Instituto de Computação
Universidade Estadual de Campinas

Problemas de Comparação de Genomas

Ulisses Martins Dias

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Ulisses Martins Dias e aprovada pela Banca Examinadora.

Campinas, 08 de Fevereiro de 2012.

Prof. Dr. Zanoni Dias
Instituto de Computação (IC) - Unicamp
(Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Substitua pela ficha catalográfica

(Esta página deve ser o verso da página anterior mesmo no caso em que não se imprime frente e verso, i.é., até 100 páginas.)

Substitua pela folha com as assinaturas da banca

Instituto de Computação
Universidade Estadual de Campinas

Problemas de Comparação de Genomas

Ulisses Martins Dias[1]

08 de Fevereiro de 2012

Banca Examinadora:

- Prof. Dr. Zanoni Dias
  Instituto de Computação (IC) - Unicamp (Orientador)
- Prof. Dr. João Carlos Setubal
  Instituto de Química (IQ) - USP
- Prof. Dr. Nalvo Franco de Almeida Jr.
  Faculdade de Computação (FACOM) - UFMS
- Prof. Dr. João Meidanis
  Instituto de Computação (IC) - Unicamp
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação (IC) - Unicamp

[1]Suporte financeiro de:

- FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo (2007/05574-4)
- CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (BEX 1757/09-1)

Agradecimentos

Aos meus pais Otacílio Rodrigues Dias e Regina Martins Dias, meus exemplos de vida.

Às minhas irmãs Cleópatra, Ísis e Danusa, companheiras de uma vida inteira.

Ao meu orientador Zanoni Dias pelo incentivo durante esses anos de doutorado, pelas críticas construtivas feitas ao meu trabalho e pela confiança em mim depositada. Esse tempo de trabalho foi significativo para meu crescimento como pesquisador e como pessoa.

Ao professor João Carlos Setubal pela oportunidade de complementar este trabalho durante os 13 meses de doutorado sanduíche realizados no *Virginia Tech*.

Às pessoas que me auxiliaram em minha chegada no exterior. De modo especial, gostaria de agradecer a Lilian Lefol, Viviane Oliveira, Michele Hecke e Marina Quadros. Um agradecimento também a Gustavo Riboldi, com quem compartilhei todas as despesas por um ano inteiro no exterior.

Aos meus amigos que me acompanharam aqui no Brasil e nos Estados Unidos. Em especial André Atanasio, Andrew Warren, Elaine Batista, Javier Montoya, Kuan Yang e Priscila Sabóia pela participação nesta jornada.

Aos funcionários do Instituto de Computação da Unicamp pela cooperação. De modo especial a Fernando Okabi, Daniel Capeleto, Ademilson dos Reis e Flávio Luzia.

Gostaria de agradecer às agências Fapesp pelo suporte financeiro no período em que trabalhei no Brasil e Capes pelo suporte financeiro no doutorado sanduíche.

Finalmente, dedico este trabalho a minha esposa Danielle Furtado dos Santos Dias pelos anos que passamos juntos, pelo amor incondicional e por dar sentido a minha vida.

Resumo

Esta tese aborda três aspectos da comparação entre genomas: primeiro, eventos de transposição; segundo, eventos de reversão e de reversão quase-simétrica; terceiro, estudo da distância entre genomas sem ligação com algum tipo específico de rearranjo. O estudo do primeiro aspecto, eventos de transposição, permitiu a criação de um novo algoritmo de aproximação na razão 1.375 e de modelos exatos usando programação em lógica com restrições para o problema da distância de transposição. Ambas as abordagens foram comparadas com outras semelhantes encontradas na literatura e, em ambos os casos, foi mostrado que o produto aqui apresentado é superior àqueles previamente conhecidos. Sob o segundo aspecto, eventos de reversões e de reversões quase-simétricas, houve avanços relacionados ao entendimento do processo de diferenciação das espécies na família *Pseudomonadaceae* e nos gêneros *Mycobacterium*, *Shewanella* e *Xanthomonas* com a criação de uma ferramenta de simulação capaz de gerar um histórico evolutivo com características semelhantes às observadas nos referidos grupos. Além disso, foram obtidos avanços em abordagens algorítmicas para o problema de construção de *scaffolds* usando um genoma de referência. De modo particular, foi obtida uma ferramenta superior às demais existentes na literatura para construção de *scaffolds* de genomas bacteriais. Ainda neste segundo aspecto, tratou-se do problema da distância de reversões quase-simétricas com a geração de um algoritmo guloso que fornece uma sequência de reversões quase-simétricas para ordenar qualquer permutação, além de algoritmos exatos para várias famílias específicas de permutações. No que diz respeito ao terceiro aspecto, foram desenvolvidas duas medidas que podem ser calculadas de forma eficiente (poucos segundos). Uma das medidas é adequada para genomas próximos e a outra é adequada para genomas distantes. Ambas foram avaliadas com genomas bacteriais reais, o que mostrou as vantagens e limitações de cada medida. Com esta tese, espera-se ter contribuído para a área de comparação de genomas em geral e, em particular, para a área de rearranjo de genomas.

Abstract

In this PhD thesis, we work on three aspects of genome comparison: first, transposition events; second, inversion and almost-symmetric inversion events; third, whole-genome distance measures that are not connected to any specific kind of rearrangement event. The study of transposition events (first aspect) allowed us to create a new 1.375-approximation algorithm and some exact models using constraint logic programming. These approaches were compared to other published methods and in all cases our methods perform best. The second aspect of this thesis concerns inversion and almost-symmetric inversion events. In this regard, we developed a simulation tool for the study of symmetric inversions in bacterial genomes. Through this work we were able to contribute to the understanding of the evolutionary differentiation process in species of the following groups: the *Pseudomonadaceae* family, the *Xanthomonas* genus, the *Shewanella* genus, and the *Mycobacterium* genus. We used the knowledge acquired in building our simulation tool to establish a method that uses inversion signatures to generate draft genome sequence scaffolds using a complete genome as a reference. Apart from the practical applications of this research, we contribute to the computer science field by providing a theoretical framework for the almost-symmetric distance problem that can be improved in the future and can serve as a basis for approximation and heuristic algorithms. This framework is comprised of a greedy algorithm for any permutation, exact algorithms for specific families of permutation, and several lemmas and conjectures related to these problems. The third and last aspect of this thesis addresses the need for methods that can quickly and effectively compare large sets of genome sequences. We propose two new methods for efficiently determining whole genome sequence distance measures. One of them is aimed at comparing closely related genomes, and the other is meant to compare more distant genomes. Both measures were evaluated in order to find their limitations and their efficacy. It is our hope that this work represents a contribution to knowledge of the genome comparison field in general, and the genome rearrangement field, in particular.

