j}$ *in the same cycle of the configuration such that* $i<j$; *it is easy to observe that each execution returns a pair that is already in A. So far, Firoz et al.'s method has failed to extend A.*
3. *Since A is not a component, unmark all the cycles in A.*
4. *The marked cycles are now* C_3 *and* C_4. *Considering either* $A = \{C_3\}$ *or* $A = \{C_4\}$, *Firoz et al.'s method only extends A as far as* $\{C_3, C_4\}$. *Again, A is not a component.*

Therefore, Firoz et al.'s method fails to find the component $\{C_1, C_2, C_3, C_4\}$.

Although the permutation in Example 1 has only one small component, it is a counterexample to the correctness of Firoz et al.'s strategy for dealing with type 2 extensions. The same problem happens for sufficient configurations with more than nine cycles, such as:

$$\sigma = [\mathbf{0}\ 25\ 24\ 23\ 22\ 1\ 21\ 26\ 20\ 19\ 18\ 2\ 17\ 27\ 16\ 15\ 14\ 3\ 13\ 28\ 12\ 11\ 10\ 4\ 9\ 29\ 8\ 7\ 6\ 5\ \mathbf{30}].$$

By Lemma 1, every configuration of nine cycles has an 11/8-sequence. Figure 3 shows an example of a breakpoint graph of σ with 10 cycles, for which any configuration with 9 cycles has an 11/8-sequence. However, Firoz et al.'s approach fails to find such a sequence, for it performs the following sequence of operations: i) starting from any configuration having a unique cycle, the first call to *Query* correctly finds another intersecting cycle, which is added to the configuration (Fig. 4); ii) with this configuration having two interleaving cycles, every possible invocation of *Query* returns one of the cycles already in the configuration, which means that their strategy cannot further extend the configuration; iii) the resulting configuration, with only two cycles, is a bad small component, so the cycles are unmarked; iv) if an unmarked cycle still remains, it is selected to start a configuration, and we return to the first step in this sequence of operations. The procedure finishes after unmarking all 10 cycles, incorrectly presuming that the breakpoint graph only has bad small components with two cycles.

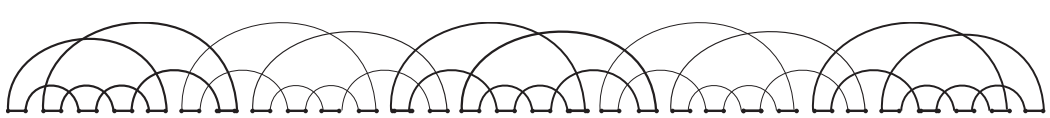

FIG. 3. Breakpoint graph of a permutation σ for which Firoz et al.'s method fails. Note that σ has 10 cycles, and that σ is obtained by setting $k = 5$ in Equation (2).

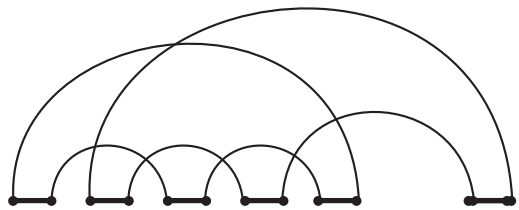

FIG. 4. A configuration that is not maximal returned by a *Query* call on σ.

Notice that, according to Step 18 in Algorithm 1, if a permutation contains only bad small components, then an 11/8-sequence can be applied. The permutation γ (whose breakpoint graph is in Fig. 5) is one of those permutations:

$$\gamma = [\mathbf{0}\ 5\ 4\ 3\ 2\ 1\ 6\ 11\ 10\ 9\ 8\ 7\ 12\ 17\ 16\ 15\ 14\ 13\ 18\ 23\ 22\ 21\ 20\ 19\ 24\ 29\ 28\ 27\ 26\ 25\ \mathbf{30}],$$

Notice how the breakpoint graphs in Figures 4 and 5 differ, but according to Firoz et al.'s approach they would be indistinguishable. In Figure 5, the bad small components can be separately handled, which is not the case for configurations with two interleaving cycles in Figure 4, for these configurations intersect other cycles. Algorithm 1 does not have a rule to deal with this last case.

Note that the permutation of Example 1 and the permutation above σ just describe examples belonging to a family of permutations such that any type 2 extension fails. Actually, an infinite family can be constructed as follows: let k be any integer greater than or equal to 2, and let $f(i)$ be the sequence of six integers

$$i\ 5k-4i\ 5k+i\ 5k-4i-1\ 5k-4i-2\ 5k-4i-3. \tag{1}$$

Consider σ^k a permutation of $6k-1$ elements defined using Equation (1) as:

$$[\mathbf{0}\ 5k\ 5k-1\ 5k-2\ 5k-3\ f(1)\ f(2)\ldots\ f(k-1)\ k\ \mathbf{6k}], \tag{2}$$

whose breakpoint graph has a similar structure to those in Figure 1 (where in Equation (2) we set $k=2$) and 3. If we start from a configuration having any cycle, it is impossible to extend it past a configuration of more than two cycles using Firoz et al.'s approach.

Some other configurations cannot be extended using only the *Query* procedure either, such as the full configuration illustrated in Figure 2. This is a bad small configuration (Elias and Hartman, 2006) that does not correspond to the breakpoint graph of any permutation, but this configuration may appear during the sorting of a larger permutation.

4. FINDING AND APPLYING A (2,2)-SEQUENCE IN LINEAR TIME

In order to implement Step 2 of Algorithm 1, Elias and Hartman (2006) proved that, given a simple permutation, a (2,2)-sequence can be found in $O(n^2)$ time.

Firoz et al. (2011) described a strategy for finding and applying a (2,2)-sequence in $O(n\log n)$ time using permutation trees. But, according to their strategy, for each one of the $O(n)$ oriented 3-cycles, we apply a 2-move and check in $O(n)$ time for the existence of an oriented cycle in the resulting graph, which implies that Firoz et al.'s strategy may run in $\Omega(n^2)$ time in the worst case. One such case is illustrated in Figure 6, where the cycles drawn in solid lines—one oriented, the other unoriented—are interleaving, so by case 3 of Lemma 5 there is a (2,2)-sequence that affects them. However, if the first 2-move is applied to any of the

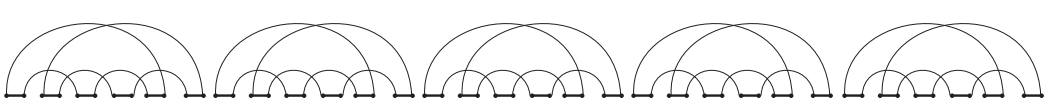

FIG. 5. The breakpoint graph of permutation γ.

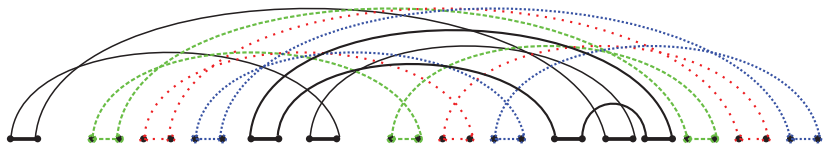

FIG. 6. A breakpoint graph for which Firoz et al.'s strategy takes $\Omega(n^2)$ time in the worst case to find a (2,2)-sequence.

dashed cycles, the resulting breakpoint graph has no oriented cycle, hence the transposition is undone and another oriented cycle is selected; this procedure would continue until the solid oriented cycle is selected.

Algorithm 5 summarizes our proposed approach toward finding and applying a (2,2)-sequence in $O(n)$ time. It is a direct application of Algorithms 3 and 4 to the cases stated in Lemma 5.

Algorithm 3: Search (2,2)-sequence from K_1.

1 **for** $i =$ min $K_1 + 1, \ldots,$ mid $K_1 - 1$ **do**
2 **if** $\overrightarrow{i}$ *belongs to an oriented cycle* K_j **then**
3   **if** mid $K_j <$ mid K_1 **then**
4   **return** (2,2)-sequence that affects K_1 and K_j.
5   **else if** max $K_j <$ max K_1 **then**
6   **return** (2,2)-sequence that affects K_1 and K_j.
7   **else if** max $K_1 <$ mid K_j **then**
8   **return** (2,2)-sequence that affects K_1 and K_j.
9 **if** $\overrightarrow{i}$ *belongs to an unoriented cycle* L_j **then**
10   **if** mid $K_1 <$ mid $L_j <$ max $K_1 <$ max L_j **then**
11   **return** (2,2)-sequence that affects K_1 and L_j
12   **else if** min $L_j <$ min $K_1 <$ mid $L_j <$ mid $K_1 <$ max $L_j <$ max K_1 **then**
13   **return** (2,2)-sequence that affects K_1 and L_j.
14 **for** $i =$ mid $K_1 + 1, \ldots,$ max $K_1 - 1$ **do**
15 **if** $\overrightarrow{i}$ *belongs to an oriented cycle* K_j **then**
16   **if** mid $K_1 <$ min K_j **then**
17   **return** (2,2)-sequence that affects K_1 and K_j.
18 **for** $i =$ max $K_1 + 1, \ldots, n - 1$ **do**
19 **if** $\overrightarrow{i}$ *belongs to an oriented cycle* K_j **then**
20   **if** max $K_1 <$ min K_j **then**
21   **return** (2,2)-sequence affecting K_1 and K_j.

Lemma 5 *(Bafna and Pevzner, 1998; Christie, 1999; Elias and Hartman, 2006) Given a breakpoint graph of a simple permutation, there exists a* (2,2)*-sequence if any of the following conditions is met:*

1. *There are either four* 2*-cycles, or two intersecting* 2*-cycles, or two nonintersecting* 2*-cycles, and the resulting graph contains an oriented cycle after the first transposition is applied;*
2. *There are two noninterleaving oriented* 3*-cycles;*
3. *There is an oriented cycle interleaving an unoriented cycle.*

Our strategy to find a (2,2)-sequence in linear time starts by checking whether the breakpoint graph satisfies first case of Lemma 5, as described in detail between lines 1 and 4 in Algorithm 5. In our approach, it is unnecessary to try all pairs of cycles to verify that conditions 2 and 3 in Lemma 5 are satisfied. It differs from previous methods (Elias and Hartman, 2006; Firoz et al., 2011) in that the leftmost oriented cycle of the breakpoint graph, named K_1, is fixed when verifying for conditions 2 and 3.

Given a simple permutation π, it is immediate to enumerate all of its cycles in linear time. The size of each cycle, and whether it is oriented, are both determined in constant time.

Christie (1999) proved that every permutation has an even number (possibly zero) of even cycles; he also showed that, given a simple permutation, when the number of even cycles is not zero, there exists a (2,2)-

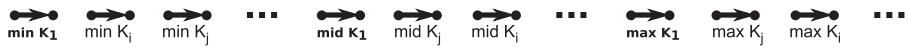

FIG. 7. Oriented cycles represented by their reality edges. All oriented cycles interleave K_1, but there are i and j such that K_i and K_j are noninterleaving.

sequence that affects those cycles if, and only if, there are either four 2-cycles, or there are two intersecting even cycles. Therefore, in these cases, a (2,2)-sequence can be applied in $O(\log n)$ using permutation trees. If there is only a pair of nonintersecting 2-cycles, it remains to check if there is a 3-cycle intersecting both even cycles: i) if the 3-cycle is oriented, then first we apply the 2-move applied to the 3-cycle, and the second 2-move is applied to 2-cycles; ii) if the 3-cycle is unoriented, then first we apply the 2-move applied to the 2-cycles, and the second 2-move is applied to the 3-cycle, which turns oriented after the first transposition. There is also a (2,2)-sequence if there is an oriented cycle intersecting at most one even cycle.

However, if there are no even cycles in the permutation, but there is an oriented cycle, the 3-cycles must be scanned for the existence of a (2,2)-sequence, as conditions 2 and 3 require in Lemma 5.

Algorithm 4: Finding intersecting oriented cycles interleaving K_1.

```
s1 = sequence of edges belonging to oriented cycles from left to right between min K1 and mid K1.
s2 = sequence of edges belonging to oriented cycles from left to right between mid K1 and max K1.
if the sequences of cycles corresponding to s1 and s2 are different then
    There is a pair of intersecting oriented cycles, exists a (2,2)-sequence.
else
    All oriented cycles are mutually interleaving.
```

To check, in linear time, for the existence of a pair of cycles satisfying either condition 2 or 3 in Lemma 5, consider the oriented cycles of the breakpoint graph, in the order $K_1 = \langle a_1\ b_1\ c_1\rangle$, $K_2 = \langle a_2\ b_2\ c_2\rangle$, $K_3 = \langle a_3\ b_3\ c_3\rangle, \ldots$ such that $a_1 < a_2 < a_3 < \ldots$, and the unoriented cycles in the order $L_1 = \langle x_1\ y_1\ z_1\rangle$, $L_2 = \langle x_2\ y_2\ z_2\rangle$, $L_3 = \langle x_3\ y_3\ z_3\rangle, \ldots$ such that $x_1 < x_2 < x_3 < \ldots$. Given any 3-cycle $C = \langle abc\rangle$, let min $C = a$, mid $C = \min\{b,c\}$, and max $C = \max\{b,c\}$, that is, if C is unoriented, then min $C = a$, mid $C = b$, max $C = c$, whereas if C is oriented, then min $C = a$, mid $C = c$, max $C = b$. The main idea is:

1. Check for the existence of an oriented cycle K_j noninterleaving K_1 or an unoriented cycle L_j interleaving K_1. Algorithm 3 searches for an oriented cycle K_i noninterleaving K_1 or an unoriented cycle L_i interleaving K_1. The search is done between $minK_1$ and mid K_1, between mid K_1 and max K_1, and to the right of max K_1.
2. If Algorithm 3 does not return any oriented cycle noninterleaving K_1, then every oriented cycle interleaves K_1 but no unoriented cycle interleaves K_1. Hence, we must check for the existence of two oriented cycles K_i, K_j that are intersecting but not interleaving. Note that if K_i, K_j were nonintersecting oriented cycles, Algorithm 3 would have this case already covered (see Fig. 7), since K_i or K_j would not interleave K_1. Algorithm 4 describes how to verify the existence of two intersecting oriented cycles that are also interleaving with K_1.

Algorithm 5: Find and Apply (2,2)-sequence

```
if there are four 2-cycles then
    Apply (2,2)-sequence.
else if there is a pair of intersecting 2-cycles then
    Apply (2,2)-sequence.
else if there is a 3-cycle intersecting a pair of 2-cycles then
    Apply (2,2)-sequence.
else if there is a pair of 2-cycles and an oriented 3-cycle intersecting at most one of them then
    Apply (2,2)-sequence.
else if Search (2,2)-sequence from K1 returns a sequence then
    Apply (2,2)-sequence.
else if Finding intersecting oriented cycles interleaving K1 then
    Apply (2,2)-sequence.
else
    There are no (2,2)-sequences to apply.
```

5. SUFFICIENT EXTENSIONS USING THE *QUERY* PROCEDURE

In section 3, we discussed Firoz et al.'s use of the permutation tree and proved that their strategy does not account for every configuration with less than nine cycles that is not a component, since successive invocations of *Query* may result in a full configuration with less than nine cycles that is not a small component. Our proposed strategy generalizes the definitions regarding small components to *small configurations*—configurations with less than nine cycles.

A small configuration is *full* if it has no open gates. Small configurations are also classified as *good* if they have an 11/8-sequence, or as *bad* otherwise.

Algorithm 1 applies an 11/8-sequence to every sufficient unoriented configuration of nine cycles, and also to every good small component. After that, the permutation contains just bad small components, and Lemma 3 states that there exists an (11,8)-sequence for every combination of bad small components with at least eight cycles.

Our approach can handle bad small full configurations, which may or may not be bad small components, during the course of an extension via successive invocations of *Query*. The possible bad small full configurations are the bad small components A, B, C, D, and E, from Lemma 2, and the full configuration $F = \{\langle 079\rangle, \langle 136\rangle, \langle 2411\rangle, \langle 5810\rangle\}$, the only bad small full configuration that is not a component (Elias and Hartman, 2006).

Our strategy (Algorithm 6) is similar to Elias and Hartman's (Algorithm 1): apply an 11/8-sequence to every sufficient unoriented configuration of nine cycles and also to every good small full configuration; the main difference is that, whenever a combination of bad small full configurations is found, a decision to apply an 11/8-sequence is made according to Lemmas 6 and 7.

We developed a tool (Cunha et al., 2014a) that finds 11/8-sequences for a given configuration using branch-and-bound, where a *branch* is obtained by either applying a 2-move or a 0-move, and the moves are *bounded* by the ratio between the number of total moves and the number of 2-moves, which cannot be greater than 1.375. The algorithm either returns an 11/8-sequence, whenever it exists, or fails after trying all possible sequences.

Lemma 6 *Every combination of F with one or more copies of either B, C, D, or E has an* 11/8*-sequence.*

Proof. Consider all breakpoint graphs of F and its circular shifts combined with B, C, D, E, and their circular shifts. A combination of a pair of small full configurations is obtained by starting from one small full configuration and inserting a new one in different positions in the breakpoint graph. Altogether, there are 324 such graphs. A computerized case analysis (Cunha et al., 2014a) enumerates all possible breakpoint graphs and provides an 11/8-sequence for each of them. ■

Notice that Lemma 6 considers neither combinations of F with F, nor combinations of F with A. We have found that almost every combination of F with F has an 11/8-sequence, as Lemma 7 states. Let F_iF^j be the configuration obtained by inserting the circular shift $F+j$ between the edges $\overrightarrow{i}$ and $\overrightarrow{i+1}$ of F.

Lemma 7 *There exists an* 11/8*-sequence for* F_iF^j*, if:*

- $i \in \{0,4\}$ *and* $j \in \{0,1,2,3,4,5\}$*;*
- $i \in \{1,2,3\}$ *and* $j \in \{1,2,3,4,5\}$*; or*
- $i = 5$ *and* $j \in \{1,5\}$.

Proof. The 11/8-sequences for the cases enumerated above were also found through a computerized case analysis (Cunha et al., 2014a). Note that F_iF^j is equivalent to $F_{i+6}F^j$ for $i = \{0, 1, \ldots, 5\}$, which simplifies our analysis. ■

Only seven combinations of F with F have no 11/8-sequence: F_1F^0, F_2F^0, F_3F^0, F_5F^0, F_5F^2, F_5F^3, and F_5F^4. We will return to them shortly.

All combinations of one copy of F and one copy of A have less than eight cycles. It only remains to analyze the combinations of F and two copies of A, denoted F–A–A. The *good F–A–A combinations* are the

F–A–A combinations for which an 11/8-sequence exists. Out of 57 combinations of *F–A–A*, only 31 are good. The complete list of combinations is in Cunha et al. (2014a).

Combinations of F with A, B, C, D, E, and F that have 11/8-sequences are called *well-behaved combinations*—the ones in Lemmas 6, 7, and the good *F–A–A* combinations. The remaining combinations having F are called *naughty*: the seven combinations of $F - F$ that have no 11/8-sequence, and the 57 combinations of *F–A–A*.

For extensions that yield a bad small configuration, Algorithm 6 adds their cycles to a set $\mathscr{S}$ (line 32). Later, if a well-behaved combination is found among the cycles in $\mathscr{S}$, an 11/8-sequence is applied (line 37) and the set is emptied. If all combinations in $\mathscr{S}$ are naughty, another bad small configuration can be obtained and added to it in the next iteration (line 6).

We have shown (Cunha et al., 2014a) that every combination of three copies of F is well-behaved, even if each pair of $F - F$ is naughty; the same can be said of every combination of F and three copies of A, even if each triple *F–A–A* is naughty. Therefore, at most 12 cycles are in $\mathscr{S}$, since it may contain at most three copies of F, or one copy of F and three copies of A, in the worst case. For each of these cases, there exists an 11/8-sequence (Cunha et al., 2014a).

Proposed algorithm Algorithm 6 is a direct application of the results in the section. In a nutshell, it obtains configurations using the *Query* procedure and applies 11/8-sequences to configurations of size at most 9. Algorithm 6 differs from Algorithm 1 not only in the use of permutation trees, but also because the main loop handles bad small full configurations, instead of only dealing with them at the end.

Algorithm 6: Proposed algorithm based on Elias and Hartman's algorithm

```
1  Transform permutation π into a simple permutation π̂.
2  Find and apply (2,2)-sequence (Algorithm 5).
3  While G(π̂) contains a 2-cycle, apply a 2-move.
4  π̂ consists of 3-cycles. Mark all 3-cycles in G(π̂).
5  Let 𝒮 be an empty set.
6  while G(π̂) contains at least eight 3-cycles do
7      Start a configuration 𝒞 with a marked 3-cycle.
8      if the cycle in 𝒞 is oriented then
9          Apply a 2-move.
10     else
11         Try to sufficiently extend 𝒞 eight times using the Query procedure.
12         if 𝒞 is a sufficient configuration with nine cycles then
13             Apply an 11/8-sequence.
14         else
               𝒞 is a small full configuration
15             if 𝒞 is a good small configuration then
16                 Apply an 11/8-sequence.
17             else
                   𝒞 is a bad small configuration
18                 Add every cycle in 𝒞 to 𝒮.
19                 Unmark all cycles in 𝒞.
20                 if 𝒮 contains a well-behaved combination then
21                     Apply an 11/8-sequence.
22                     Mark the remaining 3-cycles in 𝒮.
23                     Remove all cycles from 𝒮.
24 While G(π̂) contains a 3-cycle, apply a 4/3-sequence or a 3/2-sequence.
25 Mimic the sorting of π using the sorting of π̂.
```

Theorem 1 *Algorithm 6 runs in O(n*log*n) time.*

Proof. Steps 1 through 5 can be implemented to run in linear time [Elias and Hartman (2006); Feng and Zhu (2007), and section 4]. Step 17 runs in $O(\log n)$ time using permutation trees. The comparisons in Steps 12, 15, and 20 are done in $O(1)$ time using lookup tables whose sizes are bounded by a constant. Updating the set $\mathscr{S}$ also requires constant time, since it has at most 12 cycles (case where $\mathscr{S}$ contains $F - F - F$). Every sequence of transpositions of size bounded by a constant can be applied in time O (log n) due to the use of permutation trees. The time complexity of the loop between Steps 6 to 23 is $O(n\log n)$, since the number of 3-cycles is linear in n, and the number of cycles decreases, in the worst case, every third iteration. In Step 24, the search for a 4/3– or a 3/2-sequence is done in constant time, since the number of cycles is bounded by a constant. Steps 24 and 25 also run in time $O(n\log n)$, according to Feng and Zhu (2007). ■

6. FINAL REMARKS

Although sorting permutations by transpositions is an NP-hard problem, some approximation strategies have been successful. This article describes a 1.375-approximation algorithm that rectifies a previous attempt (Firoz et al., 2011) of using the permutation tree data structure to achieve a running time of $O(n\log n)$. We have managed to achieve both the 1.375 approximation and the $O(nlogn)$ running time. The approximation ratio is guaranteed by a new computational case analysis (Cunha et al., 2014a) that finds 11/8-sequences for bad small full configurations. The running time is attained by providing, for the first time, a correct linear-time strategy for finding and applying a (2,2)-sequence.