Conteúdo

Agradecimentos **vii**

Resumo **ix**

Abstract **xi**

Apresentação **1**

I Noções Básicas **3**

1 Rearranjo de Genomas **5**

1.1 Transposição 6

1.1.1 Limitantes para o Problema da Distância de Transposição 8

1.1.2 Algoritmo de Bafna e Pevzner 12

1.1.3 Algoritmo de Elias e Hartman 15

1.2 Reversão 17

1.2.1 Reversão Simétrica 18

II Transposições **23**

2 Heurísticas para o Problema da Distância de Transposição **25**

2.1 Introdução 26

2.2 Heurísticas 26

2.3 Algoritmo 31

2.4 Análise do *Look-Ahead* 32

2.5 Análise Comparativa com Outros Algoritmos 35

2.6 Publicações 37

3 Programação em Lógica com Restrições para Transposições 39
3.1 Introdução . . . 40
3.2 Programação por Restrição . . . 41
3.3 Modelos para a Distância de Transposição . . . 46
3.4 Análise Comparativa dos Modelos . . . 51
3.5 Publicações . . . 54

III Reversões 55

4 Estudo de Reversões Simétricas em Genomas Bacteriais 57
4.1 Introdução . . . 58
4.2 Parâmetros do SIB . . . 59
4.3 Validação do SIB . . . 62
4.4 Considerações Finais sobre os Programas de Reconstrução . . . 67
4.5 Publicações . . . 68

5 Distância de Reversão Quase-Simétrica 69
5.1 Introdução . . . 70
5.2 Definições . . . 70
5.3 Limitantes . . . 73
5.4 Algoritmos de Ordenação . . . 75
5.5 Famílias e Classes de Permutações . . . 79
5.5.1 Famílias de permutações . . . 80
5.5.2 Classes de Permutações . . . 85
5.6 Publicações . . . 87

6 Geração de Scaffolds usando Assinaturas de Reversão 89
6.1 Introdução . . . 90
6.2 Algoritmos . . . 91
6.2.1 Algoritmo Básico . . . 94
6.2.2 Algoritmo para Reversões Genéricas . . . 99
6.3 Análise dos Algoritmos com Genomas Incompletos Simulados . . . 102
6.3.1 Testes Simulados . . . 102
6.3.2 Testes Simulados com Genomas Reais . . . 105
6.4 Análise dos Algoritmos com Genomas Incompletos Reais . . . 111
6.4.1 Dados de Entrada . . . 111
6.4.2 Resultados . . . 112
6.4.3 Influência das Duplicações . . . 116

6.5 Publicações 118

IV Distância entre Genomas 119

7 Distâncias entre Genomas 121
7.1 Introdução 122
7.2 Medidas de Distância 123
7.3 Análise Comparativa das Medidas 124
7.3.1 Análise com Organismos Próximos 126
7.3.2 Análise com Organismos Distantes 126
7.4 Publicações 132

8 Considerações Finais 135

A Revisão Bibliográfica para a Distância de Transposição 137
A.1 Sorting by Transpositions 137
A.1.1 Breakpoints 138
A.1.2 Grafo Orientado Cíclico com Arestas de Cores Alternadas 138
A.1.3 Propriedades do Grafo de Ciclos 140
A.2 Sorting permutations by block-interchanges 141
A.3 Sorting a Bridge Hand 141
A.3.1 Modelo toroidal de permutações 142
A.3.2 Algoritmo Ótimo para a Inversa da Identidade 142
A.3.3 Limitante superior para o problema do diâmetro 143
A.3.4 Conjecturas sobre o diâmetro 143
A.4 A 1.375-Approximation Algorithm for Sorting by Transpositions 144
A.4.1 Diâmetro de transposição 145
A.5 New Bounds and Tractable Instances for the Transposition Distance 146
A.5.1 Conceitos Iniciais 146
A.5.2 O grafo Γ 147
A.5.3 Permutações γ 148
A.5.4 Extensões para permutações α 150
A.6 Polynomial-sized ILP Models for Rearrangement Distance Problems 151
A.6.1 Restrições comuns a todos os modelos 152
A.6.2 Restrições específicas para transposições 152
A.6.3 Restrições específicas para reversões 154
A.6.4 Distância de transposição e reversão 155
A.7 Faster Algorithms for Sorting by Transpositions and Block Interchanges . . 156

A.7.1 Árvore de Permutação 156
A.8 An Alternative Algebraic Formalism for Genome Rearrangements 159
A.8.1 Ciclos 160
A.8.2 Formalisto algébrico aplicado a rearranjo de genomas 160
A.9 Transposition Distance based on the Algebraic Formalism 161

Bibliografia **165**

Lista de Tabelas

1.1 Significância Estatística 20

2.1 Número de configurações presentes no banco de dados por número de elementos. 31
2.2 Quantidade de respostas não ótimas fornecidas por cada algoritmo ao variar o valor do *look-ahead* no intervalo [0,2]. Na tabela, t-LA identifica o algoritmo proposto com a Heurística 4 usando o valor t para *look-ahead*. . . 34
2.3 Média do tempo (em minutos) consumido para cada permutação. **0-LA** é o algoritmo sem *look-ahead*. **1-LA** é o algoritmo com 1 *look-ahead*, **2-LA** é o algoritmo com 2 *look-aheads*. 37
2.4 Resultados para outros algoritmos encontrados na literatura: (WDM) se refere ao algoritmo descrito e implementado por Walter, Dias e Meidanis [96], (C/WCO) se refere ao algoritmo descrito por Christie [27] e implementado por Walter, Curado and Oliveira [95], (HS/H) se refere ao algoritmo descrito por Hartman e Shamir [58] e implementado por Honda [63], (BP/W) é o algoritmo de Bafna e Pevzner [9] implementado com heurísticas por Walter e co-autores [97], (M) se refere ao algoritmo descrito e implementado por Mira e co-autores [76], (EH/DD) se refere ao algoritmo de Elias and Hartman implementado por Dias e Dias [35]. 38

3.1 Tempo médio (em segundos) para calcular a distância de permutações de acordo com o seu tamanho. O símbolo "−" é mostrado caso a bateria de testes não seja finalizada em 15 horas. 52
3.2 Tempo médio (em segundos) do modelo **cg_csp** quando são aplicadas as técnicas para melhorar a performance. 53

4.1 Genomas da família *Pseudomonadaceae*. 60

5.1 Valores médios de $Lower(\pi)$, $d(\pi)$ e $Upper(\pi)$ para todas as permutações π com tamanho menor ou igual a 10. A coluna **Diam** mostra o valor do diâmetro para um dado tamanho de permutação. 76

5.2 Performance do Algoritmo 4. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata 79

5.3 Performance do Algoritmo 5. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata 80

5.4 Performance do Algoritmo 6. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata 81

5.5 Lista de distâncias fornecidas pelas conjecturas sobre as famílias F_1 a F_{10}. Todos os valores conjecturados na tabela coincidem com os valores exatos de distância para $N \leq 10$. A última coluna mostra o valor do diâmetro para cada valor de N. 83

6.1 Organismos usados nos testes. A coluna **Cobertura** contém a fração de cada cromossomo coberta pelos contigs. A coluna **Cromossomo** contém a chave de referência do cromossomo no NCBI. A coluna **Tamanho** mostra o tamanho do cromossomo em pares de base. A coluna **Contigs** mostra o número de contigs presentes no genoma incompleto. 112

6.2 Performance de cada programa em cada uma das instâncias de testes. Os números se referem a uma média das porcentagem de adjacências corretas. O melhor resultado é realçado em negrito. 113

6.3 Porcentagem das instâncias em que cada programa forneceu os melhores resultados em termos de adjacências nos testes com *drafts* reais. As colunas não somam a 100% devido aos casos de empate. Melhor resultado realçado em negrito. 113

6.4 Performance de cada programa em cada uma das instâncias de testes. Os números se referem a uma média das coberturas. O melhor resultado é realçado em negrito. 116

6.5 Performance de cada programa nos testes. Os números representam a quantidade de vezes que um dado programa fornece o melhor resultado, usando como critério a cobertura. Os melhores resultados são realçados em negrito. 117
6.6 Porcentagem de adjacências corretas para contigs com duplicações. Resultados obtidos com SIS (nucmer e promer). 117
6.7 Porcentagem de adjacências corretas para contigs sem duplicações. Resultados obtidos com SIS (nucmer e promer). 118

7.1 Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas próximos. Os resultados correspondem ao número de organismos presentes na MAST calculada usando como entrada a árvore de referência do organismo e as árvores geradas usando as três medidas. A coluna "Organismos" indica o número de espécies utilizadas. Os melhores resultados são realçados em negrito. 127
7.2 Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas distantes. Os resultados correspondem ao número de organismos presentes na MAST calculada usando como entrada a árvore de referência do organismo e as árvores geradas usando as três medidas. A coluna "Organismos" indica o número de espécies utilizadas. Os melhores resultados são realçados em negrito. 127
7.3 Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas distantes. Os resultados correspondem à razão $\frac{|g|}{|G|}$, onde $|g|$ é o número de espécies no conjunto g e $|G|$ é o número de espécies na sub-árvore G, que é a menor sub-árvore que contém todos os organismos de g. Essa razão mede o quão agrupados na árvore T estão as espécies de g. Valores próximos de 1 indicam um agrupamento mais coeso que valores próximos de 0. A coluna "Média" contém a média simples dos valores de MUMi, NUCMi e PROMi. 131
7.4 Porcentagem de acertos das medidas MUMi, NUCMi e PROMi por grupo de organismos. Valores próximos de 100 indicam uma maior semelhança com a matriz de referência. A coluna "Média" contém a média simples dos valores de MUMi, NUCMi e PROMi. 132

8.1 Resumo da produção científica. 136

Lista de Figuras

1.1 Transposição $\rho(i, j, k)$ agindo em uma permutação. 7

1.2 Grafo de ciclos $G(\pi)$ para π = (5 2 1 4 3). O grafo contém dois ciclos: C_1 = (5,1,3) e C_2 = (6,4,2). C_1 é um ciclo orientado e C_2 é um ciclo não orientado. 10

1.3 Grafo Γ para π = (5 2 1 4 3) 11

1.4 Exemplo de transposições: a) 2-move válido, b) 0-move válido e c) 0-move bom. 12

1.5 Exemplo de sequências: a) (4,3)-sequence, b) (3,2)-sequence. 13

1.6 Procedimento para transformar um k-ciclo longo em dois ciclos: o primeiro ciclo terá tamanho 3 e o segundo ciclo terá tamanho $k - 2$. 16

1.7 Alinhamento de genomas usando a ferramenta MUMmer [68]. O alinhamento apresenta com nitidez o padrão em "X" frequentemente observado em bactérias. Neste gráfico, as regiões em vermelho correspondem a regiões conservadas (*matches*) que ocorrem na mesma direção em ambos os genomas, enquanto que as regiões em azul correspondem a regiões conservadas em direções opostas. 19

1.8 Modelo de evolução proposto por Eisen e co-autores [45] usando reversões simétricas à origem de replicação. Um gráfico do alinhamento foi criado para cada geração de modo a ilustrar a evolução dos genomas. Os números 1-18 nos cromossomos se referem a segmentos de DNA compartilhados entre os genomas. 21

2.1 Exemplo de *look-ahead* na permutação π = (3 2 1 5 4). 29

2.2 Diagrama de Venn mostrando a porcentagem de vezes em que cada versão do algoritmo fornece o melhor resultado. 35

2.3 Número relativo de vezes em que cada versão do algoritmo fornece o melhor resultado. É importante ressaltar que mais de uma versão pode fornecer o melhor resultado. 36

4.1 Uma assimetria ocorre caso qualquer um dos lados da inversão ocorra dentro de um bloco inquebrável. O ponto `A` da reversão foi deslocado de modo a incluir o bloco inquebrável na reversão simétrica. 59
4.2 *Dotplot* formado ao alinhar genomas da família *Pseudomonadaceae*. O dotplot é composto de regiões conservadas que ocorrem com a mesma orientação (regiões vermelhas) e regiões conservadas que ocorrem com orientações opostas (regiões azuis). 61
4.3 Histograma das distâncias euclidianas entre *dotplots* obtidos de organismos reais e o *dotplot* mais próximo obtido na simulação 63
4.4 Comparação entre a árvore da família *Pseudomonadaceae* produzida pelo SIB e a árvore proposta por Setubal et al [90]. 64
4.5 Comparação entre a árvore do grupo *Mycobacterium* produzida pelo SIB e a árvore proposta por Alam et al [2]. 65
4.6 Comparação entre a árvore do grupo *Xanthomonas* produzida pelo SIB e a árvore proposta por Moreira et al [77]. 65
4.7 Comparação entre a árvore do grupo *Shewanella* produzida pelo SIB e a árvore proposta por Williams et al [99] 66
4.8 Exemplo de incongruência entre a árvore gerada pelo SIB e árvore de máxima verossimilhança. O agrupamento obtido pelo SIB é mais consistente com os *dotplots*. 66
4.9 Cenário evolutivo criado para avaliar a reconstrução do genoma ancestral assumindo que o único evento possível é a reversão. A reconstrução foi feita com o programa MGR. 67
4.10 Na parte de cima estão os *dotplots* que resultam do alinhamento do ancestral reconstruído e dos três descendentes A, B and C. Na parte de baixo da figura são mostrados *dotplots* corretos, que deviam ter sido obtidos pela ferramenta de reconstrução. 68

5.1 Representação de genomas. A origem de replicação é representada pelo número "0" e a ordem de leitura está no sentido horário. 71
5.2 Exemplo das funções Position (p), Sign (s) e Slice ($slice$) para a permutação $(-7\ +1\ +3\ +6\ +8\ +5\ -2\ +4)$. 71
5.3 Reversão simétrica 72
5.4 Número de reversões simétricas necessárias para posicionar um elemento quando o caminho direto resultaria em um posicionamento com o sinal negativo. Dois casos são ilustrados, um para π de tamanho par e outro para π de tamanho ímpar. 74