ACKNOWLEDGMENTS

This work has been partially supported by grants from Brazilian agencies Faperj, CNPq, and CAPES.

AUTHOR DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Bafna, V., and Pevzner, P.A. 1998. Sorting by transpositions. *SIAM J. Discrete Math.* 11, 224–240.

Bulteau, L., Fertin, G., and Rusu, I. 2012. Sorting by transpositions is difficult. *SIAM J. Discrete Math.* 26, 1148–1180.

Christie, D.A. 1999. Genome rearrangement problems [Ph.D. thesis]. University of Glasgow, UK.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2013a. Advancing the transposition distance and diameter through lonely permutations. *SIAM J. Discrete Math.* 27, 1682–1709.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2013b. On the 1.375-approximation algorithm for sorting by transpositions in O(nlogn) time. Proceedings of the 8th Brazilian Symposium on Bioinformatics, volume 8213 of Lectures Notes in Bioinformatics, pp. 126–135.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2014a. 1.375-approximation for SBT in O(n log n) time. Available at: http://compscinet.org/research/sbt1375 Accessed September 2, 2015.

Cunha, L.F.I., Kowada, L.A.B., de A. Hausen, R., and de Figueiredo, C.M.H. 2014b. A faster 1.375-approximation algorithm for sorting by transpositions. Proceedings of the 14th Workshop on Algorithms in Bioinformatics, volume 8701 of Lectures Notes in Bioinformatics, pp. 26–37.

Elias, I., and Hartman, T. 2006. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 369–379.

Feng, J., and Zhu, D. 2007. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Trans. Algorithms* 3, 1549–6325.

Fertin, G., Labarre, A., Rusu, I., et al. 2009. *Combinatorics of Genome Rearrangements*. The MIT Press, New York.

Firoz, J.S., Hasan, M., Khan, A.Z., and Rahman, M.S. 2011. The 1:375 approximation algorithm for sorting by transpositions can run in O(nlogn) time. *J. Comput. Biol.* 18, 1007–1011.

Hannenhalli, S., and Pevzner, P.A. 1999. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM* 46, 1–27.

Hartman, T., and Shamir, R. 2006. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Inf. Comput.* 204, 275–290.

Labarre, A. 2006. New bounds and tractable instances for the transposition distance. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 380–394.

Address correspondence to:
Luís Felipe I. Cunha
Centro de Tecnologia
Universidade Federal do Rio de Janeiro
Bloco H, Sala 317-10
Rio de Janeiro 21941972
Brazil

E-mail: lfignacio@cos.ufrj.br

Apêndice B

Anexo: Manuscrito "On the computational complexity of closest string problems"

On the computational complexity of closest string problems

Luís Felipe I. Cunha[a], Pedro Feijão[c], Vinícius F. dos Santos[c], Luis Antonio B. Kowada[b], Celina M.H. de Figueiredo[a]

[a] *Universidade Federal do Rio de Janeiro, Brazil*
[b] *Universidade Federal Fluminense, Brazil*
[c] *Universidade Federal de Minas Gerais, Brazil*
[d] *Universität Bielefeld, Germany*

Abstract

The Closest Object Problem aims to find one object in the center of all others. It was studied for strings with respect to the Hamming distance to be the metric to compute distances. The Hamming Closest String Problem (Hamming–CSP) was settled to be NP-complete in the case of binary strings. The Closest Permutation Problem was also studied, since permutations are the natural restrictions of general strings. A permutation is a string with a unique occurrence of each letter of an alphabet, and by considering the Cayley distance, the Cayley Closest Permutation Problem (Cayley–CPP) was settled to be NP-complete. In this paper, we consider well-known metrics to compute distances of permutations in the context of genome rearrangements, and we prove two NP-completeness: the block-interchange–CPP and the breakpoint–CPP.

Keywords: Closest permutation, NP-completeness, Hamming distance, block-interchange distance, breakpoint distance

1. Introduction

The Closest Permutation Problem (CPP) is a combinatorial challenge with applications in computational biology [15], where an input permutation set models a set of genomes, and we want to find a solution genome that is closely related to all others, i.e. a permutation corresponding to the *radius* of the input

☆Supported by FAPERJ, CAPES and CNPq.

Email addresses: `lfignacio@cos.ufrj.br` (Luís Felipe I. Cunha), `pfeijao@cebitec.uni-bielefeld.de` (Pedro Feijão), `viniciussantos@dcc.ufmg.br` (Vinícius F. dos Santos), `luis@ic.uff.br` (Luis Antonio B. Kowada), `celina@cos.ufrj.br` (Celina M.H. de Figueiredo)

Preprint submitted to Discrete Applied Mathematics (String Algorithms)

permutation set. Several metrics corresponding to genome rearrangements such as Cayley, Transposition, Block-interchange, Breakpoint, and Reversal have been studied with respect to the distance problem, but only a few with respect to the more general CPP [10]. Popov [16] studied the CPP regarding the Cayley metric, and proved that the Cayley–CPP is NP-complete. The CPP has not been studied regarding other metrics to compute distances, for instance with respect to a metric for which the distance problem is polynomially solvable, called the block-interchange [6], a generalization of the Cayley metric. There exist NP-complete distance problems (for instance, the transposition distance [4]), therefore we have that the CPP is also NP-complete for the same metric of distance (Transposition–CPP is NP-complete). However, if we restrict the input permutation set to some class, we may have the CPP polynomially solvable [7].

A more general problem, asking for a closest string regarding the Hamming distance, was proved to be NP-complete by Lanctot *et al.* [13]. Gramm *et al.* [11] proposed a *fixed-parameter* algorithm with respect to the radius parameter, and they also considered another variation, by asking for the closest substring of a given set of strings, where they proved to be $W[1]$-hard with respect to the number of input strings, which corroborates the hypothesis that the Hamming Closest String Problem is easier than the more general Hamming Closest Substring Problem.

In this paper, we consider the Closest Permutation Problem with respect to two well-known metrics, for which the distance problem is known to be polynomially solvable, by proving that the block-interchange and the breakpoint Closest Permutation Problems are NP-complete.

This paper is organized as follows: in Section 2 we define the Hamming Closest String Problem, the Closest Permutation Problem, and the metrics we deal with in this paper; in the subsequent sections we prove NP-completeness, where in Section 3 we prove that Block-interchange–CPP is NP-complete, and in Section 4 we prove that Breakpoint–CPP is NP-complete; and finally in Section 5 we discuss some open questions for further work about complexity and approximation algorithms on the closest and the related median problems.

2. Preliminaries

An *alphabet* Σ is a non empty set of letters, and a *string* over Σ is a finite sequence of letters of Σ. The *Hamming distance of two strings* of the same length s and σ denoted $d_H(s, \sigma)$ is defined as the number of mismatched positions between s and σ. The *Hamming distance of a string* s of length m denoted $d_H(s)$ is the Hamming distance of s and $\iota = 0^m$.

The Hamming Closest String Problem is defined as follows:

HAMMING CLOSEST STRING PROBLEM (H–CSP) INPUT: Set of strings $\{s_1, s_2, \ldots, s_\ell\}$ over alphabet Σ of length m each, and a non-negative integer f. QUESTION: Is there a string σ of length m such that $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$?

In case of positive answer for H–CSP, we call a *solution of H–CSP* any string σ that satisfies $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$.

H–CSP was proposed by Lanctot *et al.* [13], in the context of some biological problems, such as: discovering potential drug targets, creating diagnostic probes, universal primers or unbiased consensus sequences. All these problems reduce to the task of finding a pattern that, with some error, occurs in one set of strings (closest string problem). They also proved the following result.

Theorem 2.1. *[13] H–CSP is* NP*-complete for a binary alphabet.*

A *permutation* of length n is a particular string with a unique occurrence of each letter, since it is a bijection from the set $\{1, 2, \ldots, n\}$ onto itself $\pi = [\boldsymbol{\pi(0)}\, \pi(1)\, \pi(2) \cdots \pi(n) \boldsymbol{\pi(n+1)}]$, such that $\boldsymbol{\pi(0)} = 0$ and $\boldsymbol{\pi(n+1)} = n+1$. The operations we consider will never act on $\boldsymbol{\pi(0)}$ nor $\boldsymbol{\pi(n+1)}$.

The *union* of the permutations α and β of lengths n and m, respectively, is the permutation π constructed by the juxtaposition of α and β, $\pi = [\mathbf{0}\, \alpha(1)\, \alpha(2)\, \ldots \alpha(n)\, \beta(1)+n\;\; \beta(2)+n\;\; \ldots\;\; \beta(m)+n\;\; \boldsymbol{n+m+1}]$. For instance the permutation $[\mathbf{0}\ 1\ 2\ 3\ 4\ 7\ 6\ 5\ 8\ \mathbf{13}]$ is the union of $[\mathbf{0}\ 1\ 2\ 3\ 4\ \mathbf{5}]$ and $[\mathbf{0}\ 3\ 2\ 1\ 4\ \mathbf{5}]$.

Given a metric M and $d_M(p, \pi)$ the distance between permutations p and π with respect to the metric M, the *distance of a permutation* π of length m denoted $d_M(\pi)$ is the distance of π and the *identity permutation* $\iota = [\mathbf{0}\, 1\, 2\, \cdots\, n\, \boldsymbol{n+1}]$.

The CLOSEST PERMUTATION PROBLEM is defined as follows:

METRIC M CLOSEST PERMUTATION PROBLEM (M–CPP) INPUT: Set of permutations $\{p_1, p_2, \ldots, p_k\}$ of length n each, and a non-negative integer d. QUESTION: Is there a permutation π of length n such that $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$?

In case of positive answer for M–CPP, we call a *solution of M–CPP* any permutation π that satisfies $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$.

Given a set of permutations, the CLOSEST PERMUTATION PROBLEM aims to find a solution permutation that minimizes the maximum distance between the

solution and all other input permutations. The metric of distances depends on the context of the problem.

In this work, we consider two metrics: the *block-interchange distance*, and the *breakpoint distance*.

The *block-interchange* operation transforms one permutation into another one by exchanging two blocks, and generalizes the *transposition* operation (whose blocks are consecutive) and the *Cayley* operation (whose blocks are unitary). The *block-interchange distance of two permutations* is the minimum number of block-interchanges to transform one permutation into another one, and the *block-interchange distance of a permutation* π is the block-interchange distance of π and ι.

Note that a block-interchange generalizes a transposition and generalizes also a Cayley operation. Nevertheless, with respect to the distance problem, general operations do not imply the same computational complexity of more particular operations. For instance, with respect to the block-interchange distance, it can be computed in polynomial time [6], whereas the transposition distance is an NP-complete problem [4], and the Cayley distance is a polynomial problem.

On the other hand, if a distance problem is NP-complete, then the closest problem for the same operation is also NP-complete. Indeed, by considering the input set of permutations with two permutations π, ι such that $\pi \neq \iota$ and we ask for a permutation with distance for a metric M at most d for each, we can see the distance as the closest problem with a particular instance. Since, by the triangular inequality, it is necessary that $d_M(\pi, \iota) \leq 2d$.

Although the Transposition–CPP is NP-complete, if we consider some particular input sets then we can determine a closest permutation in polynomial time. For instance, let us consider the *toric equivalence*, which is an equivalence relation between permutations, where permutations in the same class have the same transposition distance [7]. Given a permutation for which we know its transposition distance (let us say it is equal to d), hence we know the distances of any other permutation and the distance of any pair of permutations in the same toric class. Therefore, if any $d' \geq d$ is the input radius asked in the problem, then the identity permutation is a solution permutation. Another example is by considering $n \leq 15$, we know the exact distance of any permutation as discussed in [7] (let us say d is the maximum distance of any permutation). Therefore, if any $d' \geq d$ is chosen to be the radius in the CPP, then the identity permutation is a solution.

The *breakpoint distance* is the number of consecutive elements in one permutation that are not consecutive in another one. Note that on the breakpoint distance we do not apply any operation to transform a permutation into another one.

Both problems of the block-interchange and the breakpoint distances were pro-

posed in the context of genome rearrangements, which is a field of bioinformatics and the two problems model phylogenetic distances between species compared by their genomes. Similar to the Hamming distance on strings with respect to the CSP problem, which was motivated by applications in bioinformatics, the metrics studied in the present paper are much studied by the bioinformatics community [6, 10].

We review next how to obtain in polynomial time the distances between permutations based on the reality and desire diagram, or on the breakpoints.

The block-interchange distance. Bafna and Pevzner [2] proposed a useful graph, the reality and desire diagram, which allowed non-trivial bounds on the transposition distance [2], and also provided, as established by Christie [6], the exact block-interchange distance.

Given a permutation π of length n, the *reality and desire diagram* $G(\pi)$ of π, is a multigraph $G(\pi) = (V, R \cup D)$, where $V = \{0, -1, +1, -2, +2, \cdots, -n, +n, -(n+1)\}$, each element of π corresponds to two vertices and we also include the vertices labeled by 0 and $-(n+1)$, and the edges are partitioned into two sets: the reality edges R and the desire edges D. The *reality edges* represent the adjacency between the elements on π, that is $R = \{(+\pi(i),\ -\pi(i+1))\ |i = 1, \cdots, n-1\} \cup \{(0,\ -\pi(1)),\ (+\pi(n),\ -(n+1))\}$; and the *desire edges* represent the adjacency between the elements on ι, that is $D = \{(+i,\ -(i+1))\ |i = 0, \cdots, n\}$. Fig. 1 illustrates the reality and desire diagram of a permutation.

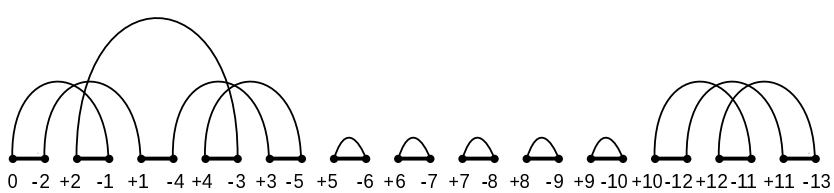

Figure 1: The reality and desire diagram of the permutation [**0** 2 1 4 3 5 6 7 8 9 10 12 11 **13**].

As a direct consequence of the construction of this graph, every vertex in $G(\pi)$ has degree 2, so $G(\pi)$ can be partitioned into disjoint *cycles*. We say that a cycle in π has length k, or that it is a k-cycle, if it has exactly k reality edges (or, equivalently, k desire edges). Hence, the identity permutation of length n has $n+1$ cycles of length 1. We denote $C(G(\pi))$ the number of cycles in $G(\pi)$.

After applying a block-interchange $b\ell$ in a permutation π, the number of cycles $C(G(\pi))$ changes in such a way that: $C(G(\pi b\ell)) = C(G(\pi)) + x$, such that $x \in \{-2, 0, 2\}$. The block-interchange $b\ell$ is thus classified as an *x-move* for π.

Christie [6] proved for the block-interchange operation the existence of a 2-

move for any permutation, which says that the number of cycles yields the exact block-interchange distance:

Theorem 2.2. *[6] The block-interchange distance of a permutation π of length n is $d_{BI}(\pi) = \frac{(n+1)-C(G(\pi))}{2}$.*

On the other hand, by allowing only the particular case of the transposition operation, a 2-move is not always possible to be found. We say a transposition *affects* a cycle if the extremities of the two blocks of the transposition eliminates a reality edge of a cycle and creates another edge. This new edge may increase, decrease, or keep the number of cycles.

Bafna and Pevzner [2] showed conditions of a cycle for a transposition to be an x-move. If a transposition t is a -2-move, then t affects three distinct cycles. However, if a transposition t is a 0-move or a 2-move, then t affects at least two elements of the same cycle [2]. We shall see that, when considering the block-interchange operation, it is useful to apply 0-move transpositions.

An interesting transformation in a permutation is the reduction, since the permutation obtained after the transformation preserves the block-interchange distance. The *reduced permutation* of π, denoted $gl(\pi)$, is the permutation whose reality and desire diagram $G(gl(\pi))$ is equal to $G(\pi)$ without the cycles of length 1, and has its vertices relabeled accordingly. For instance the reduced permutation corresponding to the permutation in Fig. 1 is [**0** 2 1 4 3 5 7 6 **8**]. Christie [6] proved an important equality.

Theorem 2.3. *[6] The block-interchange distances of a permutation π and its reduced permutation $gl(\pi)$ satisfy $d_{BI}(\pi) = d_{BI}(gl(\pi))$.*

The breakpoint distance. An *adjacency* (resp. a *reverse adjacency*) in a permutation π with respect to δ is a pair $(a,\ b)$ of consecutive elements in π such that $(a,\ b)$ (resp. the pair $(b,\ a)$) is also consecutive in δ. If a pair of consecutive elements is neither an adjacency nor a reverse adjacency, then $(a,\ b)$ is called a *breakpoint*, and we denote $b(\pi, \delta)$ the number of breakpoints of π with respect to δ. Hence, the breakpoint distance between π and δ is $d_{BP}(\pi, \delta) = b(\pi, \delta)$.

Next, we apply transformations from a generic instance of H–CSP to particular instances of M–CPP with respect to the block-interchange, and the breakpoint metrics. In each one, we establish a relationship between the Hamming distance of binary strings and the distance on the corresponding metric on permutations.

3. Block-interchange–CPP is NP-complete

Firstly, we apply Algorithm 1 that transforms an arbitrary binary string s of length m into a particular permutation λ_s of length $2m$.

Algorithm 1: $Permut_{BI}(s)$

input : Binary string s of length m
output: Permutation λ_s

1 each occurrence of 0 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i-1$ and $2i$, respectively.
2 each occurrence of 1 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i$ and $2i-1$, respectively.

Note that any permutation obtained in Algorithm 1 is constructed by successive unions of [**0** 1 2 **3**] and [**0** 2 1 **3**]. Fig. 2 illustrates the construction of a permutation for a given string with respect to Algorithm 1.

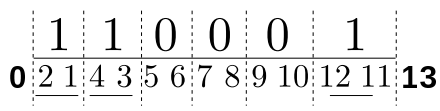

Figure 2: Permutation $\lambda_s = [\mathbf{0}\,2\,1\,4\,3\,5\,6\,7\,8\,9\,10\,12\,11\,\mathbf{13}]$ obtained from Algorithm 1 — its reality and desire diagram is in Fig. 1 — where $s = 110001$, with reduced permutation $gl(\lambda_s) = [\mathbf{0}\ 2\ 1\ 4\ 3\ 5\ 7\ 6\ \mathbf{8}]$, and number of cycles of its reality and desired diagram $C(G(gl(\lambda_s))) = 2$. The Hamming distance of s is equal to the Block-interchange distance of λ_s.

Lemma 3.1. *Given a string s of length m and the permutation λ_s of length $2m$ obtained in Algorithm 1, then the reduced permutation $gl(\lambda_s)$ has length n', where $2d_H(s) \leq n' \leq 3d_H(s) - 1$.*

Proof. If the string s is $s = 1^{d_H(s)}0^{m-d_H(s)}$ (or $s = 0^{m-d_H(s)}1^{d_H(s)}$), then to obtain the reduced permutation of λ_s we remove $2(m - d_H(s))$ elements. Therefore, the associated permutation has length $n' = 2m - 2(m - d_H(s)) = 2d_H(s)$. On the other hand, if s is $s = (01)^{\frac{m}{2}}$ (or $s = (10)^{\frac{m}{2}}$), then each adjacency $2i-1, 2i$ is removed to obtain the reduced permutation, excepted the first adjacency $1, 2$ (or the last one $2m-1, 2m$), for which both elements are removed. So, the length of the reduced permutation is $n' = 2d_H(s) + d_H(s) - 1$. Since these are the cases of maximum and minimum number of 0s adjacent, hence they correspond to the minimum and maximum lengths of the associated permutations, respectively. □

Lemma 3.2. *If λ_s is a permutation obtained in Algorithm 1 and $gl(\lambda_s)$ its reduced of length $2d_H(s) + x$, for $0 \leq x \leq d_H(s) - 1$, then $C(G(gl(\lambda_s))) = x + 1$.*

Proof. There exist x contiguous sequences of bits 0 in s, and a sequence of bits 0 is between two contiguous sequences of bits 1. It implies a cycle in the reality and desire diagram for each contiguous sequence of bits 1. □

Next, we establish the key equality between the Hamming distance of an input string and the block-interchange distance of its output permutation obtained from Algorithm 1.

Lemma 3.3. *Given λ_s the permutation obtained from a binary string s by Algorithm 1, the block-interchange distance of λ_s is equal to the Hamming distance of s, $d_{BI}(\lambda_s) = d_H(s)$.*

Proof. Since $d_{BI}(\lambda_s) = d_{BI}(gl(\lambda_s))$, from Lemma 3.2 we have that $d_{BI}(gl(\lambda_s)) = \frac{2d_H(s)+x+1-(x+1)}{2}$, which implies $d_{BI}(\lambda_s) = d_H(s)$. □

Now, we show how a solution for the H–CSP implies in a solution for the block-interchange–CPP, and vice versa.

Lemma 3.4. *Given a set of k permutations obtained by Algorithm 1, there is a block-interchange closest permutation with max distance at most d if, and only if, there is a Hamming closest string with max distance equal to d.*

Proof. ($\Rightarrow$) If λ' can be built by Algorithm 1 for some input string s', then, by Lemma 3.3, s' is a closest string.

Otherwise, given a solution permutation, we search from the left to the right to find the first position where the corresponding element is different from the one intended to be by the algorithm, which can be a position $2i-1$ or a position $2i$. In each case, we transform to a new permutation with a longer prefix agreeing with the algorithm output, without increasing the distance to any input permutation.

Hence, we apply transpositions on the solution permutation to obtain a new one. To guarantee that the distance of this new permutation and every one of the input is not increasing, we show in each case, the worst operation is a 0-move with respect to every input permutation, since such transposition affects elements of either the same cycle in the reality and desire diagram, or it creates new adjacencies.

By repeating this process, a string agreeing with the algorithm output can be found and, by Lemma 3.3, a string with maximum distance equal to d can be constructed.

1. in position $2i-1$ there is a correct element, but in position $2i$ there is not the element $2i-1$ nor the element $2i$, either.

Given any solution permutation, the elements until the position $2i-1$ are correct, but an element $a > 2i$ is in position $2i$. Let a' be an adjacency of a with respect to all input permutations. Let us assume without loss of generality $i = 2$. Since

if $i > 2$, then all elements between 1 and $2i-2$ are already before the position $2i-1$. Hence, we consider a solution permutation $[1\,2\,3\,a\,\ldots\,4\,\ldots]$, such that it can be either: i) $[1\,2\,3\,a\,\ldots\,4\,\ldots\,a'\,\ldots]$, or ii) $[1\,2\,3\,a\,\ldots\,a'\,\ldots\,4\,\ldots]$. Therefore, we compare this solution to any kind of input permutation.