5.5 Funções *Lower* e *Upper* aplicadas à permutação $\pi = (-7\ -1\ +3\ +6\ +8\ -5\ -2\ +4)$. 76

5.6 Distribuição das distâncias exatas para todas as permutações com 10 ou menos elementos 77

5.7 Distâncias médias (e limitantes) para permutações de tamanho menor ou igual a 100. Nos testes foram geradas 100 permutações aleatórias para cada valor n no conjunto $\{5, 10, 15, 20, \ldots, 100\}$. 82

5.8 Gráficos das Famílias de Permutações. Os gráficos mostram os resultados obtidos usando os algoritmos específicos para cada família, os valores dos limitantes e os resultados obtidos pelos algoritmos básico e guloso. 86

6.1 Número de genomas microbiais sequenciados até o estágio *draft* e genomas microbiais completamente sequenciados. 90

6.2 Duas regiões com assinaturas de reversão: a região a_1, a_2 no genoma incompleto ("draft") foi pareada com a região b_1, b_3 do genoma de referência. O fato de b_1 e b_3 não serem consecutivos no genoma de referência caracteriza o que chamamos de breakpoint e o fato de o alinhamento entre a_1 e b_1 possuir uma orientação diferente do alinhamento de a_2 e b_3 caracteriza o que chamamos de assinatura de reversão. A mesma situação explicada acima ocorre entre a assinatura de reversão correspondente às regiões $(a3, a4)$ de A e $(b2, b4)$ de B. Ambas as assinaturas se relacionam por terem sido formadas pela ação de uma única reversão e, neste caso, denominamos estas assinaturas de IS-Pairs. 92

6.3 Três reversões simétricas ($\rho(3,6)$, $\rho(4,5)$ e $\rho(2,7)$) aplicadas na permutação identidade. Note que o resultado final não é alterado caso as reversões simétricas fossem aplicadas em outra ordem. Os círculos colorido representam IS-Pairs criados pelas reversões simétricas. 93

6.4 Três reversões aninhadas ($\rho(1,7)$, $\rho(3,6)$ e $\rho(5,5)$) aplicadas na permutação identidade. Note que o resultado final é alterado caso essas reversões sejam aplicadas em outra ordem. Qualquer sequência de reversões simétricas pode ser ordenada de forma aninhada, com o mesmo resultado original. Os círculos colorido representam IS-Pairs criados pelas reversões aninhadas. . . 93

6.5 Quatro reversões seguras ($\rho(8,8)$, $\rho(1,2)$, $\rho(4,6)$ e $\rho(5,5)$) aplicadas na permutação identidade. Toda sequência de reversões aninhadas é uma sequência de reversões seguras (o inverso não é verdade). Os círculos colorido representam IS-Pairs criados pelas reversões seguras. 94

6.6 Cinco reversões genéricas ($\rho(1,5)$, $\rho(1,4)$, $\rho(2,3)$, $\rho(7,7)$ e $\rho(3,7)$) aplicadas na permutação identidade. Os círculos brancos representam os ISs destruídos por reversões não seguras. 95

6.7 Exemplo de um genoma com 40 blocos conservados em relação ao um genoma de referência. Os blocos conservados estão divididos em 9 contigs. A diferenciação dos genomas pode ser explicada por 7 reversões seguras. Das 14 assinaturas de reversões, apenas 9 podem ser identificadas nos contigs. Ambas as assinaturas do IS-Pair gerado pela reversão que agiu nos contigs 3 e 7 foram completamente perdidas por falta de cobertura dos contigs. Na primeira linha de contigs exibimos a reconstrução perfeita, enquanto na segunda linha, a reconstrução realizada pelo nosso algoritmo. 98

6.8 Gráfico com o número médio de adjacências corretas em relação ao número de reversões de cada conjunto. 103

6.9 Gráfico com o número médio de breakpoints perdidos em relação ao número de reversões de cada conjunto. 104

6.10 Porcentagem de casos de testes com número de adjacências corretas acima de um dado valor no eixo X. 104

6.11 Porcentagem de scaffold perfeitos em relação ao número de reversões de cada conjunto. 105

6.12 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Mycobacterium*. Nesse gráfico, foram considerados todos os 420 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds. 106

6.13 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Pseudomonadaceae*. Nesse gráfico, foram considerados todos os 306 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds. 107

6.14 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Shewanella*. Nesse gráfico, foram considerados todos os 380 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds. 107

6.15 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Xanthomonas*. Nesse gráfico, foram considerados todos os 72 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds. 108

6.16 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Mycobacterium* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds. 108

6.17 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Pseudomonadaceae* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds. 109

6.18 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Shewanella* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds. 109

6.19 Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Xanthomonas* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds. 110

6.20 Variação de performance de cada programa quando a referência varia do genoma mais próximo para o genoma mais distante, em um total de 20 genomas. O termo **fs** foi usado para referenciar o programa fillScaffolds. . . 114

6.21 Variação de performance de cada programa quando a referência varia do genoma mais próximo para o genoma mais distante, em um total de 20 genomas. Esta é uma versão com os valores médios acumulados do gráfico exibido na Figura 6.20. O termo **fs** foi usado para referenciar o programa fillScaffolds. 115

7.1 A projeção dos segmentos conservados nos eixos x e y é ilustrada com a cor verde. Essa projeção é usada para calcular as medidas NUCMi e PROMi 124

7.2 MAST calculada a partir de duas árvores de entrada (Árvore 1 e Árvore 2). A MAST é a sub-árvore com maior número de folhas contida em ambas as árvores de entrada. 125

7.3 Comparação da árvore gerada pelo NUCMi e pelo MUMi com genomas de espécies próximas (família *Pseudomonadaceae*). As linhas sólidas são utilizadas para demarcar a concordância entre as árvores geradas pelas duas abordagens e a as árvores de referência (MAST), enquanto que as linhas tracejadas evidenciam as regiões com discordâncias. 128

7.4 Comparação da árvore gerada pelo NUCMi e pelo MUMi com genomas de espécies próximas (gênero *Mycobacterium*). As linhas sólidas são utilizadas para demarcar a concordância entre as árvores geradas pelas duas abordagens e a as árvores de referência (MAST), enquanto que as linhas tracejadas evidenciam as regiões com discordâncias. 129

A.1 Grafo de ciclos para a permutação $\pi = (4\ 2\ 5\ 3\ 1)$ 139
A.2 Ciclos C_1 e C_2 para a permutação $\pi = (4\ 2\ 5\ 3\ 1)$. 140
A.3 Grafo Γ para a permutação $\pi = (4\ 1\ 6\ 2\ 5\ 7\ 3)$ 148
A.4 Grafo $\Gamma(\pi)$ para a permutação $\pi = (3\ 2\ 1\ 4\ 7\ \ 6\ 9\ 8\ 5)$. Esta permutação é um exemplo de permutações γ, que são permutações reduzidas com os elementos pares na posição correta. 148
A.5 Árvore de permutação para $\pi = (9\ 6\ 1\ 4\ 7\ 5\ 2\ 3\ 8)$ 156

Apresentação

Evolução é o conjunto de mudanças sofridas pelos organismos ao longo do tempo. Tais mudanças são observáveis tanto em intervalos curtos, como de uma geração para outra, quanto em intervalos mais longos na ordem de milhões de anos. Neste último caso, determinar o parentesco entre as espécies depende de dois fatores:

1. A identificação de características que indiquem um ancestral comum;

2. A obtenção de um grupo de fenômenos que explicariam as diferenças observadas nas espécies.

A primeira abordagem utilizada pelos cientistas determinava o parentesco entre as espécies baseando-se em observações fenotípicas. Neste contexto, seria possível obter conclusões do tipo: "O homem é mais próximo de um chimpanzé do que de um cavalo", devido a uma série de semelhanças como, por exemplo, habilidade das mãos, rotação e liberdade de movimentos dos ombros e capacidade de andar com apenas duas pernas.

Entretanto, ainda não havia nenhum entendimento sobre a origem das diferenças observadas nas duas espécies. Tal conhecimento começou a ser delineado com o advento da genética moderna. As características fenotípicas foram substituídas pelo material genético dos organismos e os fenômenos causadores das divergências evolutivas são as mutações ocorridas nesse material genético.

Essas mutações alteram a cadeia genética e podem resultar em três cenários.

1. A mutação não afeta o organismo. Neste caso, pode ter ocorrido em uma região não codificante.

2. A mutação debilita o organismo. Neste caso, a seleção natural atuará sobre o organismo diminuindo as chances de reprodução;

3. A mutação proporciona vantagem evolutiva. Neste caso, o organismo possui grandes chances de propagar o gene mutante para as gerações seguintes.

Os eventos de mutação mais comuns afetam pontualmente o genoma, inserindo, removendo ou substituindo nucleotídeos. Obter o parentesco entre duas espécies usando mutações pontuais consiste em encontrar sequências com alta similaridade nos genomas. Essas sequências indicam a existência de um ancestral comum, mas ainda é preciso obter as mutações que explicam as diferenças entre as sequências. Para tanto, utiliza-se a distância de edição, número mínimo de operações de inserção, deleção e substituição que transformam uma sequência em outra.

A distância de edição parte do princípio de que a natureza utiliza o número mínimo de operações, uma teoria conhecida como método da máxima parcimônia. A distância de edição estima a distância evolutiva entre duas espécies. Entretanto, ela não detecta operações globais, os chamados rearranjos de genomas.

Os rearranjos produzem genomas com grandes blocos de sequências com ordem ou orientação diferentes do original. Em alguns casos, os rearranjos modificam a arquitetura dos genomas, criando novos cromossomos ou unindo dois cromossomos em um só.

Dados dois genomas, a presença de blocos conservados em regiões diferentes é um indício de ascendência comum. Entretanto, estimar a distância evolutiva entre os genomas exige identificar o grupo de rearranjos causadores do posicionamento desigual dos blocos conservados nos genomas. Os eventos mais comuns de rearranjo são: reversões, transposições, translocações, fusões e fissões.

Um conceito de distância pode ser definido para qualquer classe de rearranjo como o menor número de ocorrências necessárias para transformar um genoma em outro. Esse conceito utiliza novamente o método da máxima parcimônia para explicar a evolução de organismos por rearranjo de genomas. Por exemplo, chama-se distância de reversão o menor número de reversões necessárias para transformar um genoma em outro e a distância de transposição é, dessa forma, o menor número de transposições.

Os rearranjos são mais raros que mutações pontuais, isso torna a presença deles eficaz para calcular a distância evolutiva entre as espécies. Na literatura, são documentados vários avanços relacionados a implementação de métodos computacionais para rearranjos de genomas, alguns serão revistos no decorrer desta tese.

Esta tese está inserida em três temas: distância de transposição (Parte II), distância de reversão simétrica (Parte III), um tipo mais específico de reversão, e distâncias não vinculadas a nenhum tipo específico de rearranjo (Parte IV). Um capítulo com conceitos básicos da área de rearranjo foi inserido para permitir a leitura da tese sem a necessidade de material suplementar (Parte I).

Parte I

Noções Básicas

Capítulo 1

Rearranjo de Genomas

Durante o curso da evolução, mudanças genéticas criaram diferentes espécies de seres-vivos. Muitas das mudanças impedem que a informação contida nos genes seja expressa ou resultam em um produto de expressão diferente do original. Na maioria dos casos, o organismo portador é debilitado e não sobrevive à pressão evolutiva. Entretanto, algumas mutações proporcionam vantagens no processo de seleção natural e criam organismos capazes de propagar os genes mutantes para as gerações seguintes.

As espécies evoluíram por esse mecanismo, o que em tese permitiria calcular graus de parentesco contabilizando o número de mutações entre os genomas. Infelizmente, é impossível provar que o número calculado de mutações reflete exatamente o que ocorreu, mas o princípio da máxima parcimônia sugere que o número mínimo é uma boa aproximação. A máxima parcimônia tem como diretriz que a explicação mais simples costuma ser a correta. Assim, o número mínimo de mutações é o mais usado para geração de árvores filogenéticas.

Rearranjos de genomas são um grupo de mutações que afetam um trecho do genoma, em contraponto com as mutações pontuais que afetam apenas um nucleotídeo. Os eventos de rearranjo são mais raros que mutações pontuais, o que os torna mais eficazes para calcular a distância evolutiva entre espécies. Os eventos mais comuns de rearranjo são: reversões, transposições, translocações, fusões e fissões.

- **Reversões** ocorrem quando um segmento do genoma é destacado e inserido no mesmo lugar, mas com ordem invertida em relação à posição original [8].

- **Transposições** ocorrem quando um segmento do genoma é destacado e inserido em outra posição. A ordem do segmento permanece a mesma [7, 9].

- **Translocações** ocorrem quando dois cromossomos $X = (X_1, X_2)$ e $Y = (Y_1, Y_2)$ interagem para formar os cromossomos $X' = (X_1, Y_2)$ e $Y' = (Y_1, X_2)$ [55].

- **Fusões** ocorrem quando dois cromossomos se unem para formar um só cromossomo [42].

- **Fissões** ocorrem quando um cromossomo se divide em dois cromossomos [42].

Os rearranjos produzem genomas com blocos de sequências em posições diferentes do original. Os blocos são as regiões com alta similaridade entre os genomas e podem ser genes, conjunto de genes ou qualquer subsequência conservada. Para que seja possível modelar as operações de rearranjo computacionalmente, cada bloco recebe um identificador. Seja ξ um conjunto de identificadores, um cromossomo é uma permutação de identificadores do conjunto ξ e um genoma é uma coleção de cromossomos.

Em geral, assume-se que nenhum dos genomas possui blocos duplicados de sequências. Assim, cada elemento no conjunto ξ identificará apenas um bloco. Normalmente, $\xi \subset \mathbb{N}$, sendo que sinais positivos e negativos são associados aos identificadores quando houver informação sobre a orientação dos blocos.