If the solution is i), we obtain $[1\,2\,3\,\underline{a\,\ldots}\,\underline{4\,\ldots\,a'}\,\ldots] \rightarrow [1\,2\,3\,4\,\ldots\,a'\,a\,\ldots]$. For each possible input permutation, we justify below that the distance between the new permutation and each input does not increase.

- Case 1. If an input has the adjacency a, a', then:

  Subcase 1.1: $[1\,2\,3\,4\,\ldots\,a\,a'\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency $3, 4$. Hence, such transposition is at least a 0-move;

  Subcase 1.2: $[1\,2\,4\,3\,\ldots\,a\,a'\,\ldots]$, the elements a and a' are in the same cycle. Hence, such transposition is at least a 0-move;

  Subcase 1.3: $[2\,1\,3\,4\,\ldots\,a\,a'\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a, a'. Hence, such transposition is at least a 0-move, similar the Subcase 1.1;

  Subcase 1.4: $[2\,1\,4\,3\,\ldots\,a\,a'\,\ldots]$, the elements $0, 1, 2, 3, 4$ are in the same cycle. Hence, such transposition is at least a 0-move, since it affects the elements 3, 4.

- Case 2. If an input has the adjacency a', a, then:

  Subcase 2.1: $[1\,2\,3\,4\,\ldots\,a'\,a\,\ldots]$, we are creating two cycles of length 1, by creating the adjacencies $3, 4$ and a', a;

  Subcase 2.2: $[1\,2\,4\,3\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a', a. Hence, such transposition is at least a 0-move;

  Subcase 2.3: $[2\,1\,3\,4\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency $3, 4$. Hence, such transposition is at least a 0-move, similar the Subcases 1.1 and 1.3;

  Subcase 2.4: $[2\,1\,4\,3\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a', a. Hence, such transposition is at least a 0-move.

If the solution is ii), we obtain $[1\,2\,3\,\underline{a\,\ldots}\,\underline{4\,\ldots\,b}\,c\,\ldots] \rightarrow [1\,2\,3\,4\,\ldots\,b\,a\,\ldots]$, for the pair b, c being is a universal breakpoint, i.e. a breakpoint with respect to all input permutations. Note the existence of such breakpoint, since the element a is before the element 4 in the permutation and there is some place at the right of 4

where a should be. For each possible input permutation we justify below that the distance between the new permutation and each input does not increase.

In Subcases 1.1, 2.1, 1.3, 2.3 we are creating one cycle of length 1, by the adjacency $3, 4$. Hence, such transposition is at least a 0-move; The remaining cases the transposition applied is between elements of same cycle.

2. in position $2i-1$ there is not the element $2i-1$ nor the element $2i$, either.

In this case we assume the solution is $[1\,2\,a\,\ldots\,3\,\ldots\,4\ldots]$. We consider the inputs with the adjacency $3, 4$ or the adjacency $4, 3$, so we do not need to deal with the case of the solution $[1\,2\,a\,\ldots\,4\,\ldots\,3\ldots]$.

In all cases $[1\,2\,4\,3\,\ldots]$, $[1\,2\,3\,4\,\ldots]$, $[2\,1\,4\,3\,\ldots]$, and $[2\,1\,3\,4\,\ldots]$, the elements $2, 4$ are in the same cycle. Hence, any transposition affecting such elements that fix element 4 after 2 is at least a 0-move.

($\Leftarrow$) Given a solution string s, we obtain the associated permutation λ_s given by Algorithm 1. By Lemma 3.3 we have the solution s regarding the H–CSP corresponding to the permutation λ_s with the same value of max distance d. $\square$

Theorem 3.1. *The* Block-interchange–CPP *is* NP*-complete.*

4. Breakpoint–CPP is NP-complete

Firstly, we apply Algorithm 2 that transforms an arbitrary binary string s of length m into a particular permutation β_s of length $4m$.

Algorithm 2: $Permut_{BP}(s)$

input : Binary string s of length m.
output: Permutation β_s

1 each occurrence of 0 in position i corresponds to the elements $4i-3$, $4i-2$, $4i-1$ and $4i$ in positions $4i-3$, $4i-2$, $4i-1$, $4i$, respectively.
2 each occurrence of 1 in position i corresponds to the elements $4i-3$, $4i-2$, $4i-1$ and $4i$ in positions $4i-2$, $4i-3$, $4i-1$, $4i$, respectively.

Note that any permutation obtained in Algorithm 2 is constructed by successive unions of [**0** 1 2 3 4 **5**] and [**0** 2 1 3 4 **5**]. Fig. 3 illustrates the construction of a permutation for a given string with respect to Algorithm 2.

Next, we establish the following relationship between the Hamming distance of an input string and the Breakpoint distance of its output permutation obtained from Algorithm 2.

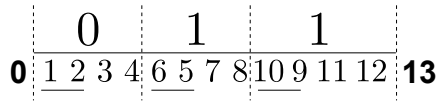

Figure 3: Permutation $\beta_s = [\mathbf{0}\ 1\ 2\ 3\ 4\ 6\ 5\ 7\ 8\ 10\ 9\ 11\ 12\ \mathbf{13}]$ where $s = 011$, obtained from Algorithm 2, with the breakpoint distance $d_{BP}(\beta_s) = 4$, since the breakpoints are (4, 6), (5, 7), (8, 10) and (9, 11). The Hamming distance of s is $d_H(s) = \frac{d_{BP}(\beta_s)}{2} = 2$.

Lemma 4.1. *Given β_s the permutation obtained from a binary string s by Algorithm 2, the Breakpoint distance of β_s is $d_{BP}(\beta_s) = 2d_H(s)$.*

Proof. Each element 1 of the binary string yields an exchange between two consecutive elements, hence we are creating exactly two breakpoints. □

Now, we show how a solution for the H–CSP implies in a solution for the Breakpoint–CPP, and vice versa.

Lemma 4.2. *Given a set of k permutations obtained by Algorithm 2, there is a breakpoint closest permutation with max distance equal to $2d$ if, and only if, there is a Hamming closest string with max distance equal to d.*

Proof. ($\Rightarrow$) If β' can be built by Algorithm 2 for some input string s', then, by Lemma 4.1, s' is a closest string.

Otherwise (similar to Lemma 3.4), given any solution permutation we search from the left to the right to find the first position where the corresponding element is different from the one intended to be by the algorithm. In each case, we transform to a new permutation with a longer prefix agreeing with the algorithm output, without increasing the distance to every input permutation. By repeating this process a string agreeing with the algorithm output can be found and, by Lemma 4.1, a string with maximum distance equal to d can be constructed.

We consider each case of a position not agreeing with an element intended to be by Algorithm 2.

- position $4i - 3$: We call $a = 4i - 3$ and $b = 4i - 2$ the possible elements that could be in this position.
  - If a and b are not consecutive to the right of the position $4i - 3$, then there is a universal breakpoint, *i.e.* a breakpoint with respect to all input permutations, on the right of a or on the right of b. In this case we apply a flipping from the position $4i - 3$ until such universal breakpoint.

- If a and b appear consecutive to the right of the position $4i-3$, then there is a universal breakpoint. If such breakpoint is on the right of a (in the case of b, a) or on the right of b (in the case of a, b), then we apply a flipping from the position $4i-3$ until such universal breakpoint.

  From now on, we consider a and b consecutive, but there is a breakpoint on the left of such pair.
- If there is a universal breakpoint on the right of the pair a, b (or b, a), then we apply a transposition such that the first block starts at position $4i-3$ and ends at the element before the pair a, b (or b, a), and the second block ends at the universal breakpoint.
- If there is not a universal breakpoint on the right of the pair a, b (or b, a), then every consecutive pair after the position $4i-3$ has difference at most 2, since any pair of consecutive elements is also a pair in an input permutation, for which by Algorithm 2 such property holds. Let x, y, z be the elements in positions $4i-3, 4i-2, 4i-1$, respectively. Since the a, b (or b, a) are both less than x, hence the pair $x-1, x+1$ appears consecutive and on the right of a, b (or b, a), otherwise such difference would be greater than 2, since x is in position $4i-3$.
  * If the transposition putting x between $x-1$ and $x+1$ does not create any breakpoint, then we apply such transposition. The resulted permutation is approximating to our intended solution, since the element $4i-3$ is becoming closer to the intended position.
  * If such transposition putting x between $x-1$ and $x+1$ creates a breakpoint with respect to some input permutation, but it only happens when x, y is an adjacency in such input permutation. In this case, $y = x-2$ or $y = x+2$. Without loss of generality, we assume $y = x-2$. Since the difference between two consecutive elements is at most 2, hence $x-3$ is also on the right of a, b and is consecutive to $x-1$. So, we apply a transposition to put x between $x-1$ and $x+1$, and afterwards we apply a transposition to put $y = x-2$ between $x-3$ and $x-1$. Now, we prove that in this case the number of breakpoints does not increase for no one permutation. For any input permutation, after the transpositions we have at most 3 breakpoints: one on the left of z, another one on the left or the right of x, and another one on the left or the right of y. Before the transpositions we have at least 3 breakpoints: by the construction, one between $x-3$ and $x-1$ or between $x-1$ and $x+1$. Similarly, we have at least one breakpoint between x and y, or between y and z (if yz is not a universal breakpoint, then $z = x-4$, and for the same reason $x, x-2, x-4$ cannot be all

adjacencies). Moreover, we have a third breakpoint, on the left of x. Therefore, for any input permutation, the number of breakpoints does not increase after these transpositions.

- position $4i-2$. If $4i-3$ is already correct, we apply a flipping from the element in position $4i-2$ until the element $4i-2$, this operation always decreases the number of breakpoints, since the element $4i-3$ is adjacent to the element $4i-2$ with respect to all input permutations. If after such operation we have created a breakpoint after the position where the $4i-2$ is, then there is no problem, since we are removing one breakpoint in putting $4i-3$ adjacent to $4i-2$. Hence, we are not increasing the number of breakpoints.

- position $4i-1$. If there is a universal breakpoint on the right of the element $4i-1$, the we apply a flipping and the number of breakpoints does not increase. Otherwise, the element $4i$ is adjacent on the right of $4i-1$, hence we apply a transposition putting the $4i-1, 4i$ in the correct positions. In the worst case the number of breakpoints is the same, since we are creating a breakpoint between the elements before $4i-1$ and after the element $4i$, but creating an adjacency between $4i-2$ and $4i-1$.

- position $4i$. In this case we can apply a flipping form the position $4i$ until the position where the element $4i$ is. For the same reason of the second case, the number of breakpoints is not increasing.

($\Leftarrow$) Given a solution string s, we obtain the associated permutation β_s given by Algorithm 2. By Lemma 4.1 we have the solution s regarding the H–CSP corresponding to the permutation β_s with the value of max distance equal to $2d$. □

Theorem 4.1. *The* Breakpoint–CPP *is* NP*-complete.*

5. Further Work

This paper describes the complexity of the Closest Permutation Problem with respect to two well-known metrics. Table 1 summarizes the state of the art of the computational complexity of the distance, closest and median problems with respect to five well-known metrics. We find in the second row the complexity status for the closest problem, with respect to the two metrics studied in this paper (block-interchange and breakpoint), for the transposition, and for the *double cut and join* (DCJ) and the *single cut or join* (SCJ), which are two well-known metrics studied in the context of comparative genomics [8, 9], but so far not considered with respect to the closest problem.

Despite the hardness to decide the closest permutation with respect to the metrics shown in the present paper, some interesting questions arise.

	Block-interchange	Breakpoint	Transposition	DCJ	SCJ
Distance	Polynomial	Polynomial	NP-complete	Polynomial	Polynomial
Closest	NP-complete	NP-complete	NP-complete	Open	Open
Median	Open	NP-complete	NP-complete	NP-complete	Polynomial

Table 1: Computational complexity of the distance, closest and median problems.

How to improve the 2-approximation algorithm that computes all pairwise distances?. By the triangular inequality, a necessary condition for a permutation to be the center with radius at most d is that the distance of any pair in the input permutation set must be at most $2d$. Hence, if such condition is true for a given input permutation set, then any permutation of the input is a solution with approximation ratio 2 of an optimal solution. Since the two considered metrics admit polynomial algorithms to compute the distances, one question would be to lower the approximation ratio.

What can be said about the Kendall-τ Median problem*?.* A related problem is the Median Problem, where we ask for the solution string/permutation that minimizes the sum of the distances between the solution and the input string/permutation set. The Hamming Median String Problem is a polynomial problem [11], but regarding permutations the breakpoint [14], transposition [1], and reversal [5] Median Permutation Problems are NP-complete. For the DCJ, Tannier *et al.* [18] proved that DCJ Median problem is NP-complete, and Feijão and Meidanis [8] proved that the SCJ Median problem is polynomial, as described in Table 1.

The Kendall-τ operation, also known as the *bubble sort*, is an exchange between two consecutive elements in a permutation. Hence, the Block-interchange, as well as the Transposition and the Cayley, are all generalizations of the Kendall-τ operation. The Kendall-τ Median problem is known to be NP-complete, but its complexity is open when the input set has three permutations [3].

Note that the relationship between the closest and the median problems often appears in classical combinatorial optimization problems. The closest problem is a min-max problem and the median problem is a min-sum problem. For instance, in graph theory, given two sets of vertices, there are the min-max disjoint path and the min-sum disjoint path problems. These path problems were considered in several papers, where they are polynomial or NP-complete, according to distinct classes of graphs [12, 17]. Hence, it is interesting to investigate and contrast

the computational complexity of the closest and the median problems for several metrics of distances.

[1] Bader, M.: The transposition median problem is NP-complete. Theor. Comput. Sci., 412, 1099–1110 (2011)

[2] Bafna, V., Pevzner, P. A.: Sorting by transpositions. SIAM J. Discrete Math, 11, 224–240 (1988)

[3] Blin, G., Crochemore, M., Hamel, S., Vialette, S.: Medians of an odd number of permutations. Pure Math. Appl., 21(2), 161–175 (2010)

[4] Bulteau, L., Fertin, G., Rusu, I. Sorting by transpositions is difficult. SIAM J. Discrete Math, 26(3), 1148–1180 (2012)

[5] Caprara, A.: The reversal median problem. INFORMS J. Comput., 15, 93–113 (2003)

[6] Christie, D. A.: Sorting permutations by block-interchange. Inf. Process. Lett., 60, 165–169 (1996)

[7] Cunha, L. F. I., Kowada, L. A. B., Hausen, R. A., Figueiredo, C. M. H.: Advancing the transposition distance and the diameter through lonely permutations. SIAM. J. Discrete Math., 27(4), 1682–1709 (2013)

[8] Feijão, P., Meidanis, J.: SCJ: a breakpoint-like distance that simplifies several rearrangement problems. IEEE IEEE/ACM Transactions on Computational Biology and Bioinformatics, 8:1318-1329 (2011)

[9] Friedberg R, Darling, A.E., Yancopoulos, S.: Genome rearrangement by the double cut and join operation.. Methods Mol Bio. 285 (452): 385-416 (2008)

[10] Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. The MIT Press (2009)

[11] Gramm, J., Niedermeier, R., Rossmanith, P.: Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems. Algorithmica, 37, 25–42 (2003)

[12] Kobayashi, Y., Sommer, C.: On shortest disjoint paths in planar graphs. Discrete Optim., 7, 234–245 (2010)

[13] Lanctot, J. K., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Inform. and Comput., 185(1), 41–55 (2003)

[14] Pe'er, I., Shamir, R.: The median problems for breakpoints are NP-complete. Technical report TR98-071. The Electronic Colloquium on Computational Complexity (1998)

[15] Pevzner, P. A.: Computational Molecular Biology: An Algorithmic Approach. The MIT Press (2000)

[16] Popov, V. Y.: Multiple genome rearrangement by swaps and by element duplications. Theor. Comput. Sci., 385, 115–126 (2007)

[17] Robertson, N., Seymour, P. D.: An outline of a disjoint paths algorithm. in: Paths, Flows, and VLSI-Layout, Springer-Verlag, 267–292 (1990)

[18] Tannier, E., Zheng, C., Sankoff, D..: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics, 10:120 (2009)

Apêndice C

Anexo: Manuscrito "Sorting by short block-moves and the complexity of the short block-move closest permutation problem"

SORTING BY SHORT BLOCK-MOVES AND THE COMPLEXITY OF THE SHORT BLOCK-MOVE CLOSEST PERMUTATION PROBLEM

LUÍS FELIPE I. CUNHA*, VINICIUS F. DOS SANTOS†, LUIS ANTONIO B. KOWADA‡, RODRIGO DE A. HAUSEN§, ANTHONY LABARRE¶, AND CELINA M. H. DE FIGUEIREDO†

Abstract. *Sorting by transpositions* (SBT) is an NP-complete problem, which asks for the minimum number of adjacent block exchanges required to sort a given permutation. A restricted form of SBT, whose computational complexity is still open, is *sorting by short block-moves* (SSBM), where the sum of the lengths of the exchanged blocks is at most 3. We identify classes of *non-reduced* permutations already studied for the SBT problem that remain tractable for the SSBM problem; identify an equivalence relation such that permutations in the same equivalence class have the same SSBM distance; give a family of permutations for which an optimal sorting sequence of transpositions is of short block-moves; provide a sufficient condition to determine the SSBM distance by showing that an optimal solution can be obtained by sorting each connected component of the permutation graph separately.

In this paper, we also address the *closest permutation problem* (CPP) for the short block-move distance, i. e., finding a permutation whose distance to every other permutation is bounded by a given integer. The closest string problem (CSP) was studied for the Hamming distance, and was shown to be NP-complete. From the *Hamming*–CSP we prove that the *short block-move*–CPP is also NP-complete.

Key words. Short block-move, Closest permutation problem, NP-completeness

1. Introduction. *Sorting by transpositions* (SBT) is a classical problem in genome rearrangements, in which genomes are represented by permutations, and aims at finding the minimum number of adjacent block exchanges that transforms a given permutation into the identity permutation. This number is called the *transposition distance*. SBT is an NP-complete problem [2], tight bounds on the distance are known [1, 13], but known tractable classes for the transposition distance are rare [5, 13].

Heath and Vergara [8] studied restricted forms of SBT, in which the sum of the lengths of the exchanged blocks is bounded. Sorting permutations by short block-moves (SSBM) refers to the variant in which the sum of those lengths is at most 3. The complexity of SSBM is open, although some results are known: a 5/4-approximation [10] has been proposed and some tractable classes of permutations have been identified [9, 11].

The *closest permutation problem* (CPP) considers an input permutation set, and aims to find a permutation minimizing the *radius* of the input permutation set. Popov [17] studied the CPP in the case of the Cayley distance, and proved that the resulting problem is NP-complete. CPP was also studied using other polynomially computable distances, such as the breakpoint and the block-interchange distances, where we have settled that these problems are NP-complete [4]. Another variant of the closest problem was considered for strings, using the Hamming distance, and

*Universidade Federal do Rio de Janeiro {lfignacio,celina}@cos.ufrj.br
†Universidade Federal de Minas Gerais, Brazil viniciussantos@dcc.ufmg.br
‡Universidade Federal Fluminense, luis@vm.uff.br
§Universidade Federal do ABC, hausen@compscinet.org
¶Université Paris-Est Marne-la-Vallée, France anthony.labarre@univ-mlv.fr

Lanctot *et al.* [15] proved that the Hamming–CSP is NP-complete.

This paper addresses both SSBM and CPP. Section 2 contains some preliminary definitions. Section 3 discusses the reduction operation for permutations and concludes with the exact SSBM distance for permutations whose reduced representation belongs to two well studied classes with respect to SBT: the reverse and the α-permutation. Section 4 presents the toric equivalence relation and a proof that this relation preserves the p-bounded block-move distance, for $p > (n+1)/2$; also in this section, the reverse complement operation is explained, along with a demonstration that this operation preserves the distance for all bounded block-move problems. Since the short block-move distance is an upper bound for the transposition distance, Section 5 displays permutations whose transposition distance equals the short block-move distance. The strategy used to find the SSBM distance for those permutations leads to a proof that every permutation can be optimally sorted by short block-moves through sorting each connected component of the permutation graph separately, which is a property that holds for SSBM but neither holds for greater bounded block-moves, nor for SBT in general. In Section 6, we prove that SSBM–CPP is NP-complete, via a transformation from Hamming–CSP. Section 7 concludes the paper.

2. Background and notation.

Transposition distance and short block-move distance. Given an integer n, a *permutation* $\pi_{[n]}$ is a bijection of the set $[n] = \{1, \cdots, n\}$ onto itself such that $\pi(i) = \pi_i$, denoted by $\pi_{[n]} = \begin{smallmatrix} 1 & 2 & \cdots & n \\ [\pi_1 & \pi_2 & \cdots & \pi_n] \end{smallmatrix}$, or by $\pi_{[n]} = [\pi_1\ \pi_2\ \cdots\ \pi_n]$. The *length* of $\pi_{[n]}$ is n. A *transposition* $t(i,j,k)$, where $1 \le i < j < k \le n+1$, is a permutation that exchanges the contiguous blocks $i\ i{+}1 \cdots j{-}1$ and $j\ j{+}1 \cdots k{-}1$; when composed with a permutation $\pi_{[n]}$, it yields $\pi_{[n]} \cdot t(i,j,k) = [\pi_1\ \pi_2\ \cdots \pi_{i-1}\ \underline{\pi_j\ \cdots\ \pi_{k-1}}\ \underline{\pi_i\ \ldots \pi_{j-1}}\ \pi_k \cdots \pi_n]$.

A *p-bounded block-move* is a transposition $t(i,j,k)$ such that $k - i \le p$, and a 3-bounded block-move is called a *short block-move*. Hence, a short block-move is either a transposition $t(i, i+1, i+2)$—called a *skip*—a transposition $t(i, i+1, i+3)$, or a transposition $t(i, i+2, i+3)$—both called *hops*. The *transposition distance* $d_t(\pi_{[n]})$ is the minimum number of transpositions needed to transform $\pi_{[n]}$ into the *identity permutation* $\iota = [1\ 2\ 3\ \cdots\ n]$; if the transpositions are restricted only to p-bounded block-move, then one obtains the *p-bounded block-move distance* $d_{pbbm}(\pi_{[n]})$, we call the *short block-move distance* $d_{sbm}(\pi_{[n]})$ when $p = 3$.