Ao comparar dois genomas com n blocos conservados, é comum numerar os blocos de um deles com os números naturais no intervalo $[1..n]$, sendo que o elemento 1 identificará o primeiro bloco, o elemento 2 identificará o segundo bloco e assim sucessivamente até que o elemento n identifique o último bloco. O segundo genoma é numerado de acordo com o primeiro, levando-se em conta que o sinal negativo deve ser usado sempre que os blocos possuírem orientações diferentes.

Nesta tese, novos resultados foram obtidos para os eventos de transposição e reversão simétrica, uma classe específica de reversão. A Seção 1.1 detalha os avanços para o problema de transposição, focando em soluções algorítmicas clássicas para o problema, em especial os trabalhos de Bafna e Pevzner [9] e de Elias e Hartman [46].

A Seção 1.2 lista uma série de avanços para o problema da reversão. Como os resultados apresentados nesta tese não estão relacionados ao problema das reversões em si, nenhum desses avanços foi explicado em detalhes. A Seção 1.2.1 introduz o problema de reversão simétrica e foca no modelo de Eisen e co-autores [45].

1.1 Transposição

Eventos de transposição ocorrem quando blocos adjacentes no genoma trocam de posição. Formalmente, representa-se um genoma como uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, $\pi_i \in \mathbb{N}$, $0 < \pi_i \leq n$ e $i \neq j \leftrightarrow \pi_i \neq \pi_j$. Transposições não alteram a orientação dos blocos, o que elimina a necessidade de associar sinais aos identificadores.

Uma transposição $\rho(i, j, k)$, onde $1 \leq i < j < k \leq n+1$, remove o intervalo $[i, j-1]$ de π e o insere entre π_{k-1} e π_k. O resultado é a permutação $\pi' = (\pi_1\ \ldots\ \pi_{i-1}\ \pi_j\ \ldots \pi_{k-1}\ \pi_i$

$\pi_{i+1} \dots \pi_{j-1}\ \pi_k \dots \pi_n$). A Figura 1.1 mostra uma transposição atuando em uma permutação.

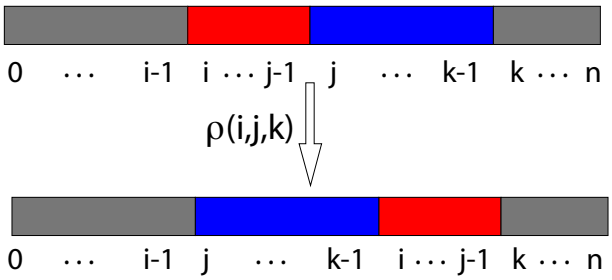


Figura 1.1: Transposição $\rho(i, j, k)$ agindo em uma permutação.

A distância de transposição $d(\pi, \sigma)$ entre duas permutações π e σ é o número mínimo t de transposições $\rho_1, \rho_2, \dots \rho_t$ tal que $\pi\rho_1\rho_2 \dots \rho_t = \sigma$. Seja ι a permutação identidade, $\iota = (1\ 2 \dots\ n)$, a distância de transposição entre π e σ é equivalente à distância de transposição entre $\sigma^{-1}\pi$ e ι, onde $\sigma^{-1}\pi$ é a permutação gerada ao listar a posição em σ de cada elemento em π. Dessa forma, o problema da distância de transposição é equivalente ao problema de ordenação por transposição. A distância entre uma permutação π e a identidade ι é representada por $d(\pi)$.

O primeiro algoritmo de aproximação foi apresentado por Bafna e Pevzner [7, 9] e possui razão de aproximação 1.5. Bafna e Pevzner afirmaram que o algoritmo poderia ser executado em tempo $O(n^2)$. Entretanto, Christie [27] argumenta que detalhes técnicos para implementação do algoritmo foram omitidos, sendo que a complexidade de $O(n^2)$ não pode ser atingida na prática.

Dentre as contribuições do trabalho de Bafna e Pevzner, é possível citar uma estrutura em grafos chamada de grafo direcionado cíclico com arestas de cores alternadas, ou simplesmente grafo de ciclos. Essa estrutura permite a definição de limitantes inferiores e superiores para o problema da distância de transposição.

Christie produziu um algoritmo com fator de aproximação 1.5 mais simples que o de Bafna e Pevzner, mas que executa em $O(n^4)$ [27].

Walter, Dias e Meidanis desenvolveram um algoritmo com complexidade de tempo da ordem $O(n^2)$, porém o fator de aproximação é 2.25 [96]. Esse algoritmo é baseado no conceito de *breakpoints*, elementos adjacentes em um dos genomas, mas não no outro.

Elias e Hartman [46] desenvolveram um algoritmo de aproximação na razão de 1.375, um avanço na razão de aproximação que não ocorria desde a publicação do trabalho de Bafna e Pevzner, oito anos antes.

Embora o problema da distância de transposição seja NP-difícil [16], algumas classes de instâncias polinomiais podem ser definidas. Neste contexto, incluem-se os trabalhos

de Fortuna [51] e Labarre [69]. Em ambos os casos, o tamanho da classe é polinomial e muito inferior ao número de instâncias possíveis para o problemas.

Uma segunda contribuição de Labarre [69] são os limitantes inferiores definidos para o problema. Os limitantes foram definidos com a utilização de uma estrutura de dados chamada de grafo Γ.

O Apêndice A fornece resumos de alguns dos trabalhos citados nesta seção. Recomenda-se a leitura desses resumos caso não haja familiaridade com a extensa área de distância de transposição.

Este capítulo está organizado da seguinte forma. A Seção 1.1.1 define os principais limitantes para o problema da distância de transposição, bem como as estruturas de dados utilizadas para gerá-los. A Seção 1.1.2 descreve o algoritmo de Bafna e Pevzner [9]. A Seção 1.1.3 descreve o algoritmo de Elias e Hartman [46].

1.1.1 Limitantes para o Problema da Distância de Transposição

Um breakpoint para uma permutação π é um par (π_i, π_{i+1}) tal que $\pi_{i+1} \neq \pi_i + 1$. Representa-se por $b(\pi)$ o número de breakpoints na permutação π. Uma transposição ρ aplicável a π pode modificar o número de breakpoints. A notação $\Delta b(\rho, \pi)$ representa a variação no número de breakpoints de π, após ser aplicada a transposição ρ: $\Delta b(\rho, \pi) = b(\pi\rho) - b(\pi)$.

Christie [27] apresentou o Lema 1.1 relacionado a breakpoints de transposição.

Lema 1.1 *Para qualquer permutação π, existe uma sequência mínima de transposições $\rho_1, \rho_2, \ldots, \rho_t$ tal que $\pi\rho_1\rho_2 \ldots \rho_t = \iota$ e $\Delta b(\rho_i, \pi\rho_1\rho_2 \ldots \rho_{i-1}) \leq 0$, para $1 \leq i \leq t$.*

Os breakpoints dividem uma permutação em *strips*, que são intervalos maximais sem breakpoints. Christie [27] mostrou que toda permutação π pode ser transformada em uma permutação reduzida π_{red} com $d(\pi) = d(\pi_{red})$. A transformação é feita em quatro passos:

1. Remover a primeira *strip*, se ela iniciar com 1;
2. Remover a última *strip*, se ela terminar com n;
3. Substituir cada *strip* pelo menor elemento contido nela;
4. Renumerar a sequência final de modo a obter uma permutação válida.