Previous works investigated variants of block-move distances where bounds are imposed on the lengths of at least one of the blocks moved [8, 9]. The problem of sorting permutations using 2-bounded block-moves, i.e. adjacent swaps, is easily solved by the *Bubble-Sort* algorithm [12]. In general, the complexity of the problem of sorting a permutation by p-bounded block-moves is unknown for fixed $p > 2$, whereas the analogous problem of limiting $k - i \le f(n)$, is NP-hard [8], since SBT is NP-hard.

Bounds on SBT and on SSBM. Some structures have been proposed to provide bounds on both SBT and SSBM distances. Given a permutation $\pi_{[n]}$, the *breakpoint graph* [1] is the graph $BG(\pi_{[n]}) = (V, R \cup D)$ that has $V = \{0,\ -1,\ +1,\ -2,\ +2,\ \ldots, -n,\ +n,\ -(n+1)\}$ as its vertex set, and whose edges are partitioned into two sets: the *reality edges* R and the *desire edges* D, where $R = \{(+\pi_i, -\pi_{i+1}) \mid i = 1, \ldots, n-1\} \cup \{(0, -\pi_1), (+\pi_n, -(n+1))\}$, and $D = \{(+i, -(i+1)) \mid i = 0, ..., n\}$. From this definition, it follows that $BG(\pi_{[n]})$ can be partitioned into disjoint cycles. A cycle that has k reality edges (or, equivalently, k desire edges) is said to have *length* k. Using the number of cycles of odd length in $BG(\pi_{[n]})$, denoted by $c_{odd}(\pi_{[n]})$, Bafna

and Pevzner [1] proved a lower bound on the SBT distance:

$$d_t(\pi_{[n]}) \geq \left\lceil \frac{(n+1) - c_{odd}(\pi_{[n]})}{2} \right\rceil. \tag{2.1}$$

To estimate the SSBM distance, Heath and Vergara [8, 9] used the *permutation graph* $PG(\pi_{[n]}) = (V_\pi^p, E_\pi^p)$, where $V_\pi^p = \{1,\ 2,\ \ldots, n\}$ and $E_\pi^p = \{(i,j) \mid \pi_i > \pi_j,\ i < j\}$; each edge of PG is called an *inversion* in π. Heath and Vergara proved that in on SSBM shortest sequence for $\pi_{[n]}$, every short block-move decreases the number of inversions by at least one unit, and by at most two units, therefore:

$$\left\lceil \frac{|E_\pi^p|}{2} \right\rceil \leq d_{sbm}(\pi_{[n]}) \leq |E_\pi^p|. \tag{2.2}$$

Given a permutation, our aim is to minimize the number of operations that decrease only one inversion in PG. Examples of permutations that are tight with respect to the above the lower and upper bounds are [2 4 3 5 1] and [2 1 4 3 6 5], respectively.

We can apply the graph-theoretic terminology directly to permutations instead of their permutation graphs; for instance, we may say that π is connected—meaning that its permutation graph is—or that a permutation σ is a connected component of a permutation π—meaning that the permutation graph of σ is a connected component of the permutation graph of π.

Heath and Vergara [8] proposed another auxiliary graph for obtaining bounds on the SSBM distance. The *arc graph* $A_\pi^a = (V_\pi^a, E_\pi^a)$ of π is an undirected graph with vertex set $V_\pi^a = E_\pi^p$, and two arcs $(a,b), (c,d)$ are adjacent if either:

i) $a = c$, and in π, between b and d there is no element x such that $b < x < d$; or

ii) $b = d$, and in π, between a and c there is no element x such that $a < x < c$.

It follows from the definition that A_π^a is a subgraph of the line graph of $PG(\pi_{[n]})$.

Heath and Vergara [8] noted that a hop is represented by an edge on the arc graph, therefore a maximum matching M on the arc graph may yield a maximization on the number of the hops to be applied on π, whereas the unmatched vertices U represents the number of skips, therefore:

$$d_{sbm}(\pi_{[n]}) \geq |M| + |U|. \tag{2.3}$$

The bound obtained in Equation (2.3) is, in general, greater than the bound in Equation (2.2). If there is a perfect matching in A_π^a, then both bounds are equal. Heath and Vergara [8] showed a class of permutations where Equation (2.3) yields an optimal sorting sequence of SSBM. Figure 2.1 illustrates the arc graph of [2 4 6 8 1 3 5 7] and how we achieve the sequence of short block-moves to sort it. Note that there is no edge $\{(8,1),(8,5)\}$, since element 3 satisfies $1 < 3 < 5$ and it is between 1 and 5 in the permutation.

A short block-move is a *correcting* move if it is a skip that eliminates one inversion, or a hop that eliminates two inversions in π. Otherwise, the block-move is called *non-correcting*. Heath and Vergara [9] proved that each sorting sequence can be performed

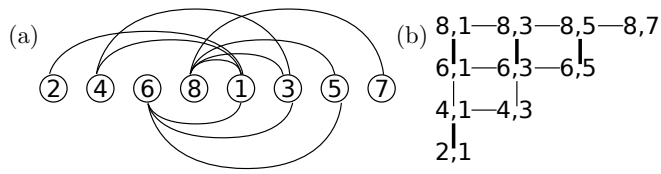

FIG. 2.1. *(a) $PG([2\,4\,6\,8\,1\,3\,5\,7])$ (b) $A^a_{[2\,4\,6\,8\,1\,3\,5\,7]}$. Bold edges are those of the maximum matching $M = \{\{(6,1),(8,1)\},\{(6,3),(8,3)\},\{(6,5),(8,5)\},\{(2,1),(4,1)\}\}$, the set of unmatched vertices is $U = \{(4,3),(8,7)\}$. The first short block-move is obtained by searching for the rightmost possible move involving pairs in M or vertices in U, for instance we take the pair $(6,1),(8,1)$ applying the hop $[2\ 4\ \underline{6\ 8}\ \underline{1}\ 3\ 5\ 7] \rightarrow [2\ 4\ 1\ 6\ 8\ 3\ 5\ 7]$. Hence, we continue the search in M and U.*

by using just correcting moves. Table 2.1 shows replacements from non-correcting moves to correcting moves in an optimal sorting sequence, which we will use it later in Theorem 5.2.

case	π	$\pi' = \pi\beta_i$	$\pi'' = \pi\beta'_i$
1	$\cdots ef \cdots$	$\cdots fe \cdots$	$\cdots ef \cdots$
2	$\cdots exf \cdots$	$\cdots xfe \cdots$	$\cdots xef \cdots$
3	$\cdots exf \cdots$	$\cdots fex \cdots$	$\cdots efx \cdots$
4	$\cdots xef \cdots$	$\cdots fxe \cdots$	$\cdots exf \cdots$
5	$\cdots efx \cdots$	$\cdots fxe \cdots$	$\cdots exf \cdots$

TABLE 2.1

How to replace a non-correcting move β_i with a correcting move β'_i [9]; in all cases, $e < f$, and x is arbitrary.

Closest string and closest permutation. An *alphabet* Σ is a non empty set of *letters*, and a *string* over Σ is a sequence of letters of Σ. The *Hamming distance between two strings* s_i and σ, denoted by $d_H(s_i, \sigma)$, is defined as the number of mismatched positions between s_i and σ. We call the *Hamming distance of a string* the number of mismatched positions between s_i and 0^m, such that s_i has length m. The HAMMING CLOSEST STRING PROBLEM is:

> HAMMING CLOSEST STRING PROBLEM (HAMMING–CSP)
> INPUT: A set of strings $\{s_1, s_2, \ldots, s_\ell\}$ over an alphabet Σ each of length m, and a non-negative integer f.
> QUESTION: is there a string σ of length m such that $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$?

If a positive answer exists for Hamming–CSP, we call a *solution to Hamming–CSP* any string σ that satisfies $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$. Lanctot *et al.* [15] proved the following result.

THEOREM 2.1. *[15] Hamming–CSP is NP-Complete for a binary alphabet.*

Given a metric M and $d_M(p_i, \pi)$ the minimum number of operations regarding the metric M to transform p_i into π, the CLOSEST PERMUTATION PROBLEM is:

> METRIC M CLOSEST PERMUTATION PROBLEM (M–CPP)
> INPUT: a set of permutations $\{p_1, p_2, \ldots, p_k\}$ of length n and a non-negative integer d.
> QUESTION: is there a permutation π of length n such that $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$?

In case of a positive answer to M–CPP, we call a *solution for M–CPP* any per-

mutation π that satisfies $\max_{i=1,\ldots,k} d_M(p_i, \pi) \leq d$.

Given a set of permutations and a metric M, the aim of the M–Closest Permutation Problem is finding a permutation that minimizes the maximum distance between the solution and all other input permutations.

3. Non-reduced permutations. Christie [3] defined the reduced permutation $gl(\pi)$ of a given permutation π, which is obtained by "gluing" all the adjacencies of each *strip* of π together, where a pair $\pi_i\pi_{i+1}$ is an adjacency if $\pi_{i+1} = \pi_i + 1$, and a maximal sequence of adjacencies is called a strip. We discard the leftmost (resp. the rightmost) strip if it begins with 1 (resp. if it ends with n), then keep the minimal element of each strip in π, and finally renumber the remaining elements appropriately. For instance $gl([6\,7\,8\;4\,5\;1\,2\,3]) = [3\,2\,1]$.

Christie [3] proved that $d_t(\pi) = d_t(gl(\pi))$, but the reduction operation does not preserve the SSBM distance. Consider for instance $\pi = [2\,1] = gl(\sigma = [3\,4\,1\,2])$: $d_{sbm}(\pi) = 1$, while $d_{sbm}(\sigma) = 2$. It is clear that if only the first elements or the last elements are already sorted, then $d_{sbm}(\pi) = d_{sbm}(gl(\pi))$, for instance if $\pi = [1\,2\,3\,5\,8\,7\,6\,4]$, then $d_{sbm}(\pi) = d_{sbm}(gl(\pi))$, since the first 3 elements are already sorted, $gl(\pi) = [2\,5\,4\,3\,1]$.

Next, we consider permutations to the reduce reverse permutation and to an α-permutation, two well-studied classes of permutations in the context of SBT.

3.1. Non-reduced reverse permutation. Heath and Vergara [8] computed the exact short block-move distance for the *reverse permutation*, $\rho_{[n]} = [n\,n{-}1\,\cdots\,2\,1]$. A *non-reduced reverse permutation* is a permutation π such that $gl(\pi) = \rho_{[n]}$. If π is a non-reduced reverse permutation, then $PG(\pi)$ is a *k-partite complete graph*, where each *block of adjacencies* corresponds to an independent set, and each element in a block is adjacent to every element that belongs to a block to its right. See, for instance, the permutation $[8\,9\;4\,5\,6\,7\;1\,2\,3]$, whose permutation graph is illustrated in Figure 3.1.

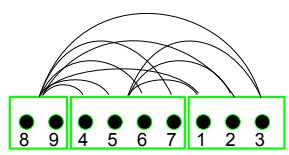

Fig. 3.1. *The permutation graph of* $[8\,9\;4\,5\,6\,7\;1\,2\,3]$. *There are arcs between all elements in a block of adjacencies on the left and all elements in a block of adjacencies on the right.*

Heath and Vergara [8] showed that the lower bound of Equation (2.3) is tight for permutations having *bipartite* permutation graphs, and the matched and unmatched vertices also correspond to a sorting sequence for those permutations. We show below a maximum matching and the unmatched vertices also yielding to an optimal sorting sequence for the non-reduced reverse permutations.

Theorem 3.1. *If $\pi_{[n]}$ is a non-reduced reverse permutation, then*

$$d_{sbm}(\pi_{[n]}) = |M| + |U|,$$

where M is a maximum matching and U is the set of unmatched vertices in $A^a_{\pi_{[n]}}$.

Proof. Let $\pi_{b_1,b_2,\cdots,b_m}$ be a non-reduced reverse permutation:

$$\pi = [\ \boxed{\quad B_m \quad}\ \boxed{\quad B_{m-1} \quad} \cdots \boxed{\quad B_1 \quad}\ ],$$

where each B_i is a block of adjacencies. We will show that the corresponding arc graph $A^{\pi_{b_1,b_2,\cdots,b_m}}$ of $\pi_{b_1,b_2,\cdots,b_m} = [b_{m,1}\, b_{m,2} \cdots b_{m,|B_m|} \cdots b_{2,1}\, b_{2,2} \cdots b_{2,|B_2|}\, b_{1,1}\, b_{1,2} \cdots b_{1,|B_1|}]$ contains a spanning subgraph that either has a perfect matching or only one unmatched vertex remains.

Let $G(B_x, B_y)$ be a grid graph with vertex set $\{b_{x,i}, b_{y,j}; i = 1, \ldots, |B_x|, j = 1, \ldots, |B_y|\}$. The *vertical edges* connect $b_{x,i}, b_{y,j}$ to $b_{x,i}, b_{y,j+1}$, for $i = 1, \ldots, |B_x|$ and $j = 1, \ldots, |B_y| - 1$, whereas the *horizontal edges* connect $b_{x,i}, b_{y,j}$ to $b_{x,i+1}, b_{y,j}$, for $i = 1, \ldots, |B_x| - 1$ and $j = 1, \ldots, |B_y|$ (Fig. 3.2).

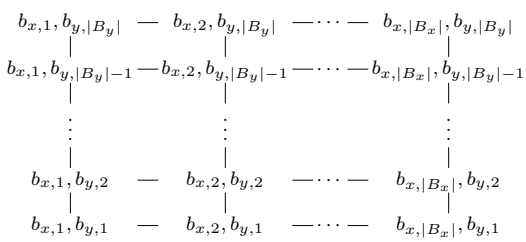

FIG. 3.2. *Grid graph $G(B_x, B_y)$.*

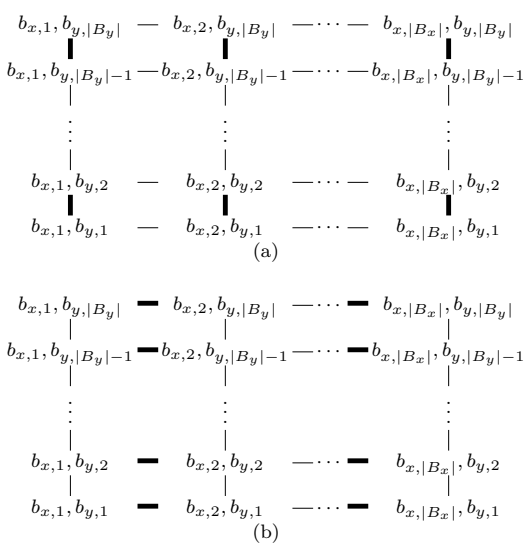

FIG. 3.3. *$G(B_x, B_y)$. Bold edges represent the matching when: (a) $|B_x|$ is even. (b) $|B_x|$ is odd and $|B_y|$ is even.*

Notice that the union of the graphs $G(B_x, B_y)$ and $G(B_{x+1}, B_y)$, added with the horizontal edge set $H(x, y)$ (Fig. 3.5a), where each edge in this set connects $b_{x,|B_x|}, b_{y,j}$ to $b_{x+1,|B_{x+1}|}, b_{y,j}$, for $j = 1, \ldots, |B_y|$, is also a subgraph of the arc graph. The same can be said of the union of the graphs $G(B_x, B_y)$ and $G(B_x, B_{y+1})$, along with the vertical edges $V(x, y)$ (Fig. 3.5b) connecting $b_{x,i}, b_{y,|B_y|}$ to $b_{x,i}, b_{y+1,1}$, for $i = 1, \ldots, |B_x|$.

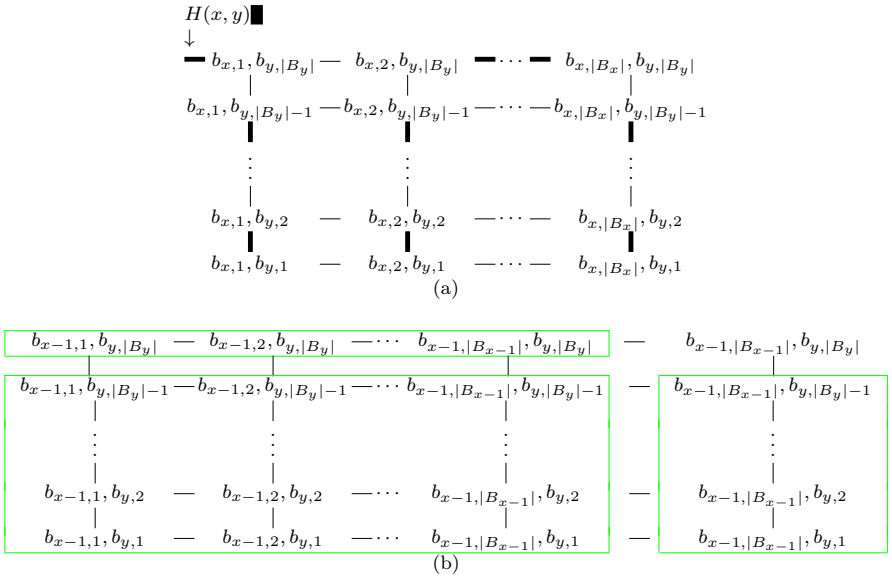

FIG. 3.4. *(a) Grid graph $G(B_x, B_y)$, for $|B_x|$ and $|B_y|$ are odd. (b) Grid graph $G(B_{x-1}, B_y)$ after the matching of (a).*

The union of all the graphs $G(B_x, B_y)$, for $y = 1, \ldots, m-1$ and $x = y+1, \ldots, m$, has the same vertex set as the arc graphs, which means that it is a spanning subgraph. If we add all the vertical edges $V(x, y)$ and horizontal edges $H(x, y)$, this graph is still a spanning subgraph (Fig. 3.6).

In Figure 3.3 we show a matching with respect to the parity of B_x and B_y, where each vertical edge corresponds to a hop $\underline{b_{x,i}}\ \underline{b_{y,j}b_{y,j+1}}$, whereas each horizontal edge corresponds to a hop $\underline{b_{x,i}b_{x,i+1}}\ \underline{b_{y,j}}$, except in Figure 3.3c the edge on the matching corresponding to the vertex $\overline{b_{x,1}, b_{y,|B_y|}}$, which corresponds to the hop $\underline{b_{x+1,|B_{x+1}|}b_{x,1}}\ \underline{b_{y,|B_y|}}$ of $H(x, y)$. Note that if there is an even number of vertices in $A^{\pi_{b_1,b_2,\ldots,b_m}}$, then there exists a perfect matching given by the edges showed in Figure 3.3. If there is an odd number of vertices on $A^{\pi_{b_1,b_2,\ldots,b_m}}$, then there is no perfect matching, but there is only one unmatched vertex, corresponding to a skip. □

3.2. α^k-permutation. Labarre [14] proposed the α-permutation in the context of the SBT. A permutation π of length n is an α-permutation if all even elements are in the correct positions and all odd elements form either an increasing or a decreasing cycle in the graph $\Gamma(\pi) = (V, E)$, which is the directed graph where $V = \{1, \ldots, n\}$, and $E = \{(i, \pi_i) | i = 1, \ldots, n\}$. In the following, we propose a class that generalizes the class of the α-permutations.

DEFINITION 3.2. *A permutation π of length n is an* α^k-permutation *if: each even element is already in its correct position; there are k odd elements in their correct positions; and the other odd elements form either an increasing or a decreasing cycle in $\Gamma(\pi)$.*

If $k = 0$, then both definitions of α-permutation and α^k-permutation coincide. An α^k-permutation, for $k > 0$, is a non-reduced α-permutation, but there are non-

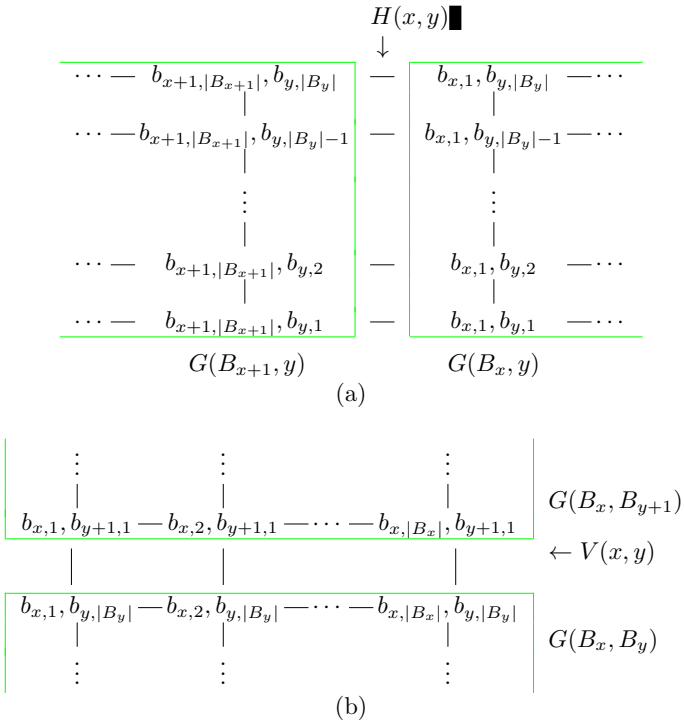

FIG. 3.5. *Unions of grid graphs.*

reduced α-permutations which are not α^k-permutations, for instance, $\pi = [3\ 2\ 5\ 6\ 4\ 1]$, which is a non-recuded of the α-permutation $gl(\pi) = [3\ 2\ 5\ 4\ 1]$, but π is not an α^k-permutation.

For a given n, there is an exponential number of α^k-permutations, that is to say $\sum_{i=0}^{\lceil \frac{n}{2} \rceil - 1} \binom{\lceil \frac{n}{2} \rceil}{i}$. In the following, we compute the short block-move distance of a α^k-permutation.

THEOREM 3.3. *For any α^k-permutation $\pi_{[n]}$, we have $d_{sbm}(\pi_{[n]}) = \left\lceil \frac{|E^p_{\pi_{[n]}}|}{2} \right\rceil$, where $|E^p_{\pi_{[n]}}|$ is the number of inversions of $\pi_{[n]}$.*

Proof. Let $\pi_{[n]}$ be α^k-permutation: $\pi = [\cdots e \overset{\cdots}{B_1}\ \overset{e}{n}\ \overset{\cdots}{B_2}\ \overset{n}{1}]$, where B_1 and B_2 are blocks of adjacencies and we call $|B_1| = \ell_1$, and $|B_2| = \ell_2$ for $\ell_1, \ell_2 \geq 1$. We distinguish four cases based on the parity of ℓ_1, and ℓ_2, and show how to sort this last part of π. We apply correcting hops and hence obtain an $\alpha^{n-\ell_1-\ell_2-2}_{k'}$-permutation.

- ℓ_1 is even, ℓ_2 is even: We can sort the last elements, as follows: $e\ B_1\ \underline{n}\ \underline{B_2}\ 1 \underset{\ell_2/2 \text{ hops}}{\Rightarrow} \underline{e}\ \underline{B_1}\ B_2\ n\ 1 \underset{\ell_1/2 \text{ hops}}{\Rightarrow} \underline{B_1\ e\ B_2\ n}\ \underline{1} \underset{(\ell_1+\ell_2+2)/2 \text{ hops}}{\Rightarrow} 1\ B_1\ e\ B_2\ n$.
- ℓ_1 is even, ℓ_2 is odd: $\underline{e}\ \underline{B_1}\ n\ B_2\ 1 \underset{\ell_1/2 \text{ hops}}{\Rightarrow} B_1\ e\ \underline{n}\ \underline{B_2\ 1} \underset{(\ell_2+1)/2 \text{ hops}}{\Rightarrow} \underline{B_1\ e\ B_2}\ \underline{1}\ n \underset{(\ell_1+\ell_2+1)/2 \text{ hops}}{\Rightarrow} 1\ B_1\ e\ B_2\ n$.

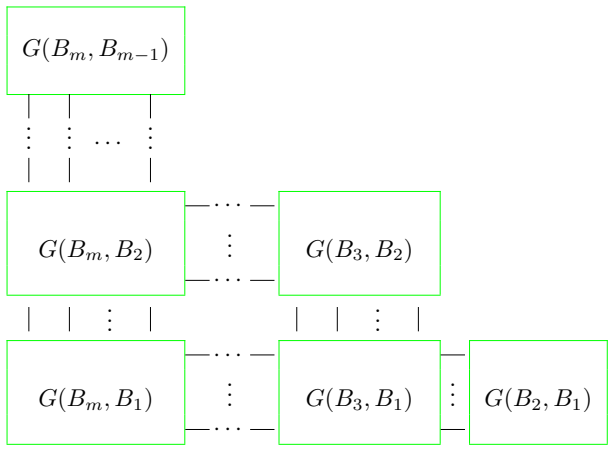

FIG. 3.6. *Spanning subgraph for* $A^{\pi_{b_1,b_2,\ldots,b_m}}$.

- ℓ_1 is odd, ℓ_2 is even: $e\ B_1\ \underline{n}\ \underline{B_2}\ 1 \underset{\ell_2/2 \text{ hops}}{\Rightarrow} e\ \underline{B_1\ B_2\ n}\ \underline{1} \underset{(\ell_1+\ell_2+1)/2 \text{ hops}}{\Rightarrow} \underline{e}\ \underline{1\ B_1}\ B_2\ n \underset{(\ell_1+1)/2 \text{ hops}}{\Rightarrow} 1\ e\ B_1\ B_2\ n$.
- ℓ_1 is odd, ℓ_2 is odd: $e\ B_1\ \underline{n}\ \underline{B_2\ 1} \underset{(\ell_2+1)/2 \text{ hops}}{\Rightarrow} e\ \underline{B_1\ B_2}\ \underline{1}\ n \underset{(\ell_1+\ell_2)/2 \text{ hops}}{\Rightarrow} \underline{e}\ \underline{1\ B_1}\ B_2\ n \underset{(\ell_1+1)/2 \text{ hops}}{\Rightarrow} 1\ e\ B_1\ B_2\ n$.

Hence, if π has an even number of $B_i's$, then we can sort π using hops alone, so we do the last strategy until the permutation is sorted, and we do not apply any skip. If π has an odd number of $B_i's$, then we apply the above strategy, except on the final, which we need one skip in the following case: $n\ 2\ 3\ \ldots\ n-1\ 1$. However, it is still tight with respect to the lower bound of Equation (2.2). □

4. Equivalence relations for block-move distances. Next, we consider two other well known transformations studied in the SBT problem: the toric equivalence, which does not preserve the short block-move distance, but we show it does preserve the p-bounded block-move distance for $p > (n+1)/2$; and the reverse complement operation, which we show it does preserve all bounded block-move distances.

4.1. Toric equivalence. A *circularization* of a permutation π is the circular permutation π° obtained from π by inserting an extra element 0 which acts both as predecessor of π_1 and as successor of π_n. The circularization of π is denoted $\pi^\circ = (0\ \pi_1 \cdots \pi_n)$. For a given circular permutation π°, its *q-step cyclic value shift* is the circular permutation $q + \pi^\circ = (\overline{q}\ \overline{q+\pi_1} \cdots \overline{q+\pi_n})$, where $\overline{x}$ is the remainder of the division of x by $n+1$. Finally, two permutations π, σ are *torically equivalent* if $\sigma^\circ \equiv q + \pi^\circ$ for some integer q. For instance, the permutation [3 4 1 2], is torically equivalent to permutations, [2 3 1 4], [1 3 4 2], [3 1 2 4] and [1 4 2 3].

Eriksson et al. [6] introduced the toric equivalence classes in the context of the SBT problem. If π and σ are torically equivalent, then $d_t(\pi) = d_t(\sigma)$. To prove that two torically equivalent permutations π and σ have the same transposition distance, one transposition $t(i,j,k)$ applied to π is transformed into one transposition $t(i',j',k')$ to be applied to σ, in such a way that i', j' and k' are computed as a function of the

circular shift that transforms π into σ [7].

Hausen [7] showed, by induction on the transposition distance, a transformation from a sorting sequence for a permutation π to a sorting sequence for any torically equivalent permutation σ. If $d_t(\pi) = 1$, then $\pi = \iota t(i, j, k)$ for some transposition, and the transformed transposition to apply $\sigma = \iota t(i', j', k')$ is obtained as a function of $t(i, j, k)$, m' and m, where m' is the position of the element $n + 1 - q$ in π, and m is the element such that $m' + m \equiv 0 \pmod{n + 1}$, shown in Table 4.1.

case	π	$\sigma = \iota t(i', j', k')$
1	$m' < i$	$\iota t(i - m', j - m', k - m')$
2	$i \leq m' < j$	$\iota t(j - m', k - m', i + m)$
3	$j \leq m' < k$	$\iota t(k - m', i + m, j + m)$
4	$k \leq m' \leq n$	$\iota t(i + m, j + m, k + m)$

TABLE 4.1

How to replace a transposition from a permutation π to its toric permutation σ [7].

If $d_t(\pi) > 1$, Hausen [7] gave a similar argument which can be used on transpositions $t(i', j', k')$.

Considering now the SSBM problem and the transformation given in Table 4.1, if $t(i, j, k)$ is a short block-move in Case 2), then $t(i', j', k')$ is not a short block-move. For instance, $\pi = [1\,2\,4\,3]$ and its torically equivalent $\sigma = [4\,1\,2\,3]$; then $d_{sbm}(\pi) = 1$, but $d_{sbm}(\sigma) = 2$.

Despite on the SSBM the torically equivalent permutations do not necessarily preserve the distance, for a p-bounded block-move problem we can measure the number of torically equivalent permutations that preserve the distance by the same argument made in Table 4.1.

PROPOSITION 4.1. *Given π and σ two torically equivalent permutations. If $p > \frac{n+1}{2}$, then $d_{pbbm}(\pi) = d_{pbbm}(\sigma)$, where d_{pbbm} is the p-bounded block-move distance.*

Proof. By considering Table 4.1, for cases 1) and 4): if $k - i \leq p$, then $k' - i' \leq p$; regarding case 2), the difference $k' - i'$ is: $k' - i' = i + m - j + m' = i + (n + 1 - m') - j + m' = i - j + n + 1$, which is less than p for $(n + 1) < 2p$, since $i - j \leq -p$; case 3) is similar to the case 2). □

4.2. Reverse complement of a permutation. The *reverse complement* of a permutation $\pi_{[n]}$ is the permutation $[n + 1 - \pi_n \;\; n + 1 - \pi_{n-1} \;\; \cdots \;\; n + 1 - \pi_1]$, or equivalently *conjugating* π by ρ, $\pi^\rho = \rho\pi\rho^{-1} = \rho\pi\rho$, where $\rho_{[n]} = [n \; n{-}1 \cdots 1]$ is the reverse permutation.

In the SBT problem, distinct classes can be merged implying permutations in distinct classes with equal distances. For instance, the toric and the reverse complement classes, by considering $\pi = [4\,2\,3\,1]$, and the permutation $\pi^\rho = [2\,4\,3\,1]$ which is not torically equivalent to π. All permutations torically equivalent to π have the same distances of all permutations torically equivalent to π^ρ.

The breakpoint graph is an equivalence relation that preserves the SBT distance. As illustrated in Figure 4.1, the reverse complement reflects the breakpoint graph. One can note that given a permutation π and its reverse complement π^ρ, permutation π has a torically equivalent permutation whose reverse complement permutation is torically equivalent to π^ρ. The breakpoint graph of a permutation torically equivalent permutation to π can be obtained by a circularization of $BG(\pi)$, next we prove that two reverse complement permutations have their breakpoint graphs reflected.

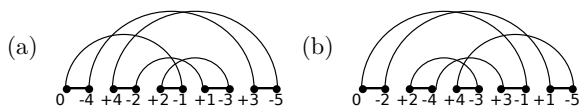

FIG. 4.1. *(a)* $BG([4\ 2\ 1\ 3])$*, (b)* $BG([2\ 4\ 3\ 1])$*, where* $[2\ 4\ 3\ 1] = [4\ 2\ 1\ 3]^\rho$*.*

THEOREM 4.2. *Given a permutation* π*, the breakpoint graph* $BG(\pi)$ *is exactly the reflected breakpoint graph of* $BG(\pi^\rho)$*, which implies* $d_t(\pi) = d_t(\pi^\rho)$*.*

Proof. By the difference of the adjacencies between the consecutive elements of π, we construct the reality edges of the breakpoint graph, the differences are: $\pi_2 - \pi_1,\ \pi_3 - \pi_2, \ldots,\ \pi_n - \pi_{n-1}$. Hence, considering the differences between the consecutive elements of π^ρ, we have: $n+1-\pi_{n-1}-(n+1-\pi_n) = \pi_n - \pi_{n-1},\ \ldots,\ \pi_3 - \pi_2,\ \pi_2 - \pi_1$, which are exactly the reflected reality edges of $BG(\pi)$. □

An analogous equality of Theorem 4.2 for the SSBM problem comes from a consequence of another property for the SBT problem with respect to the reverse complement.

LEMMA 4.3. *[14] For every transposition* $t(i,j,k)$ *of length* n*, we have:* $(t(i,j,k))^\rho = t(n-k+2, n-j+2, n-i+2)$*.*

COROLLARY 4.4. *For every* p*-bounded block-move* $t(i,j,k)$*,* $(t(i,j,k))^\rho$ *is also a* p*-bounded block-move.*

Proof. Follows from Theorem 4.3 and the fact that $n-i+2-n+k-2 = k-i \leq p$, by hypothesis. □

PROPOSITION 4.5. *For any permutation* π*, we have* $d_{pbbm}(\pi) = d_{pbbm}(\pi^\rho)$*.*

Proof. If $\pi = t_q t_{q-1} \cdots t_1$, then:

$$\rho\pi\rho = \rho t_q t_{q-1} \cdots t_1 \rho = \underbrace{\rho t_q \rho}_{t'_q} \underbrace{\rho t_{q-1} \rho}_{t'_{q-1}} \rho \cdots \rho \underbrace{\rho t_1 \rho}_{t'_1},$$

where t'_i is a short block-move for $1 \leq i \leq q$ (see Theorem 4.3). Therefore, we have $d_{pbbm}(\pi) \geq d_{pbbm}(\pi^\rho) \geq d_{pbbm}((\pi^\rho)^\rho) = d_{pbbm}(\pi)$. □

It is natural to wonder whether conjugating π by permutations other than ρ yields different permutatios with the same distance. We show below that only ι and ρ enjoy this property in the case of the short block-move distance.

PROPOSITION 4.6. *Let* $p \geq 2 \in \mathbb{N}$ *and* $\sigma \in S_n$ *be such that for all* $\pi \in S_n$*. If* $d_{pbbm}(\pi) = d_{pbbm}(\pi^\sigma)$*, then* $\sigma \in \{\rho, \iota\}$*.*

Proof. Conjungating by ι trivially preserves the distance, and we know that conjugating by ρ preserves the p-bounded block-move distance (Theorem 4.5). We now show that for all other values of σ, there exists a p-bounded block-move t such that t^σ is not a p-bounded block-move.

If $\sigma \notin \{\rho, \iota\}$, then $n \geq 3$ and σ contains a subpermutation isomorphic to $(1\ 3\ 2)$, $(2\ 3\ 1), (2\ 1\ 3)$ or $(3\ 1\ 2)$, i.e. a sequence of three consecutive elements with relative order following as those triples. Let us assume this subpermutation starts at position i, and let us conjugate $t^\sigma = \sigma \cdot t \cdot \sigma^{-1}$. We consider each case:

- $\sigma = \begin{smallmatrix} \cdots & i & i{+}1 & i{+}2 & \cdots \\ [\cdots & 1 & 3 & 2 & \cdots] \end{smallmatrix}$. Hence, in σ^{-1}, the elements $i, i+1, i+2$ are in positions represented by elements $1, 3, 2$, respectively, which is $\sigma^{-1} = \begin{smallmatrix} \cdots & 1 & \cdots & 2 & \cdots & 3 & \cdots \\ [\cdots & i & \cdots & i{+}2 & \cdots & i{+}1 & \cdots] \end{smallmatrix}$;

we consider the adjacent transposition $t = t(i, i{+}1, i{+}2) = \begin{smallmatrix} 1 & 2 & \cdots & i{-}1 & i & i{+}1 & i{+}2 & \cdots n \\ [1 & 2 & \cdots & i{-}1 & i{+}1 & i & i{+}2 & \cdots n] \end{smallmatrix}$.
Now, in $t^\sigma = \sigma \cdot t \cdot \sigma^{-1}$, the elements in positions represented by $1, 2, 3$ are:
$\sigma \cdot t \cdot \sigma^{-1}(1) = \sigma(t(\sigma^{-1}(1))) = \sigma(t(i)) = \sigma(i+1) = 3$;
$\sigma \cdot t \cdot \sigma^{-1}(2) = \sigma(t(\sigma^{-1}(2))) = \sigma(t(i+2)) = \sigma(i+2) = 2$;
$\sigma \cdot t \cdot \sigma^{-1}(3) = \sigma(t(\sigma^{-1}(3))) = \sigma(t(i+1)) = \sigma(i) = 1$.

- $\sigma = \begin{smallmatrix} \cdots & i & i{+}1 & i{+}2 & \cdots \\ [\cdots & 3 & 1 & 2 & \cdots] \end{smallmatrix}$. Hence, in σ^{-1}, the elements $i, i+1, i+2$ are in positions represented by elements $3, 1, 2$, respectively, which is $\sigma^{-1} = \begin{smallmatrix} \cdots & 1 & \cdots & 2 & \cdots & 3 & \cdots \\ [\cdots & i{+}1 & \cdots & i{+}2 & \cdots & i & \cdots] \end{smallmatrix}$;
we consider the adjacent transposition $t = t(i, i{+}1, i{+}2) = \begin{smallmatrix} 1 & 2 & \cdots & i{-}1 & i & i{+}1 & i{+}2 & \cdots n \\ [1 & 2 & \cdots & i{-}1 & i{+}1 & i & i{+}2 & \cdots n] \end{smallmatrix}$.
Now, in $t^\sigma = \sigma \cdot t \cdot \sigma^{-1}$, the elements in positions represented by $1, 2, 3$ are:
$\sigma \cdot t \cdot \sigma^{-1}(1) = \sigma(t(\sigma^{-1}(1))) = \sigma(t(i+1)) = \sigma(i) = 3$;
$\sigma \cdot t \cdot \sigma^{-1}(2) = \sigma(t(\sigma^{-1}(2))) = \sigma(t(i+2)) = \sigma(i+2) = 2$;
$\sigma \cdot t \cdot \sigma^{-1}(3) = \sigma(t(\sigma^{-1}(3))) = \sigma(t(i)) = \sigma(i+1) = 1$.
- $\sigma = \begin{smallmatrix} \cdots & i & i{+}1 & i{+}2 & \cdots \\ [\cdots & 2 & 3 & 1 & \cdots] \end{smallmatrix}$. Hence, in σ^{-1}, the elements $i, i+1, i+2$ are in positions represented by elements $2, 3, 1$, respectively, which is $\sigma^{-1} = \begin{smallmatrix} \cdots & 1 & \cdots & 2 & \cdots & 3 & \cdots \\ [\cdots & i{+}2 & \cdots & i & \cdots & i{+}1 & \cdots] \end{smallmatrix}$;
we consider the adjacent transposition $t = t(i{+}1, i{+}2, i{+}3) = \begin{smallmatrix} 1 & 2 & \cdots & i & i{+}1 & i{+}2 & i{+}3 & \cdots n \\ [1 & 2 & \cdots & i & i{+}2 & i{+}1 & i{+}3 & \cdots n] \end{smallmatrix}$.
Now, in $t^\sigma = \sigma \cdot t \cdot \sigma^{-1}$, the elements in positions represented by $1, 2, 3$ are:
$\sigma \cdot t \cdot \sigma^{-1}(1) = \sigma(t(\sigma^{-1}(1))) = \sigma(t(i+2)) = \sigma(i+1) = 3$;
$\sigma \cdot t \cdot \sigma^{-1}(2) = \sigma(t(\sigma^{-1}(2))) = \sigma(t(i)) = \sigma(i) = 2$;
$\sigma \cdot t \cdot \sigma^{-1}(3) = \sigma(t(\sigma^{-1}(3))) = \sigma(t(i+1)) = \sigma(i+2) = 1$.
- $\sigma = \begin{smallmatrix} \cdots & i & i{+}1 & i{+}2 & \cdots \\ [\cdots & 2 & 1 & 3 & \cdots] \end{smallmatrix}$. Hence, in σ^{-1}, the elements $i, i+1, i+2$ are in positions represented by elements $2, 1, 3$, respectively, which is $\sigma^{-1} = \begin{smallmatrix} \cdots & 1 & \cdots & 2 & \cdots & 3 & \cdots \\ [\cdots & i{+}1 & \cdots & i & \cdots & i{+}2 & \cdots] \end{smallmatrix}$;
we consider the adjacent transposition $t = t(i{+}1, i{+}2, i{+}3) = \begin{smallmatrix} 1 & 2 & \cdots & i & i{+}1 & i{+}2 & i{+}3 & \cdots n \\ [1 & 2 & \cdots & i & i{+}2 & i{+}1 & i{+}3 & \cdots n] \end{smallmatrix}$.
Now, in $t^\sigma = \sigma \cdot t \cdot \sigma^{-1}$, the elements in positions represented by $1, 2, 3$ are:
$\sigma \cdot t \cdot \sigma^{-1}(1) = \sigma(t(\sigma^{-1}(1))) = \sigma(t(i+1)) = \sigma(i+2) = 3$;
$\sigma \cdot t \cdot \sigma^{-1}(2) = \sigma(t(\sigma^{-1}(2))) = \sigma(t(i)) = \sigma(i) = 2$;
$\sigma \cdot t \cdot \sigma^{-1}(3) = \sigma(t(\sigma^{-1}(3))) = \sigma(t(i+2)) = \sigma(i+1) = 1$.

So, the triple $(3\ 2\ 1)$ in t^σ is not represented in a block-move. Therefore, whenever $\sigma \notin \{\rho, \iota\}$, we can always find a 2-bounded block-move whose conjugate by σ is not a p-bounded block-move, which proves the contrapositive of our claim. □

In Theorem 4.1 we showed that two torically equivalent permutations have the same p-bounded block move distance, for $p > \frac{n+1}{2}$, and by Theorem 4.5 and Theorem 4.3 we have that any two reverse complement permutations have the same p-bounded block-move distance. Since two reverse complement permutations in general are not torically equivalent, then toric equivalence classes are merged by the reverse complement in the p-bounded block move distance.

5. Short block-move distance of unions of permutations. Although the SBT and SSBM problems are closely related, the short block-move and transposition distances of a permutation can differ by an arbitrarily large amount. For instance, for the reverse permutation $\rho_{[n]}$, it is known that $d_t(\rho_{[n]}) = \left\lfloor \frac{n}{2} \right\rfloor + 1$ (see [16]), but $d_{sbm}(\rho_{[n]}) = \left\lceil \frac{n(n-1)}{4} \right\rceil$ (see [9]). Rest, determining families of permutations for which both distances are equal is a necessarily problem.

In Section 5.1 we characterize some families of permutations with equal transposition and short block-move distances. In Section 5.2 we prove that an optimal sequence of short block-moves to sort any permutation can be obtained by sorting each connected component of the corresponding permutation graph separately.

5.1. Distance of unions of small permutations. Given permutations $\pi_{[n]}$ and $\sigma_{[m]}$, the *union* $\pi_{[n]} \uplus \sigma_{[m]}$ is the permutation $\gamma_{[n+m+1]} = \pi_{[n]} \uplus \sigma_{[m]} = [\pi_1 \ldots \pi_n \ (n+1) \ (\sigma_1 + n+1) \ldots \ (\sigma_m + n+1)]$.

A union $\pi_{[n]} \uplus \sigma_{[m]} \uplus \delta_{[\ell]} \uplus \ldots$, the operations are succeeding performed from the left to the right, and we say that the permutations $\pi_{[n]}$, $\sigma_{[m]}$, $\delta_{[\ell]}$, ... are the *parts* of this union. Some properties of unions of permutations are in [5].

A permutation $\pi_{[n]}$ is *small* if $n \leq 3$. It is easy to verify that the SSBM distance of every small permutation $\pi_{[n]}$ is equal to the lower bound $\left\lceil \frac{|E^p_{\pi_{[n]}}|}{2} \right\rceil$. The shortest sequence to sort any union $\pi_{[n]} \uplus \sigma_{[m]} \uplus \delta_{[\ell]} \uplus \ldots$, either by short block-moves or by general transpositions, can be obtained by sorting each part separately, since the distance of each part equals the lower bound on the short block-move distance of Equation (2.2) and also on the transposition distance of Equation (2.1). This yields Theorem 5.1.

PROPOSITION 5.1. *Let $\pi_{[n]}$, $\sigma_{[m]}$, $\delta_{[\ell]}$, ... be small permutations and $\gamma = \pi_{[n]} \uplus \sigma_{[m]} \uplus \delta_{[\ell]} \uplus \ldots$. Hence $d_t(\gamma) = d_t(\pi_{[n]}) + d_t(\sigma_{[m]}) + d_t(\delta_{[\ell]}) + \ldots = d_{sbm}(\pi_{[n]}) + d_{sbm}(\sigma_{[m]}) + d_{sbm}(\delta_{[\ell]}) + \ldots = d_{sbm}(\gamma)$.*

Proof. Given a permutation $\gamma = \pi_{[n]} \uplus \sigma_{[m]} \uplus \delta_{[\ell]} \uplus \cdots$ constructed by unions of small permutations, we have that $c_{odd}(\gamma) = c_{odd}(\pi_{[n]}) + c_{odd}(\sigma_{[m]}) + c_{odd}(\delta_{[\ell]}) + \cdots$ and $|E^p_{\pi_{[n]} \uplus \sigma_{[m]}}| = |E^p_{\pi_{[n]}}| + |E^p_{\sigma_{[m]}}| + |E^p_{\delta_{[m]}}| + \cdots$, and for any small permutation $\pi_{[n]}$, it holds that $\left\lceil \frac{(n+1) - c_{odd}(\pi_{[n]})}{2} \right\rceil = d_t(\pi_{[n]}) = d_{sbm}(\pi_{[n]}) = \left\lceil \frac{|E^p_{\pi_{[n]}}|}{2} \right\rceil$. □

We can construct permutations with $n > 3$ elements via unions of small permutations, since each part has distance equal to the lower bound of Equation (2.1) and such transpositions are also short block-moves that also achieve the lower bound of Equation (2.2), yielding an infinite family of permutations for which both the SSBM distance, and also the SBT distance, can be determined in polynomial time.