Por exemplo, a permutação $\pi = (1\ 2\ 5\ 3\ 4\ 6\ 7\ 9\ 8\ 10\ 11)$ possui $n = 11$ e pode ser reduzida a uma permutação $\pi_{red} = (2\ 1\ 3\ 5\ 4)$ com apenas 5 elementos:

1. A permutação π possui 7 *strips*:

$$\pi = (1\ 2\ \bullet\ 5\ \bullet\ 3\ 4\ \bullet\ 6\ 7\ \bullet\ 9\ \bullet\ 8\ \bullet\ 10\ 11)$$

2. A primeira *strip* inicia com 1 e pode ser removida:

$$\pi = (5\ \bullet\ 3\ 4\ \bullet\ 6\ 7\ \bullet\ 9\ \bullet\ 8\ \bullet\ 10\ 11)$$

3. A última *strip* também pode ser removida, pois termina com $n = 11$:

$$\pi = (5\ \bullet\ 3\ 4\ \bullet\ 6\ 7\ \bullet\ 9\ \bullet\ 8)$$

4. Cada *strip* restante é substituída pelo seu menor elemento:

$$\pi = (5\ \bullet\ 3\ \bullet\ 6\ \bullet\ 9\ \bullet\ 8)$$

5. A sequência resultante é renumerada de modo a se tornar uma permutação válida:

$$\pi = (2\ \bullet\ 1\ \bullet\ 3\ \bullet\ 5\ \bullet\ 4)$$

Uma transposição ρ atua em três pontos de uma permutação π. Assim, $\Delta b(\rho, \pi) \in \{-3, -2, -1{,}0, +1, +2, +3\}$. Para qualquer permutação, é sempre possível encontrar uma transposição que diminua em pelo menos 1 o número de breakpoints. As melhores transposições são aquelas que diminuem em 3 o número de breakpoints. Os Lemas 1.2 e 1.3 resultam dessas observações.

Lema 1.2 *Para qualquer permutação π, $d(\pi) \geq \frac{b(\pi)}{3}$.*

Lema 1.3 *Para qualquer permutação π, $d(\pi) \leq b(\pi)$*

Bafna e Pevzner [9] apresentaram uma estrutura de grafos chamada de grafo de ciclos. Essa estrutura é a base do algoritmo de razão 1.5 e de uma série de limitantes. O grafo de ciclos é proposto da seguinte maneira:

Definição 1.1 *O grafo de ciclos $G(\pi)$ de uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ é formado pelo conjunto de vértices $V(\pi)$, o conjunto de arestas pretas $E_p(\pi)$ e o conjunto de arestas cinzas $E_c(\pi)$:*

- $V(\pi) = \{-\pi_1, +\pi_1, -\pi_2, +\pi_2, \ldots, -\pi_n, +\pi_n\} \cup \{0, -(n+1)\}$
- $E_c(\pi) = \{+(i-1), -i | 1 \leq i \leq n+1\}$

- $E_p(\pi) = \{-\pi_i, +\pi_{i-1} | 1 \leq i \leq n+1\}$

A Figura 1.2 mostra o grafo de ciclos para a permutação $\pi = (5\ 2\ 1\ 4\ 3)$.

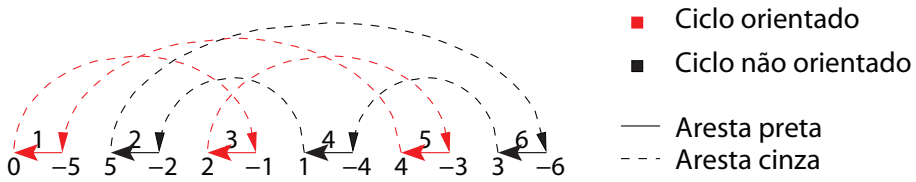


Figura 1.2: Grafo de ciclos $G(\pi)$ para $\pi = (5\ 2\ 1\ 4\ 3)$. O grafo contém dois ciclos: $C_1 = (5,1,3)$ e $C_2 = (6,4,2)$. C_1 é um ciclo orientado e C_2 é um ciclo não orientado.

Para todo vértice em $G(\pi)$, existe uma aresta cinza pareada com uma aresta preta. Isso implica em uma decomposição única das arestas de $G(\pi)$ em ciclos com arestas de cores alternadas (arestas pretas e arestas cinzas).

A Figura 1.2 ilustra algumas convenções para o grafo de ciclos:

1. As arestas pretas recebem rótulos de acordo com sua posição no grafo: o rótulo da aresta $(-\pi_i, +\pi_{i-1})$ é igual a i;

2. Um ciclo C é representado pela listagem dos rótulos das arestas pretas na ordem em que aparecem no ciclo, $C = (i_1, \ldots, i_k)$. Assume-se que i_1 é a aresta preta rotulada com o maior número em C. Os ciclos da Figura 1.2, $C_1 = (5,1,3)$ e $C_2 = (6,4,2)$,ilustram esta convenção.

Um k-ciclo em $G(\pi)$ é um ciclo que possui k arestas pretas. Este ciclo é par se k for par e, caso contrário, o ciclo é dito ímpar. As notações $c(G(\pi))$ e $c_{impar}(G(\pi))$ representam o número total de ciclos e o número de ciclos ímpares em $G(\pi)$, respectivamente.

A permutação identidade é a única com $n+1$ ciclos, todos ímpares e de tamanho 1. Assim, a sequência $\rho_1, \rho_2, \ldots, \rho_t$ que ordena π deve incrementar $c(G(\pi))$ e $c_{impar}(G(\pi))$ até $n+1$.

Sejam $\Delta c(G(\pi), \rho)$ e $\Delta c_{impar}(G(\pi), \rho)$ as variações no número total de ciclos e no número de ciclos ímpares em $G(\pi)$ após ser aplicada a transposição ρ, $\Delta c(G(\pi), \rho) = c(G(\pi\rho)) - c(G(\pi))$ e $\Delta c_{impar}(G(\pi), \rho) = c_{impar}(G(\pi\rho)) - c_{impar}(G(\pi))$. Bafna e Pevzner [9] mostraram que $\Delta c(G(\pi), \rho), \Delta c_{impar}(G(\pi), \rho) \in \{2, 0, -2\}$, o que leva ao novo limitante inferior enunciado pelo Lema 1.4.

Lema 1.4 *Para qualquer permutação* π, $d(\pi) \geq \frac{n+1-c_{impar}(G(\pi))}{2}$.