For the SBT, sorting each part of the union separately just gives us an upper bound for the transposition distance [5], and gives exactly the transposition distance if each part can be sorted with the lower bound of Equation (2.1), as showed in [5]. However, for SSBM we prove next that sorting each part of the union separately always is optimal.

5.2. Connected components of permutations graphs. Let us refer block-moves that introduce elements in connected components of the permutation as *merging moves*. For instance, $[2\ 3\ \underline{1\ 6}\ \underline{4}\ 5] \rightarrow [2\ 3\ 4\ 1\ 6\ 5]$ is a merging move.

LEMMA 5.2. *For every permutation π, sorting each connected component of π separately is optimal.*

Proof. We allow ourselves to use merging moves, which can be replaced by correcting moves as in Table 2.1. The modified sequence is not longer than the original, and we observe that these new moves never merge components.

A merging move must act on contiguous components of π. Let us assume that the leftmost component the move acts on ends with elements a and b, and that the rightmost component starts with elements c and d, as represented below:

$$\boxed{\quad a \quad b} \ \boxed{c \quad d \quad\quad}$$

It implies that $a < c, a < d, b < c$ and $b < d$. We now replace any merging move

involving those component's extremities with correcting moves. There are five cases to consider:

- $\underline{a}\ \underline{b\ c}\ d \to b\ c\ a\ d$: this move satisfies the conditions of case 2 in Table 2.1, so we replace it with $\underline{a}\ \underline{b}\ c\ d \to b\ a\ c\ d$.
- $\underline{a\ b}\ \underline{c}\ d \to c\ a\ b\ d$: this move satisfies the conditions of case 4 in Table 2.1, so we replace it with $\underline{a}\ \underline{b}\ c\ d \to b\ a\ c\ d$.
- $a\ \underline{b}\ \underline{c}\ d \to a\ c\ b\ d$: this move satisfies the conditions of case 1 in Table 2.1, and in this case we just remove that block-move from the sorting sequence.
- $a\ \underline{b}\ \underline{c\ d} \to a\ c\ d\ b$: this move satisfies the conditions of case 5 in Table 2.1, so we replace it with $a\ b\ \underline{c}\ \underline{d} \to a\ b\ d\ c$.
- $a\ \underline{b\ c}\ \underline{d} \to a\ d\ b\ c$: this move satisfies the conditions of case 3 in Table 2.1, so we replace it with $a\ b\ \underline{c}\ \underline{d} \to a\ b\ d\ c$.

None of the correcting moves that we use to replace the non-correcting moves in those five cases is a merging move, and no such replacement increases the length of our sorting sequence. Given any sorting sequence, we repeatedly apply the above transformation to the merging move with the smallest index until no such move remains; in particular, the transformation applies to optimal sequences as well, and the proof is complete.□

Note that there exist cases where allowing merging moves still yields an optimal solution. This is the case for [2 1 4 3], which can be sorted to optimally as follows: $[2\ \underline{1\ 4}\ \underline{3}] \to [\underline{2\ 3}\ \underline{1}\ 4] \to \iota$.

It is natural to wonder whether Theorem 5.2 generalizes to p-bounded block-move, for $p > 3$. However, the following counterexample shows that it is not the case, even for a bound of 4: sorting each component of [3 2 1 6 5 4] separately yields a sequence of length four, but one can do better by merging components as follows: $[3\ 2\ \underline{1\ 6}\ \underline{5\ 4}] \to [3\ \underline{2\ 5}\ \underline{4\ 1}\ 6] \to [\underline{3\ 4}\ \underline{1\ 2}\ 5\ 6] \to \iota$.

6. Short block-move–CPP **is NP-complete.** Despite the fact that the computational complexity of SSBM is open, we can investigate a more general problem than the sorting one, which is the closest problem. In Section 5.2 we showed we can sort each connected component of the permutation graph separately, which is not the case for other greater bounded block-moves distances. From Theorem 5.2, we can show SSBM–CPP is NP-complete.

Firstly, we apply Algorithm 1 that transforms an arbitrary binary string s of length m into a particular permutation λ_s of length $2m$.

Algorithm 1: $Permut_{BI}(s)$

input : Binary string s of length m
output: Permutation λ_s

1 each occurrence of 0 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i-1$ and $2i$, respectively.
2 each occurrence of 1 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i$ and $2i-1$, respectively.

Next, we establish the key equality between the Hamming distance of an input string s and the short block-move distance of its output permutation λ_s obtained from Algorithm 1.

Lemma 6.1. *Given a string of length m and a permutation λ_s of length $2m$*

obtained in Algorithm 1, the short block-move distance of λ_s is $d_{sbm}(\lambda_s) = d_H(s)$.

Proof. From Theorem 5.2, each connected component can be sorted separately, and each bit set to 1 corresponds an inversion. □

Now, we show how a solution for the H–CSP implies in a solution for the Short block-move–CPP, and vice versa.

LEMMA 6.2. *Given a set of k permutations obtained by Algorithm 1, there is a short block-move closest permutation with max distance at most d if and only if there is a Hamming closest string with max distance equal to d.*

Proof. ($\Rightarrow$) "from permutation to string". If λ' can be built by Algorithm 1 for some input string s', then, by Lemma 6.1, s' is a closest string.

Otherwise, we search from the left to the right the permutation to find the first position where the corresponding element is different from the one intended to be by the algorithm, which is a position $x \in \{2i-1, 2i\}$. In this case, all elements from position x until the position where the first element $y \in \{2i-1, 2i\}$ appears form inversions with respect to each input permutation, implying the short block-move distance between the solution $[A\,x\,B\,y\,C]$ and any input greater than the distance between the new permutation $[A\,y\,x\,B\,C]$ and any input permutation, such that A, B and C are blocks of elements.

By repeating this process, a string agreeing with the algorithm output can be found and, by Lemma 6.1, a string with maximum distance equal to d can be constructed.

($\Leftarrow$) "from string to permutation". Given a solution string s, we obtain the associated permutation λ_s given by Algorithm 1. By Lemma 6.1 we have the solution s regarding the H–CSP corresponding to the permutation λ_s with the same value of max distance d. □

THEOREM 6.3. SHORT BLOCK-MOVE–CPP *is NP-complete.*

7. Conclusions. Although sorting permutations by short block-moves is still an open problem, some advances have been proposed and some questions arise.

In this paper we obtain properties regarding equivalence classes of permutations, for the p-bounded block-move we can merge two distinct toric equivalence classes by two reverse complement permutations, which implies the same p-bounded block-move distance for all those permutations. For the short block-move distance we have that instead regarding the toric equivalence the distance is not preserved, for the reverse complement the distance is preserved. So, an interesting investigation can be the search for equivalence relations that preserve the short block-move distance, where we can merge distinct equivalence relations by the same.

We also show that with respect to the short block-move distance we can sort each connected component of the permutation graph separately, but this property does not hold for other bounded block-moves. Hence, a natural question is: What can we say about permutations that are 1-edge connected? Is it possible to obtain an optimal sorting sequence of short block-moves by separating the permutation into two connected components and then by sorting each connected component separately?

We prove from the Hamming–CSP that the Short block-move–CPP is NP-complete, which is also with respect to other classes of input permutations. An example is by transforming each generic binary string s into a permutation by associating each bit 0 to the identity permutation, and each bit 1 to any permutation π such that we know

its short block-move distance. In this case, we generalize the result in Lemma 6.1 to the equality $d_{sbm}(\lambda_s) = d_{sbm}(\pi)d_H(s)$.

Since we show that the Short block-move–CPP is NP-complete, the following question arises: is the p-Bounded block-move–CPP NP-complete? As we discussed in Section 5.2, Theorem 5.2 does not hold for p-bounded block-moves, for $p \geq 4$.

REFERENCES

[1] Vineet Bafna and Pavel A Pevzner. Sorting by transpositions. *SIAM J. Discrete Math.*, 11:224–240, 1998.

[2] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Sorting by transpositions is difficult. *SIAM J. Discrete Math.*, 26:1148–1180, 2012.

[3] David Alan Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1998.

[4] L. F. I. Cunha, V. F. dos Santos, L. A. B Kowada, and de Figueiredo C. M. The block-interchange and the breakpoint closest permutation problems are NP-complete. In *Proceedings of 18th Latin-Iberoamerican Conference on Operations Research*, pages 1–8, 2016.

[5] Luís Felipe I Cunha, Luis Antonio B Kowada, Rodrigo de A. Hausen, and Celina MH de Figueiredo. Advancing the transposition distance and diameter through lonely permutations. *SIAM J. Discrete Math.*, 27:1682–1709, 2013.

[6] Henrik Eriksson, Kimmo Eriksson, Johan Karlander, Lars Svensson, and Johan Wästlund. Sorting a bridge hand. *Discrete Math.*, 241:289–300, 2001.

[7] R. A. Hausen. *Rearranjos de Genomas: Teoria e Aplicações*. PhD thesis, University of Rio de Janeiro, 2007.

[8] Lenwood S Heath and John Paul C Vergara. Sorting by bounded block-moves. *Discrete Appl. Math.*, 88:181–206, 1998.

[9] Lenwood S Heath and John Paul C Vergara. Sorting by short block-moves. *Algorithmica*, 28:323–352, 2000.

[10] H. Jiang, H. Feng, and D Zhu. An 5/4-approximation algorithm for sorting permutations by short block moves. In *Proceedings of 25th International Symposium on Algorithms and Computation*, volume 8889, pages 491–503, 2014.

[11] H. Jiang, D. Zhu, and Zhu B. $(1+\epsilon)$-approximation algorithm for sorting by short block-moves. *Theor. Comput. Sci.*, 439:1–8, 2012.

[12] D Knuth. The art of computer programming: Sorting and searching, volume 3, 1998.

[13] Anthony Labarre. New bounds and tractable instances for the transposition distance. *IEEE/ACM Trans. on Comput. Biol. Bioinformatics*, 3:380–394, 2006.

[14] Anthony Labarre. *Combinatorial Aspects of Genome Rearrangements and Haplotype Networks*. PhD thesis, Universite Libre de Bruxelles, 2008.

[15] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Inform. and Comput.*, 185:41–55, 2003.

[16] João Meidanis, MEMT Walter, and Z Dias. Transposition distance between a permutation and its reverse. In *Proceedings of the 4th South American Workshop on String Processing*, pages 70–79, 1997.

[17] V. Y. Popov. Multiple genome rearrangement by swaps and by element duplications. *Theor. Comput. Sci.*, 385:115–126, 2007.

Apêndice D

Anexo: Manuscrito "On sorting permutations by restricted multi-break rearrangements"

ON SORTING PERMUTATIONS BY RESTRICTED MULTI-BREAK REARRANGEMENTS*

LUÍS FELIPE I. CUNHA†, LUIS ANTONIO B. KOWADA‡, RODRIGO DE A. HAUSEN§, ANTHONY LABARRE¶, LAURENT BULTEAU¶, AND CELINA M. H. DE FIGUEIREDO†

Abstract. A genome rearrangement problem for permutations consists in finding a shortest sequence of operations that sorts a given permutation with the constraint that the allowed operation is fixed in advance. The corresponding distance is the length of such a sequence. We focus on *restricted multi-breaks*, which generalize well-studied rearrangement operations such as block-interchanges, reversals, and transpositions, by selecting at most k *non-reversible blocks* and reconnecting the fragments according to a fixed order.

We study two variants of sorting by restricted multi-breaks: one where k is fixed, and one where k is arbitrary. Regarding the problem where k is fixed, we prove lower bounds on the corresponding distances, and we focus on $k = 1$. We characterize permutations that are tight with respect to a lower bound, exhibit upper bounds and permutations whose distances achieve the equality of such bounds, we give upper bounds of $\frac{3n}{4}$ and $\frac{2n}{3}$, and we determine a lower bound on the 1ϖ diameter, i.e. a lower bound on the maximal value that the distance can reach. For the latter case where k is arbitrary, we determine polylogarithmic bounds on the ϖ diameter.

Key words. Genome rearrangements, Restricted multi-break, Tractable classes of permutations, Approximation algorithms

1. Introduction. Genome rearrangement problems ask for the minimum number of evolutionary events required to transform a genome into another one. Those problems are classical in bioinformatics [16]. From a mathematical point of view, we represent genomes as permutations of positive integers under some well defined hypothesis, and aim at finding the minimum number of operations to transform a given permutation into the identity permutation, where the allowed operations are fixed beforehand. The length of such a sequence of operations is the corresponding *rearrangement distance*.

Many choices exist for the rearrangement operations, they vary according to applications and assumptions, and yield problems of varying complexity. A recent trend is to study more general operations, which started with the *double cut-and-join* operation [21], a generalization of *transpositions* [5], *reversals* [4], and *translocations* [6]. The distance problems are NP-hard for transpositions [9] and reversals [11], but polynomial for double cut-and-join [21] and translocations [6]. Other variants were subsequently proposed, including a weighted version [8] and the *single cut or join* operation [15], and efficient algorithms exist for these variations. Alekseyev and Pevzner [3] proposed a generalization of the double cut-and-join operation, called *multi-breaks*. Much less is known about that version, and only an FPT algorithm exists.

In order to better understand multi-breaks, we propose two new rearrangement models: *restricted multi-break rearrangements*, and *k restricted multi-break rearrangements*. Both of them are special cases of multi-breaks, but remain general enough to encompass several well-known models of rearrangements, such as: reversals, block-interchanges, where the distance problem is polynomial [12] but such operation generalizes the NP-hard transposition problem. Restricted forms of transpositions and

*Extended abstract appeared in [].

†Universidade Federal do Rio de Janeiro, Brazil {lfignacio,celina}@cos.ufrj.br

‡Universidade Federal Fluminense, Brazil, luis@vm.uff.br

§Universidade Federal do ABC, Brazil, hausen@compscinet.org

¶Université Paris-Est Marne-la-Vallée, France, anthony.labarre@univ-mlv.fr,l.bulteau@gmail.com

reversals were studied before and results regarding complexity and approximation algorithms are known [10, 13, 14, 17, 18].

A *restricted multi-break* is an operation that reverses an interval in a permutation, except possibly for a certain number of elements within that interval. The sub-intervals that are not reversed are called *non-reversible blocks*. A *k-restricted multi-break* is a restricted multi-break such that the reversed interval contains at most k non-reversible blocks. We shall use ϖ to denote restricted multi-breaks, and $k\varpi$ to denote k-restricted multi-breaks. On the $k\varpi$, we prove lower bounds on the corresponding distances. On the 1ϖ, we prove the exact 1ϖ distance for a class of permutations, a characterization of permutations whose 1ϖ distances are equal to the lower bound on the number of breakpoints, we propose upper bounds of $\frac{3n}{4}$ and $\frac{2n}{3}$, and we also determine a lower bound on the diameter of $n/2$ of such 1ϖ problem. Regarding the diameter problem on the ϖ distance, we show polylogarithmic lower and upper bounds.

This paper is organized as follows: Section 2 presents the basic background on $k\varpi$ and ϖ, the relationship between the corresponding distances and other well-known rearrangements, and bounds on the $k\varpi$ and ϖ distances; Section 3 is devoted to the 1ϖ distance, where we characterize tight permutations, propose an upper bound on the distances based on *permutation cycles*, exhibit a class of permutations that achieve that bound, and conclude the section with upper bounds of $\frac{3n}{4}$ and $\frac{2n}{3}$; Section 4 is devoted to the classical problem of computing the diameter, we determine a lower bound on the 1ϖ diameter, and we present a $\Omega(\log n)$ lower bound and a $O(\log^2 n)$ upper bound on the ϖ diameter; Section 5 discusses the perspectives and identifies open questions that we intend to pursue.

2. Sorting by ϖ and $k\varpi$. For our purposes, a gene is represented by a unique integer and a chromosome with n genes is a *permutation* $\pi = [\boldsymbol{\pi_0}\, \pi_1\, \pi_2\, \cdots\, \pi_n \boldsymbol{\pi_{n+1}}]$, where $\boldsymbol{\pi_0} = 0$ and $\boldsymbol{\pi_{n+1}} = n+1$. The remaining n elements form a bijection from the set $\{1, 2, \ldots, n\}$ onto itself, and the operations we consider will never act on $\boldsymbol{\pi_0}$ nor $\boldsymbol{\pi_{n+1}}$. We denote π_i^{-1} the position of element π_i.

Definition 2.1 *The* restricted multi-break $\varpi(a, b; c_1 \leftrightarrow d_1; c_2 \leftrightarrow d_2; \ldots; c_r \leftrightarrow d_r)$, *where* $1 \leq a \leq c_1 \leq d_1 \leq c_2 \leq d_2 \leq \ldots \leq c_r \leq d_r \leq b \leq n$, over π, *reverses the interval of* π *defined by positions* a *and* b, *except for the* non-reversible blocks *defined by the pairs* (c_i, d_i) *for* $1 \leq i \leq r$, *thereby transforming* π *into the permutation* $\pi\varpi$:

$$[\boldsymbol{\pi_0}\pi_1 \cdots \pi_{a-1} \boxed{\pi_b \cdots \pi_{d_r+1} \underbrace{\boxed{\pi_{c_r} \cdots \pi_{d_r}} \pi_{c_r-1} \cdots \pi_{d_1+1} \boxed{\pi_{c_1} \cdots \pi_{d_1}}}_{r \text{ non-reversible blocks}} \pi_{c_1-1} \cdots \pi_a} \pi_{b+1} \cdots \pi_n \boldsymbol{\pi_{n+1}}],$$

Note that a restricted multi-break can be seen as one reversal from π_a to π_b followed by reversals in all non-reversible blocks from π_{c_i} to π_{d_i}, for $i = 1, \cdots, r$, and $r = 0$ corresponds to a reversal. A sequence of m restricted multi-breaks *sorts* a permutation π (or *is a sorting sequence* for π) if $\pi\, \varpi_1\, \varpi_2 \cdots\, \varpi_m = \iota$, where each ϖ_j is a restricted multi-break and $\iota = [\mathbf{0}\, 1\, 2 \cdots n\ \boldsymbol{n+1}]$ is the *identity permutation*. The ϖ operation may feature an arbitrary number $r \geq 0$ of non-reversible blocks, and we shall also consider the $k\varpi$ operation, where r is at most a fixed k. The ϖ *distance* of π, denoted by $d_\varpi(\pi)$, is the length of a shortest sorting sequence of restricted multi-breaks for π. The $k\varpi$ distance, denoted by $d_{k\varpi}(\pi)$, is the length of a shortest sorting sequence of restricted multi-breaks for π where each restricted multi-break has at most k non-reversible blocks.

Restricted multi-breaks are a restricted form of the *multi-break* operation, an operation recently defined by Alekseyev and Pevzner [3], where an ℓ-break cuts the permutation into ℓ places and then rearranges those corresponding blocks in any arbitrary order, different from a $k\varpi$ that applies a fixed order of such blocks. By definition of an ℓ-break, we have that $\ell + 1 \leq k \leq 2\ell$.

Restricted multi-breaks generalize well-known rearrangements, as a transposition $\tau(i, j, k)$, a short block-move $\gamma(i, j, i+2)$ or $\gamma(i, j, i+3)$, a block-interchange $\beta(i, j, k, t)$ and a reversal $\rho(i, j)$, for $1 \leq i < j < k < t \leq n + 1$, such that:

- *transposition* [5], swapping of two contiguous blocks of elements, $\tau(a, d+1, b+1) = \varpi(a, b; a \leftrightarrow d; d+1 \leftrightarrow b)$;
- *short block-move* [17], transposition such that the sum of two blocks is at most 3, $\gamma(a, b, a+2) = \varpi(a, a+1)$, or $\gamma(a, b, a+3) = \varpi(a, a+2; a \leftrightarrow b-1; b \leftrightarrow a+2)$;
- *block-interchange* [12], swapping of two non-necessarily contiguous blocks of elements, $\beta(a, d_1, c_2+1, b) = \varpi(a, b; a \leftrightarrow d_1; d_1+1 \leftrightarrow c_2; c_2+1 \leftrightarrow b)$; and
- *reversal* [4], reversing of an interval, $\rho(a, b) = \varpi(a, b)$ correspond to $r = 0$.

By Definition 2.1, every $(k-1)\varpi$ operation is a $k\varpi$ operation, implying the inequality $d_{k\varpi}(\pi) \leq d_{(k-1)\varpi}(\pi)$ for a given permutation π. Moreover, each operation which contains exactly k non-reversible blocks can be mimicked by two $(k-1)\varpi$ operations.

PROPOSITION 2.2 *For every permutation π, $d_{k\varpi}(\pi) \leq d_{(k-1)\varpi}(\pi) \leq 2d_{k\varpi}(\pi)$.*

Proof. We show how to transform any sorting sequence of $k\varpi$ operations for π into a sorting sequence of $(k-1)\varpi$ operations for π. Note that only operations that contain exactly k non-reversible blocks need to be considered.

To lighten notation, we partition π into blocks denoted by upper-case letters, i.e. $\pi = [A\ B\ C\ D\ E\ \ldots F\ G\ N]$, and use the notation $\overline{X}$ to denote block X after it has been reversed. Let us consider a $k\varpi$ operation that reverses all blocks between B and G included, preserving k non-reversible blocks. That operation's action is illustrated below:

$$[A\ \underline{B\ \underline{C}\ D\ \underline{E} \ldots \underline{F}\ G}\ N] \Rightarrow [A\ \overline{G}\ F\ \ldots E\ \overline{D}\ C\ \overline{B}\ N].$$

The resulting permutation can also be obtained by applying two $(k-1)\varpi$ operations on π: $[A\ B\ C\ \underline{D\ \underline{E} \ldots \underline{F}\ G}\ N] \Rightarrow [A\ \underline{B\ \underline{C}\ \overline{G}\ F\ \ldots E\ \overline{D}}\ N] \Rightarrow [A\ \overline{G}\ F\ \ldots E\ \overline{D}\ C\ \overline{B}\ N]$. □

2.1. Lower bound based on breakpoints. Natural bounds on distance problems were obtained using the *breakpoints* of a permutation. An *adjacency* (resp. a *reverse adjacency*) in a permutation π is a pair (π_i, π_{i+1}) for $0 \leq i \leq n$ such that $\pi_{i+1} = \pi_i + 1$ (resp. $\pi_{i+1} = \pi_i - 1$). If it is neither an adjacency nor a reverse adjacency, i.e. $|\pi_i - \pi_{i+1}| \neq 1$, then (π_i, π_{i+1}) is called a *breakpoint*, and we denote $b(\pi)$ the number of breakpoints of π.

An *increasing strip* of π is a maximal block of consecutive adjacencies. The *reduced permutation* $gl(\pi)$ is the permutation obtained from π by removing the first increasing strip if it starts with 1, replacing all other increasing strips with their smallest element, and relabeling the s remaining elements in the resulting permutation π' so as to obtain a permutation of $\{1, 2, \ldots, s\}$ whose elements are ordered in the same way as in π' [12].

For instance, if $\alpha = [\mathbf{0} \bullet 6\ 5\ 4 \bullet 1\ 2\ 3 \bullet 10\ 9\ 8\ 7 \bullet \mathbf{11}]$, then $gl(\alpha) = [\mathbf{0} \bullet 4\ 3\ 2\ 1 \bullet 8\ 7\ 6\ 5 \bullet \mathbf{9}]$, where $\bullet$ indicates a breakpoint. A straightforward consequence of the definition is that $b(gl(\pi)) \leq b(\pi)$. Christie [12] showed that the reduction operation preserves the block-interchange and the transposition distances. The same property holds for the general ϖ distance.

PROPOSITION 2.3 *For every permutation π, we have $d_\varpi(\pi) = d_\varpi(gl(\pi))$.*

Proof. Every increasing strip can be considered as a non-reversible block in a ϖ operation, therefore $d_\varpi(\pi) \leq d_\varpi(gl(\pi))$, i.e. any ϖ operation in $gl(\pi)$ can be mimicked by a ϖ operation in π. On the other hand, to show that $d_\varpi(\pi) \geq d_\varpi(gl(\pi))$, a sorting sequence of π can be used to sort $gl(\pi)$, since if a ϖ operation affects elements in π not belonging to a strip, then the same operation can be applied in $gl(\pi)$, and if a ϖ affects elements belonging to a strip, nothing is done to sort $gl(\pi)$. □

The equality on the distances of π and $gl(\pi)$ also holds for the block-interchange and for the transposition distance problems [12], but it does not hold for restricted forms of transpositions, such as short block-move [17] nor prefix transposition [18]. Moreover, for any fixed k, the equality does not hold either, e.g. for the reversal distance, $d_{0\varpi}([\mathbf{0}\,2\,3\,1\,\mathbf{4}]) = 2$, whereas $d_{0\varpi}([\mathbf{0}\,2\,1\,\mathbf{3}]) = 1$. This does not contradict Proposition 2.3, since the ϖ distance, as opposed to the $k\varpi$ distance, can feature any number of non-reversible blocks at each step. Nevertheless, we obtain the following lower bound on the $k\varpi$ distance:

THEOREM 2.4 *For every permutation π, we have* $d_{k\varpi}(\pi) \geq \left\lceil \frac{b(\pi)}{2(k+1)} \right\rceil$.

Proof. By applying a ϖ over π with k non-reversible blocks, the number of breakpoints of $\pi \cdot \varpi$ is $b(\pi \cdot \varpi) \geq b(\pi) - (2 + 2k)$, since the best case of a $\varpi(a, b; c_1 \leftrightarrow d_1; c_2 \leftrightarrow d_2; \ldots; c_k \leftrightarrow d_k)$ we are able to eliminate a pair of breakpoints in the external interval $(\pi_{a-1}\pi_a)$, $(\pi_b\pi_{b+1})$ and each pair of breakpoints of a non-reversible block $(\pi_{c_i-1}\pi_{c_i})$, $(\pi_{d_i}\pi_{d_i+1})$ for $i = 1, \ldots, k$. Thereby we can conclude the proof by induction on the number of breakpoints of π.

$$\begin{array}{ll}
b(\pi \cdot \varpi_1) & \geq b(\pi) - (2 + 2k), \\
b(\pi \cdot \varpi_1\varpi_2) & \geq b(\pi\varpi_1) - (2 + 2k), \\
b(\pi \cdot \varpi_1\varpi_2\varpi_3) & \geq b(\pi\varpi_1\varpi_2) - (2 + 2k), \\
\ldots & \\
b(\pi \cdot \varpi_1\varpi_2\varpi_3 \ldots \varpi_d) & \geq b(\pi\varpi_1\varpi_2\varpi_3 \ldots \varpi_{n-1}) - (2 + 2k).
\end{array}$$

Since $b(\pi \cdot \varpi_1\varpi_2\varpi_3 \ldots \varpi_d) = 0$, where d is the ϖ distance of π and summing such inequalities, we have that $0 \geq b(\pi) - d(\pi).(2 + 2k)$, then $d(\pi) \geq \left\lceil \frac{b(\pi)}{2(k+1)} \right\rceil$. □

Note that Theorem 2.4 generalizes the known lower bound of $\frac{b(\pi)}{2}$ for reversals [19], but for transpositions the corresponding lower bound is $\frac{b(\pi)}{3}$ [5].

2.2. Lower bound based on the breakpoint graph. Several bounds on distance problems related to reversals, transpositions and block-interchanges were proved using variants of the *breakpoint graph*. As we shall see, that graph will also prove useful in the context of the $k\varpi$ distance (Theorem 2.6 below and Section 4.1).

The *breakpoint graph* [4] of a permutation $\pi = [\mathbf{0}\,\pi_1 \cdots \pi_n\,\boldsymbol{n+1}]$ is the graph $G(\pi) = (V, E)$ with vertex set $V = \{0, 1, \ldots, n, n{+}1\}$ and with edge set E partitioned into: *black edges*, which connect pairs of vertices that correspond to breakpoints in π; and *gray edges*, which connect pairs of vertices that correspond to elements that are consecutive in ι but not in π.

An *alternating cycle* in a breakpoint graph is a cycle whose edges use black and gray edges in an alternating way. There are several ways to decompose a breakpoint graph into edge-disjoint alternating cycles. We use the notation $c(\pi)$ to denote the cardinality of a *maximum cycle decomposition* of $G(\pi)$, i.e. a decomposition that contains the largest number of alternating cycles. Figure 2.1 illustrates $G([\mathbf{0}\,3\,1\,5\,7\,4\,2\,6\,\mathbf{8}])$ and a corresponding maximum cycle decomposition.

Breakpoint graphs were defined by Bafna and Pevzner [4] in the context of the reversal distance; they used it to derive the non-trivial following bounds. Given

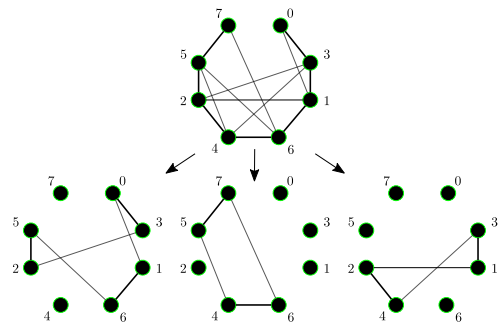

Fig. 2.1: $G([\mathbf{0}\,3\,1\,6\,4\,2\,5\,\mathbf{7}])$, and a maximum cycle decomposition into three cycles.

a permutation π and a reversal ρ over π, we denote $\Delta c(\pi\rho) = c(\pi\rho) - c(\pi)$, and $\Delta b(\pi\rho) = b(\pi\rho) - b(\pi)$. Bafna and Pevzner [4] proved that given any permutation π and any reversal ρ, we have $\Delta c(\pi\rho) - \Delta b(\pi\rho) \leq 1$ and that the reversal distance is at least $b(\pi) - c(\pi)$.

Since a $k\varpi$ operation can be viewed as at most $k+1$ reversals, corresponding to exactly $k+1$ reversals when there are k non-reversible blocks in such $k\varpi$, we can also achieve a lower bound on the $k\varpi$ distance using the *breakpoint graph*, similar to the one obtained with respect to the reversal distance.

LEMMA 2.5 *For every permutation π and any $k\varpi$ operation, $\Delta c(\pi\varpi) - \Delta b(\pi\varpi) \leq k+1$.*

Proof. The effect of a $k\varpi$ having k non-reversible blocks is the same as $k+1$ reversals, therefore a reversal applied $k+1$ times proves the result of one $k\varpi$ operation. □

THEOREM 2.6 *For every permutation π, $d_{k\varpi}(\pi) \geq \left\lceil \frac{b(\pi)-c(\pi)}{k+1} \right\rceil$.*

Proof. Considering $\varpi_t, \ldots, \varpi_1$ as a shortest sorting sequence that transforms $\pi = \pi^{(t)}$ into ι, let us denote $\pi^{(i-1)} = \pi^{(i)}\varpi_i$, for $i = 1, \ldots, t$. By Lemma 2.5, after applying a ϖ_i: $d_{k\varpi}(\pi_i) = d_{k\varpi}(\pi^{(i-1)}) + 1 \geq d_{k\varpi}(\pi^{(i-1)}) - \Delta b(\pi^{(i)}\varpi_i) + \Delta c(\pi^{(i)}\varpi_i) - k$. Therefore,

$$\begin{array}{ll}
d_{k\varpi}(\pi^{(i)}) - (b(\pi^{(i)}) - c(\pi^{(i)})) & \geq \\
d_{k\varpi}(\pi^{(i-1)}) - (b(\pi^{(i-1)}) - c(\pi^{(i-1)})) - k & \geq \\
d_{k\varpi}(\pi^{(i-2)}) - (b(\pi^{(i-2)}) - c(\pi^{(i-2)})) - 2k & \geq \\
\ldots & \geq \\
d_{k\varpi}(\pi^{(0)}) - (b(\pi^{(0)}) - c(\pi^{(0)})) - ik = -ik, &
\end{array}$$

since $d_{k\varpi}(\pi^{(0)}) = b(\pi^{(0)}) = c(\pi^{(0)}) = 0$. Substituting $i = t$, and knowing that $d_{k\varpi}(\pi^{(t)}) = t$, we conclude that: $d_{k\varpi}(\pi) - (b(\pi) - c(\pi)) \geq -d_{k\varpi}(\pi)k$, hence $d_{k\varpi} \geq \left\lceil \frac{b(\pi)-c(\pi)}{k+1} \right\rceil$. □

3. Sorting by 1ϖ.

3.1. Tight permutations for 1ϖ distance. Regarding the lower bound given by the number of breakpoints, Bulteau *et al.* [9] proved that deciding whether the transposition distance is equal to $b(\pi)/3$ is NP-complete.

On the other hand, Christie [12] and Tran [20] characterized, independently, the class of *reversal-tight permutations*, i.e. permutations whose reversal distances are equal to $b(\pi)/2$, and gave a polynomial-time algorithm for their recognition. Since a reversal is a 0ϖ, it is natural to ask for a characterization of *$k\varpi$-tight* permutations, i.e. those that are tight with respect to the lower bound of Theorem 2.4. In the sequel, we characterize the class of 1ϖ-tight permutations.

Since π is 1ϖ-tight if and only if $d_{1\varpi}(\pi) = \frac{b(\pi)}{4}$, it means that in an optimal sorting sequence, each 1ϖ must remove exactly 4 breakpoints. The following notion will be useful.

Definition 3.1 *[20] Two breakpoints (π_{i-1}, π_i) and (π_j, π_{j+1}) yield a* 2-active *reversal $\rho(i,j)$, if a) $|\pi_{i-1} - \pi_j| = 1$ and $|\pi_i - \pi_{j+1}| = 1$; whereas they yield a* 2-passive *reversal $\rho(i,j)$, if b) if $|\pi_{i-1} - \pi_{j+1}| = 1$ and $|\pi_i - \pi_j| = 1$.*

There is a reversal that removes exactly two breakpoints if the permutation contains a 2-active reversal. A 2-passive reversal can be transformed into a 2-active reversal iff there exists an interval of a 2-active reversal that overlaps exactly one of the two breakpoints that yield the 2-passive reversal. Our goal is to remove 4 breakpoints in each operation, hence we must consider combinations of 2-active and 2-passive reversals. The following graph B_π will be helpful in detecting them. Given a permutation π, let $B_\pi = (V, E)$ be the graph defined by:

- $V = \{\pi_i \mid (\pi_i, \pi_{i+1})$ is a breakpoint in $\pi\}$, and
- $E = \{uv \mid u, v$ yield a 2-active or a 2-passive reversal$\}$.

Figure 3.2(a) illustrates B_π for the permutation $\pi = [\mathbf{0}\,5\,2\,7\,4\,1\,6\,3\,\mathbf{8}]$. Given $B_\pi = (V, E)$, we define the *active-passive graph* of B_π, $AP(\pi) = (V', E')$, where $V' = V \cup E$ and $E' = E \cup E_1 \cup E_2$, where:

- $E_1 = \{\pi_i\pi_j\pi_i$ and $\pi_i\pi_j\pi_j | \pi_i\pi_j \in E\}$, and
- $E_2 = \{\pi_i\pi_j\pi_p\pi_q | \pi_i\pi_j$ and $\pi_p\pi_q$ satisfy conditions of the next Definition 3.2$\}$.

Definition 3.2 *Given a permutation π and its corresponding graph B_π, there is an edge ij, where $i = u_1v_1$ and $j = u_2v_2$, of E_2 in $AP(\pi)$ if a pair of vertices satisfies the following constraints:*

1. *i and j both yield 2-active reversals, and $\pi^{-1}_{u_1} < \pi^{-1}_{u_2} < \pi^{-1}_{v_2} < \pi^{-1}_{v_1}$. We say i* intersects totally *j; or*
2. *i yields a 2-active reversal and j yields a 2-passive reversal, and $\pi^{-1}_{u_1} < \pi^{-1}_{u_2} < \pi^{-1}_{v_1} < \pi^{-1}_{v_2}$. We say i* intersects partially *j; or*
3. *i and j yield both a 2-passive reversal, and $\pi^{-1}_{u_1} < \pi^{-1}_{u_2} < \pi^{-1}_{v_1} < \pi^{-1}_{v_2}$; or*
4. *i yields a 2-active reversal and j yields a 2-passive reversal, and $\pi^{-1}_{u_1} < \pi^{-1}_{u_2} < \pi^{-1}_{v_2} < \pi^{-1}_{v_1}$.*

Figure 3.1 illustrates such cases.

Figure 3.2(b) illustrates $AP(\pi)$ for the permutation $\pi = [\mathbf{0}\,5\,2\,7\,4\,1\,6\,3\,\mathbf{8}]$.

Note that some possible 1ϖ operations to be applied in π are represented by pairs of edges of E_2 in $AP(\pi)$ (for instance the edge 0154, since one extremity intersects totally another one), and some 1ϖ operations may affect the order of some pairs of 2-active and 2-passive intervals, turning them into pairs of 2-passive and 2-active reversals, respectively.

We say a 1ϖ operation is *good* if such a 1ϖ eliminates exactly 4 breakpoints transforming the permutation into a permutation with 4 less breakpoints.

After we have obtained $AP(\pi)$, we must consider pairs of vertices of E in $AP(\pi)$ satisfying the constraints of either there is a good 1ϖ (eliminating 4 breakpoints), or pairs that can be transformed in such a good 1ϖ after some other 1ϖ. These constraints are considered in Definition 3.2.

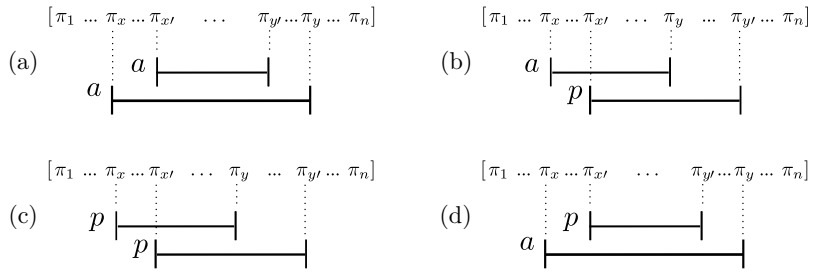

Fig. 3.1: Cases of intervals of 2-active and 2-passive reversals to be considered in the construction of E_2 in $AP(\pi)$. Labels a and p represent intervals of 2-active and 2-passive reversals, respectively.

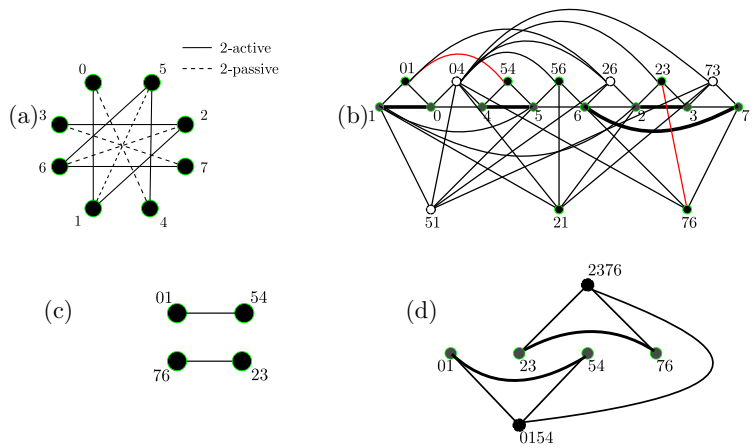

Fig. 3.2: Steps of a characterization to decide whether $\pi = [\mathbf{0}\,5\,2\,7\,4\,1\,6\,3\,\mathbf{8}]$ is 1ϖ tight: (a) B_π represents all 2-active and 2-passive pairs, (b) $AP(\pi)$ represents all combinations of 2-active and 2-passive pairs, (c) B'_π obtained from the perfect matching M of Theorem 3.3, (d) $AP'(B'_\pi)$.

An edge in E_2 is associated to a 1ϖ if its endpoints are in case of Definition 3.2(1) or Definition 3.2(4). A necessary condition for a permutation to be 1ϖ tight is the following.

THEOREM 3.3 *If a permutation π is 1ϖ tight, then the subgraph induced by the vertices of V in $AP(\pi)$ has a perfect matching.*

Proof. If there is not a perfect matching, then an unmatched vertex yields a reversal (0ϖ operation), where cannot remove 4 breakpoints and therefore π is not 1ϖ tight. □

Assume the subgraph induced by the vertices of V in $AP(\pi)$ has a perfect matching M, note that such a matching is exactly a matching in B_π. Nevertheless, $AP(\pi)$ is useful for the next step that is to obtain the graph B'_π, the subgraph of $AP(\pi)$ induced by the vertices in E corresponding to the edges of M. Figure 3.2(c) illustrates

the resulting graph B'_π.

Note that Theorem 3.3 does not give us a sufficient condition for a permutation to be 1ϖ tight, since an edge in a matching can be obtained by two 2-passive reversals, for instance as Definition 3.2(3), where it is not possible to be a good 1ϖ operation.

Now, we proceed by building, similar to $AP(\pi)$, the graph $AP'(B'_\pi) = (M \cup F, F \cup D \cup D')$, where:

- $F = \{abcd \mid abcd$ is an edge of $B`_\pi\}$,
- $D = \{abcdab$ and $abcdcd \mid abcd \in F\}$,
- $D' = \{abcdefgh \mid abcd \in F$ and $efgh \in F$, and $abcd$ and $efgh$ satisfy the constraints of next Definition 3.4$\}$.

Figure 3.2(d) illustrates the resulting graph $AP'(B'_\pi)$.

Definition 3.4 *Given a permutation π, if there is a perfect matching in $E(AP(\pi))$, let D' be the edges of $AP'(B'_\pi)$ such that an edge $pq \in D'$, if:*

- *p is of the form of Definition 3.2(1), q of the form of Definition 3.2(4), only one 2-active of p intersects only partially the 2-passive reversals of q, and no interval of p intersect the 2-active reversals of q; or*
- *p is of the form of Definition 3.2(1), q of the form of Definition 3.2(2), and only one 2-active reversals of p intersects totally the 2-active reversals of q, and partially the 2-passive reversals of q; or*
- *p and q are of the form of Definition 3.2(1); or*
- *p is of the form of Definition 3.2(1), q of the form of Definition 3.2(4), the 2-active reversals of q intersects totally both 2-active reversals of p, and the one of the 2-active reversals p intersect totally the 2-passive reversals of q, and the other one 2-active reversals of p intersect partially such 2-passive reversals of q; or*
- *p is of the form of Definition 3.2(1), q of the form of Definition 3.2(3), one of the 2-active reversals of p intersects totally both of 2-passive reversals of q, and the other 2-active reversals of p intersects partially both of 2-passive reversals of q.*

Figure 3.3 illustrates such cases.

Next, we proceed by searching for a perfect matching M' in the subgraph induced by the vertices M of $AP'(B'_\pi)$, and if there exists, we take the subgraph induced by the vertices $abcd$, where ab and cd belong to such matching M'.

A characterization for a permutation to be 1ϖ tight is the following.

Theorem 3.5 *A permutation π is 1ϖ tight if and only if all three conditions are satisfied: a) there is a perfect matching M in V of $AP(\pi)$; b) there is a perfect matching M' in M of $AP'(B'_\pi)$; and c) each edge on the subgraph induced by F follows one of the conditions of Definition 3.4.*

Proof. If a) b) and c) are satisfied, then each edge of F is corresponding to a sequence of two good 1ϖ, eliminating 4 breakpoints at each step.

Conversely, if the permutation is 1ϖ tight then each operation cover exactly 4 breakpoints. Hence with such 1ϖ's we obtain a perfect matching of B'_π with overlapping intervals of Figure 3.3 being satisfied, otherwise in such sorting sequence some 1ϖ would affect the order of another 1ϖ in such a way transforming into a not good operation. □

Note that the number of matchings can be exponential in the graphs considered in Theorem 3.5. In order to solve that, we obtain the best matching by setting weights $+1$ for edges that yield good 1ϖ operations and -1 otherwise. This argument is similar the one proposed by Tran for the reversal tight permutations [20].

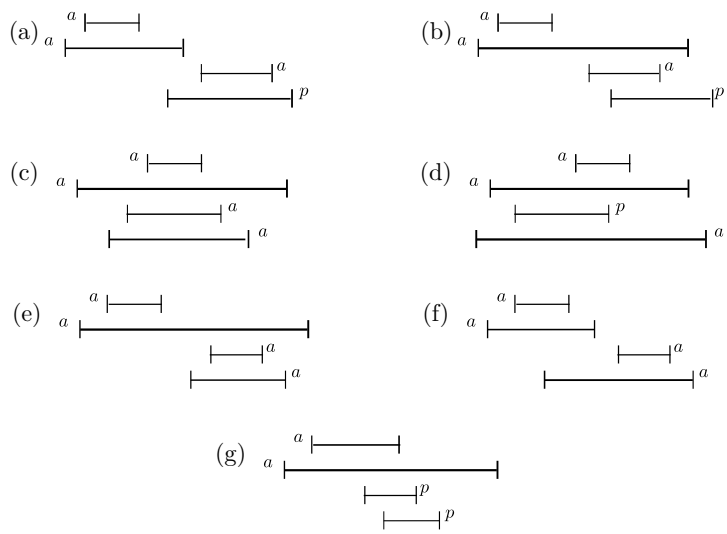

Fig. 3.3: Overlap cases.

3.2. Upper bounds for 1ϖ distance. A straightforward 4-approximation algorithm for computing the 1ϖ distance follows from the observation that a 1ϖ operation includes a reversal, which is a 0ϖ operation. Since there always exists a reversal that removes at least one breakpoint from an unsorted permutation [19], Theorem 2.4 allows us to conclude that this strategy has an approximation of ratio 4. Berman *et al.* [7] proved the best approximation ratio for sorting by reversals by proposing a 1.375-approximation algorithm, so far, implying then a 2.75-approximation for 1ϖ. We propose strategies to deal with 1ϖ operations different of reversals. In Section 3.2.1 we achieve an upper bound based on the permutation cycles of a permutation, in Section 3.2.3 we prove an upper bound of $2n/3$, implying in a 2.67-approximation, better than the previous 2.75 based on reversals.

A well-known distance between permutations is the *Cayley distance*, which is the minimum number of element exchanges that transform a given permutation into another given one.

Permutations can be represented with each element followed by its image. For instance, with $\{1, 2, 3\}$, the permutation $(1\,2\,3)$ maps 1 onto 2, 2 onto 3, and 3 onto 1, corresponding to the permutation $[\mathbf{0}\,2\,3\,1\,\mathbf{4}]$. This representation is not unique; $(2\,3\,1)$ and $(3\,1\,2)$ are equivalent. Permutations are composed of one or more permutation cycles. For instance, $\pi = [\mathbf{0}\,8\,5\,1\,3\,2\,7\,6\,4\,\mathbf{9}] = (1\,8\,4\,3)(2\,5)(6\,7)$ has three permutation cycles. We use $pc(\pi)$ to denote the number of permutation cycles of π (in this example, $pc(\pi) = 3$).

3.2.1. An upper bound based on permutation cycles. The following relationship between the Cayley distance and permutation cycles is well-known.

THEOREM 3.6 *[16] Given a permutation π of length n, the Cayley distance is $d_c(\pi) = n - pc(\pi)$.*

An exchange of a pair of elements is also a 1ϖ operation, therefore we have the

following bounds for the 1ϖ distance.

LEMMA 3.7 *Given a permutation π of length n, the 1ϖ distance of π satisfies:* $\frac{b(\pi)}{4} \leq d_{1\varpi}(\pi) \leq n - pc(\pi)$.

Proof. An exchange of a pair of elements can be viewed as $[A\, \underline{a\, \underline{B}\, b}\, C] \Rightarrow [A\, b\, B\, a\, C]$, where A, B, C are blocks of elements. Therefore, we have $d_{1\varpi}(\pi) \leq d_c(\pi)$. The lower bound is a direct consequence of Theorem 2.4. □

Indeed, Lemma 3.7 yields an approximation algorithm with a ratio that depends on the number of permutation cycles and on the breakpoints of a given permutation. For instance, in case the permutation of length n has $n/2$ permutations cycles, i.e. each cycle has two elements, and having n breakpoints, we have a 2-approximation on the 1ϖ distance.

3.2.2. Exact 1ϖ distance for star involution permutations. In Theorem 2.4, we have shown a lower bound on the $k\varpi$ distance and Theorem 3.5 characterizes the 1ϖ tight permutations. Next, we exhibit a class of permutations whose 1ϖ distance achieves the lower bound of Theorem 2.4. An *involution permutation* is a permutation such that all permutation cycles have length at most 2. We call a permutation a *star involution* if: each odd element in position $2i+1$, for $1 \leq i \leq \lfloor n/2 \rfloor$, is in correct position $\pi_{2i+1} = 2i+1$; and each other element forms permutations cycle of length 2, $(a\ a')$ such that $a' > a+2$. For instance $[\mathbf{0}\,9\,6\,3\,8\,5\,2\,7\,4\,1\,\mathbf{10}]$ is a star involution.

THEOREM 3.8 *Given a star involution permutation π, the 1ϖ distance is $d_{1\varpi}(\pi) = \frac{b(\pi)}{4}$.*

Proof. Given a star involution permutation π, each odd element is in correct position, except the first and the last (in case the permutation has odd length), and every cycle of length 2 $(a\ a')$ implies that the element a is in position a', and a' is in position more than 2 positions of a (since $a' > a+2$), so π has $n+1$ breakpoints. By Lemma 3.7 we have that $\frac{n}{4} \leq d_{1\varpi}(\pi) \leq n - pc(\pi)$, and by the construction of π we have $n - pc(\pi) = \lceil \frac{n}{4} \rceil$. □

3.2.3. A 2.67-approximation algorithm for 1ϖ distance. Although the upper bound given in Lemma 3.7 is tight for star involutions, for specific instances this strategy can yield a sorting sequence whose length is very far from the exact distance. Next, we present an upper bound on the 1ϖ distance.

LEMMA 3.9 *Given a permutation π, if there is an interval defined by a pair of breakpoints $(a, x), (a{+}1, y)$ and inside such interval there are at least two other breakpoints, then there exist two 1ϖ operations ϖ_1, ϖ_2 such that $\pi\varpi_1\varpi_2$ has at most $b(\pi) - 3$ breakpoints.*

Proof. Let us take a pair of breakpoints (a, b) and $(a+1, c)$, and let us call A the interval between elements a and $a+1$. Now, let us take another breakpoint (x, y) inside the interval A.

1. If either the breakpoint $(x+1, c')$ or $(x-1, d')$ is also inside A, then we can remove two breakpoints in one operation by applying 1ϖ affecting a and x.
2. If both such breakpoints $(x+1, c')$ and $(x-1, d')$ are outside A, let us say in the right of A, then we choose one of $(x+1, c')$ and $(x-1, d')$, let us say we have chosen $(x+1, c')$. Now we take another breakpoint (z, t) inside A, and if $(z+1, c'')$ or $(z-1, d'')$ is inside A then we remove 2 breakpoints applying one 1ϖ affecting a and z. If both $(z+1, c'')$ are $(z-1, d'')$ outside A, then we choose one of $(z+1, c'')$ and $(z-1, d'')$, let us say we have chosen $(z+1, c'')$, and we consider two subcases:

- the breakpoint $(z+1, c'')$ is between $(a+1, c)$ and $(x+1, c')$. Therefore, the first operation is $\varpi(z^{-1}+1, (x+1)^{-1}+1; (z+1)^{-1} \leftrightarrow (x+1)^{-1}+1)$, which removes the breakpoint affecting z. After that note that the breakpoints (x, y) and $(x+1, c')$ are inside the interval A between (a, b) and $(a+1, c)$. Hence, next we can remove two breakpoints similar as done in Item 1.
- the breakpoint $(x+1, c')$ is between $(a+1, c)$ and $(z+1, c'')$. Therefore, the first operation is $\varpi(z^{-1}+1, (z+1)^{-1}; z^{-1}+1 \leftrightarrow (a+1)^{-1}+1)$, which removes the breakpoint affecting z. After that the breakpoints (x, y) and $(x+1, c')$ are inside the interval A between (a, b) and $(a+1, c)$. Hence, next we can remove two breakpoints similar as done in Item 1.

After those cases we have removed at least 3 breakpoints in two 1ϖ operations. □

Lemma 3.9 gives us a strategy where in the worst case we can remove 3 breakpoints using two 1ϖ operations, except the case where between all pairs of breakpoints (a, b) and $(a+1, c)$ there is no other breakpoint (for instance the permutation [2 1 4 3 6 5 8 7]) or when there is just one other breakpoint (for instance the permutation [5 1 6 2 7 3 8 4]). We analyze those cases below.

LEMMA 3.10 *The 1ϖ distance of $\pi = [k\ k{-}1 \cdots 1\ 2k\ 2k{-}1 \cdots k{+}1 \cdots n\ n-1 \cdots]$ is $\frac{2n+k}{4k} \leq d_{1\varpi}(\pi) \leq \frac{n}{k}$.*

Proof. By Theorem 2.4 we have the lower bound of $\frac{n+k}{4k}$, which would be achieved if each operation could remove exactly 4 breakpoints. However, by the construction of π we see that there is no 1ϖ operation on π that removes 4 breakpoints, and any 1ϖ affects at most four intervals of decreasing strips. So, for each four decreasing strips we need at least one more 1ϖ. Hence, we have $d_{1\varpi}(\pi) \geq \frac{n+k}{4k} + \frac{n}{4k} = \frac{2n+k}{4k}$. The upper bound $\frac{n}{k}$ is achieved by performing a reversal in each decreasing strip. □

LEMMA 3.11 *For any permutation π, if between all pairs of breakpoints $(a\ x), (a{+}1\ y)$ there is one other breakpoint, then $\frac{b(\pi)}{4} \leq d_{1\varpi}(\pi) \leq \frac{b(\pi)}{2}$.*

Proof. The lower bound is straightforward from Theorem 2.4. A sequence of $\frac{b(\pi)}{2}$ 1ϖ can be performed by in each step i the ith. element is correctly putted in its position i. Note that after such operation, the elements before and after i forms now an adjacency. Therefore, we have created two new adjacencies in each step. □

Lemmas 3.10 and 3.11 yield 2-approximations for those classes of permutations. Therefore, in general we have a 2.67-approximation which follows.

THEOREM 3.12 *Lemma 3.9 yields a 2.67-approximation algorithm for sorting permutations by 1ϖ.*

Proof. Lemma 3.9 states that we can remove at least 3 breakpoints in two steps for all permutations except those described in Lemmas 3.10 and 3.11, which can be sorted by a 2-approximation, therefore the ratio of $\frac{8}{3} = 2.67$ follows from the lower bound of Theorem 2.4. □

4. The 1ϖ and ϖ diameters.

4.1. The 1ϖ diameter. The $k\varpi$ *diameter*, denoted $D_k(n)$, is the maximal value that the $k\varpi$ distance can reach. We establish a lower bound of the 1ϖ diameter, which is achieved by the same family of permutations as in the case of reversals [4].

DEFINITION 4.1 *[4] The* Gollan permutation *of length n is:*

- $[\mathbf{0}\ 3\,1\,5\,2\,7\,4 \cdots n{-}3\,n{-}5\,n{-}1\,n{-}4\,n\,n{-}2\,\boldsymbol{n{+}1}]$, *if n is even;*
- $[\mathbf{0}\ 3\,1\,5\,2\,7\,4 \cdots n{-}6\,n{-}2\,n{-}5\,n\,n{-}3\,n{-}1\,\boldsymbol{n{+}1}]$, *if n is odd.*

THEOREM 4.2 *[4] The reversal diameter is equal to $n-1$, and the Gollan permutation is diametral.*

THEOREM 4.3 *Given a Gollan permutation π of length n, we have $d_{1\varpi}(\pi) = \lfloor \frac{n}{2} \rfloor$.*

Proof. We have $b(\pi) = n+1$ by the construction of π, and Bafna and Pevzner [4] have proved $c(\pi) = 2$, therefore by Theorem 2.6 the lower bound of π is $d_{1\varpi}(\pi) \geq \lceil \frac{n-1}{2} \rceil$. Moreover, a sequence of $\lfloor \frac{n}{2} \rfloor$ can be obtained by sorting the inverse π^{-1}. Note that $\pi_1^{-1} = 2$ and $\pi_3^{-1} = 1$, therefore we can apply the 1ϖ operation $\varpi(1, 3; 1 \leftrightarrow 2)$ and the resulting permutation has the elements 1 and 2 in their corrected positions. Next, we can reduce the permutation to one with $n-2$ elements, which is exactly the inverse of the Gollan permutation with $n-2$ elements. Since two elements are sorted in one operation, we have that $\lfloor n/2 \rfloor$ operations are sufficient to sort the Gollan permutation.□

THEOREM 4.4 *The 1ϖ diameter is at least $\lfloor \frac{n}{2} \rfloor$.*

4.2. The ϖ diameter. Let us consider the diameter problem regarding the ϖ operation, remember that in such operation the number of non-reversible blocks is arbitrary. So, the number of possible operations on a permutation of length n is exponential on n, but in what follows we prove that such value is limited by $O(2^n)$.

LEMMA 4.5 *For every permutation of length n, the number of possible ϖ operations is $O(2^n)$.*

Proof. Let us count the number of possible ϖ operations $T(n)$ to be applied in a permutation of length n. We consider three cases of ϖ operations:

1. the first element is not affected by a ϖ. In this case the number of operations is the same of any permutation with $n-1$ elements, so there are $T(n-1)$ possible such ϖ operations;
2. the last element is not affected by a ϖ. Similar to the previous case, we have $T(n-1)$ possible such ϖ operations;
3. the first and the last elements are affected by a ϖ. In this case, an operation must start and end with the first and last elements, respectively. Let us count the operations with respect to the second element being moved to the first position, and the third element being moved to the first position, and so on.
   (a) starting with the second element, we have 1 possible operation:
   $[\underline{x_1\ \underline{x_2\ \cdots x_n}}] \rightarrow [x_2\ x_3\ \cdots x_n\ \boldsymbol{x_1}]$.
   (b) starting with the third element, we have 2 possible operations:
   $[\underline{x_1\ x_2\ \underline{x_3\ \cdots x_n}}] \rightarrow [x_3\ x_4\ \cdots x_n\ \boldsymbol{x_2}\ x_1]$, or
   $[\underline{\underline{x_1\ b}\ \underline{x_3\ \cdots n}}] \rightarrow [x_3\ x_4\ \cdots n\ x_1\ \boldsymbol{x_2}]$.
   (c) starting with the fourth element, we have 4 possible operations:
   $[\underline{x_1\ x_2\ x_3\ \underline{x_4\ \cdots x_n}}] \rightarrow [x_4\ x_5\ \cdots n\ \ \boldsymbol{c}\ x_2\ x_1]$, or
   $[\underline{x_1\ \underline{x_2\ x_3}\ \underline{x_4\ \cdots x_n}}] \rightarrow [x_4\ x_5\ \cdots n\ \ x_2\ \boldsymbol{x_3}\ x_1]$, or
   $[\underline{\underline{x_1\ x_2}\ x_3\ \underline{x_4\ \cdots x_n}}] \rightarrow [x_4\ x_5\ \cdots n\ \ \boldsymbol{x_3}\ x_1\ x_2]$, or
   $[\underline{\underline{x_1\ x_2\ x_3}\ \underline{x_4\ \cdots x_n}}] \rightarrow [x_4\ x_5\ \cdots n\ \ x_1\ x_2\ \boldsymbol{x_3}]$.

   In general, the number of ϖ operations starting with the i-th element is the double of the number of operations starting with the $i-1$-th element, for $i = 3, \ldots, n$, and 1 operation for $i = 2$. Considering $S(i)$ the number of such ϖ operations, we have the following recurrence: $S(i) = 2S(i-1)$, and $S(2) = 1$, where we can solve by determining the sum: $\sum_{i=2}^{n} S(i) = S(2) + S(3) + \cdots + S(n) = 2^n - 2$.

By Items 1 and 2, there are operations counted in both cases, these are ϖ operations where the first and the last elements are both not affected. By the same argument of such items we must remove $T(n-2)$ operations. Hence, $T(n) = 2T(n-1) - T(n-2) + 2^n - 2$, where $T(1) = 0$ and $T(2) = 1$. Therefore, the result

follows by solving this recurrence formula. □

Next, we prove a logarithmic lower bound on the ϖ diameter.

THEOREM 4.6 *The ϖ diameter is $D_n(n) = \Omega(\log n)$.*

Proof. By Lemma 4.5, given a permutation of length n, there are $O(2^n)$ permutations with distance equal to 1 of such permutation. The number of permutations of length n is $n! \approx 2^{n \log n}$, hence from any permutation, the number of permutations reachable at distance d is at most 2nd, therefore the diameter must be $d \geq \log n$. □

Theorem 4.6 shows a logarithmic lower bound on the diameter, such bound says that although there is an available exponential number of operations in each step to sort, the diameter is a function growing not less than a logarithmic number of operations.

Now, we show an algorithm to sort any permutation by ϖ, that is an approach to achieve a tighter upper bound on the ϖ diameter.

THEOREM 4.7 *The ϖ diameter is $D_n(n) = O(\log^2 n)$.*

Proof. Let us call "small" any element of a permutation of length n with value at most $\frac{n}{2} - 1$, and "big" the others. Our base step is to "almost-sort" the permutation so that all small elements are on the left side, followed by all big elements. It is rather clear that $\log n$ such base steps are sufficient to sort the permutation. Once the first is done, we work in parallel on each half of the permutation, and continue recursively.

Now, we perform the base step in $O(\log n)$ ϖ operations by grouping consecutive small (resp. big) elements into blocks named "S" (resp. "B"). Then the permutation follows the following pattern: i) $SBSB \cdots SB$, or ii) $BSBS \cdots BS$.

With b such blocks ($b \leq n$), $b/4$ parallel reversals in one ϖ operation reduce b by half. Considering i) we have: $S(BS)BS(BS)B \cdots S(BS)B \rightarrow SSBBSSBB \cdots SSBB$.

Thus, in $\log n$ ϖ operations we reach a string with only 2 blocks, SB. So, the overall cost is: $\log n$ (base steps) $\times$ $\log n$ (ϖ per base step) $= log^2 n$. □

5. Concluding remarks. The goal of this paper is to propose new forms of rearrangements, which are closely related to well-known rearrangements, and we deal with challenging problems. We have developed a technique to find 1ϖ tight permutations, and although such strategy suggests to achieve $k\varpi$ tight permutations, the complexity of such problem may grow exponentially on k. We have designed algorithms to sort permutations by 1ϖ, where we have proved exact distances for classes of permutations: the involution permutations and the Gollan permutations; a constant ratio of 2.67-approximation is given; and a lower bound on the 1ϖ diameter.

With respect to the $k\varpi$ diameter, one may ask if the Gollan permutation is also candidate to be diametral for $k > 1$, since it is the case for $k \in \{0, 1\}$. However, we have examples of permutations that have distance greater than the Gollan permutation, regarding the 2ϖ. Therefore, the Gollan permutation is not diametral for $k > 1$. Nevertheless, we believe that such class may suggest a general family of diametral permutations, in a $k\varpi$ problem.

The transposition diameter is still an open problem, for more than 20 years, and it is known to be $\Theta(n)$ [13], which is the case of the reversal diameter known to be exactly $n-1$ [4]. Surprisingly, we have proved that the ϖ diameter is logarithmic (Theorems 4.6 and 4.7).

In [2], there is a database of the diameters of several well known rearrangement operations. We have implemented a branch-and-bound algorithm to compute the diameters on the $k\varpi$ distance [1]. Refer to Table 5.1 to see the values of $D_0(n)$, $D_1(n)$ and $D_2(n)$, for $n \leq 10$, where we have highlighted distinct values along the same row. The values of the first line are from the known Theorem 4.2 for the reversal

diameter established in [4], the second line from our Theorem 4.4, and the values of the third line we have achieved by our branch-and-bound algorithm [1]. Note how slowly the diameter is growing when k increases.

Table 5.1: Values of the diameter $D_k(n)$ for $k = 0, 1$ and 2, and $n \leq 11$.

n	2	3	4	5	6	7	8	9	10	11
$\mathbf{D_0(n)}$	**1**	2	**3**	4	**5**	6	**7**	8	**9**	10
$\mathbf{D_1(n)}$	**1**	**1**	2	2	**3**	**3**	4	4	**5**	**5**
$\mathbf{D_2(n)}$	**1**	**1**	2	2	**3**	**3**	**3**	**3**	4	4

REFERENCES

[1] http://www.cos.ufrj.br/~lfignacio/rmb
[2] http://mirza.ic.unicamp.br:8080/bioinfo/index.jsf
[3] Alekseyev, M., Pevzner, P.: Multi-break rearrangements and chromosomal evolution. Theor. Comput. Sci. 395, 193–202 (2008)
[4] Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals. SIAM J. Comput. 25, 272–289 (1996)
[5] Bafna, V., Pevzner, P.: Sorting by transpositions. SIAM J. Discrete Math. 11, 224–240 (1998)
[6] Bergeron, A., Mixtacki, J., Stoye, J.: On sorting by translocations. J. Comput. Biol., 13, 567-âĂŞ578, (2006)
[7] Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation algorithm for sorting by reversals. In:ÂăProceedings of the 10th. Annual European Symposium. ESA, 2002. pp. 200–210 (1997)
[8] Braga, M., Machado, R., Ribeiro, L., Stoye, J.: On the weight of indels in genomic distances. BMC Bioinformatics 12(S–9): S13 (2011)
[9] Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Discrete Math. 26, 1148–1180 (2012)
[10] Bulteau, L., Fertin, G., Rusu, I.: Pancake flipping is hard. J. Comput Syst. Sci. 81, 1556–1574 (2015)
[11] Caprara, A.: Sorting by reversals is difficult. In: Proceedings of the 1st. International Conference on Computational Molecular Biology. ACM, 1997. pp. 75–83 (1997)
[12] Christie, D.A.: Genome Rearrangement Problems. Ph.D. thesis, University of Glasgow (1998)
[13] Cunha, L., Kowada, L., Hausen, R., de Figueiredo, C.: Advancing the transposition distance and diameter through lonely permutations. SIAM J. Discrete Math. 27, 1682–1709 (2013)
[14] Cunha, L., Kowada, L., Hausen, R., de Figueiredo, C.: A faster 1.375-approximation algorithm for sorting by transpositions. J. Comput. Biol. 22, 1044–1056 (2015)
[15] Feijão, P., Meidanis, J.: SCJ: a breakpoint-like distance that simplifies several rearrangement problems. IEEE IEEE/ACM Trans. Comput. Biol. Bioinform., 8, 1318âĂŞ-1329, (2011)
[16] Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. The MIT Press (2009)
[17] Heath, L.S., Vergara, J.P.C.: Sorting by short block-moves. Algorithmica 28, 323–352 (2000)
[18] Lintzmayer, C., Dias, Z.: Sorting permutations by prefix and suffix versions of reversals and transpositions. In: Proceedings of the 11th. Latin American Symposium, LATIN 2014. pp. 671–682 (2014)
[19] Kececioglu, J., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica 13, 180–210 (1995)
[20] Tran, N.: An easy case of sorting by reversals. J. Comput. Biol. 5, 83–89 (1997)
[21] Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340âĂŞ-6 (2005)