Bafna e Pevzner [9] também provaram o seguinte limitante superior:

Lema 1.5 *Para qualquer permutação* π, $d(\pi) \leq \frac{3}{4}(n + 1 - c_{impar}(G(\pi)))$

Limitantes foram introduzidos por Labarre [69] usando uma estrutura chamada de grafo Γ (Figura 1.3).

Definição 1.2 *O grafo* $\Gamma(\pi)$ *de uma permutação* $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ *é formado pelo conjunto de vértices* $\{\pi_1, \ldots, \pi_n\}$ *e pelo conjunto de arestas* $\{(\pi_i, \pi_j)|\pi_i = j\}$

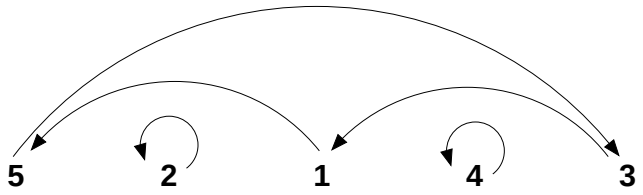


Figura 1.3: Grafo Γ para $\pi = (5\ 2\ 1\ 4\ 3)$

O grafo Γ apresenta algumas semelhanças em relação ao grafo de ciclos. Por exemplo, cada vértice em $\Gamma(\pi)$ possui uma aresta de entrada e uma aresta de saída, o que permite uma única decomposição em ciclos. Dessa forma, algumas definições para o grafo Γ são semelhantes às definições do grafo de ciclos:

1. Um k-ciclo em $\Gamma(\pi)$ é um ciclo que possui k arestas;
2. Um k-ciclo em $\Gamma(\pi)$ é par se k for par. Caso contrário, o k-ciclo é ímpar;
3. $c(\Gamma(\pi))$ representa o número de ciclos no grafo $\Gamma(\pi)$;
4. $c_{impar}(\Gamma(\pi))$ representa o número de ciclos ímpares no grafo $\Gamma(\pi)$;

A importância do grafo Γ reside principalmente no seguinte limitante superior, provado por Labarre [69].

Lema 1.6 *Para qualquer permutação* π, $d(\pi) \leq n - c_{impar}(\Gamma(\pi))$

O limitante inferior de Labarre pode ser melhorado usando circularidade. Seja π° uma permutação circular obtida de π pela adição do elemento 0, $\pi_0 = 0$, e seja $\pi^\circ_\circ$ o conjunto formado por todas as permutações $\sigma = (m\ \ m + \pi_1\ \ m + \pi_2\ \ \ldots\ \ m + \pi_n)\ (mod\ n + 1)$, para $m \in \mathbb{I}$. Eriksson e co-autores [47] mostraram que se $\sigma \in \pi^\circ_\circ$, então $d(\pi) = d(\sigma)$. Essa equivalência foi chamada de equivalência toroidal.

Labarre [69] usou as equivalências toroidais para derivar o Lema 1.7.

Lema 1.7 *Para qualquer permutação* π, $\pi \neq \iota$, $d(\pi) \leq n - \max_{\sigma \in \pi^{\circ}_{\circ}} c_{impar}(\Gamma(\sigma))$

Um novo limitante pode ser obtido usando a permutação reduzida π_{red}.

Lema 1.8 *Para qualquer permutação* π, $\pi \neq \iota$, $d(\pi) \leq m - \max_{\sigma \in (\pi_{red})^{\circ}_{\circ}} c_{impar}(\Gamma(\sigma))$, *onde* $|\pi_{red}| = m$.

1.1.2 Algoritmo de Bafna e Pevzner

Bafna e Pevzner [9] apresentaram um algoritmo que utiliza o grafo de ciclos apresentado na Definição 1.1. O algoritmo utiliza o grafo para prever o efeito de determinadas transposições no número de ciclos da permutação seguinte.

Dada uma permutação π, define-se um x-move como uma transposição ρ tal que $\Delta c(G(\pi), \rho) = x$, $x \in \{2{,}0, -2\}$. Um x-move é válido se $\Delta c(G(\pi), \rho) = \Delta c_{impar}(G(\pi), \rho)$. A Figura 1.4.a exibe um exemplo de um 2-move válido e a Figura 1.4.b exibe um exemplo de um 0-move válido.

Um 0-move ρ é bom se $\Delta c_{impar}(G(\pi), \rho) = 2$. A Figura 1.4.c exibe um exemplo de um 0-move bom. O 0-move bom é uma escolha melhor que um 0-move válido, já que a identidade possui apenas ciclos ímpares. Assim, é importante aumentar o número de ciclos ímpares mesmo que isso não aumente o número total de ciclos.

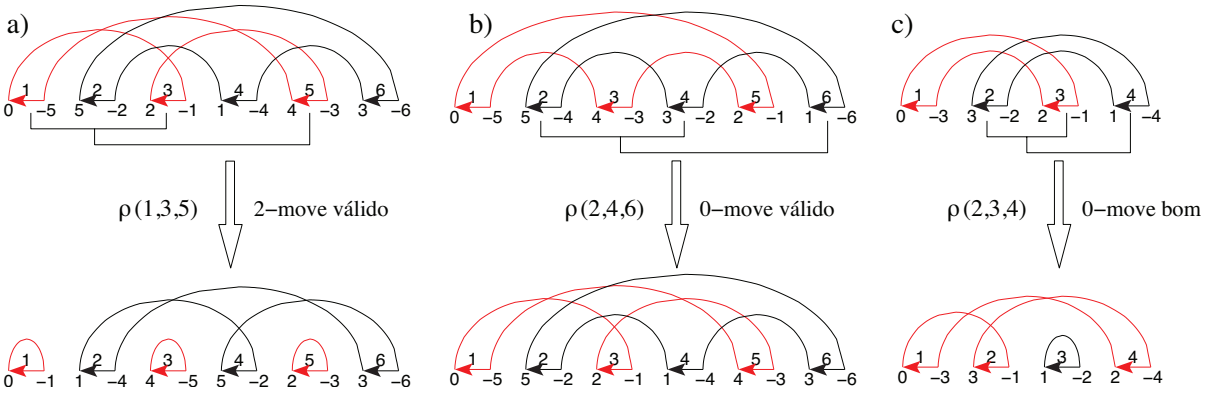


Figura 1.4: Exemplo de transposições: a) 2-move válido, b) 0-move válido e c) 0-move bom.

Para ordenar uma permutação usando o número mínimo de transposições, é necessário usar a maior quantidade possível de 2-moves válidos. Bafna e Pevzner estudaram a estrutura de ciclos que permitem 2-moves válidos e verificaram que, nos casos em que um 2-move válido não é possível, uma sequência contendo um 0-move e dois 2-move válidos pode ser aplicada.

Assim, seja (x, y)-sequence uma sequência de x transposições, tal que no mínimo y delas são 2-moves válidos. Bafna e Pevzner mostram que uma (3,2)-sequence é sempre possível quando nenhuma transposição ρ aplicável a π é um 2-move válido. A Figura 1.5.a mostra uma (4,3)-sequence e a Figura 1.5.b mostra uma (3,2)-sequence.

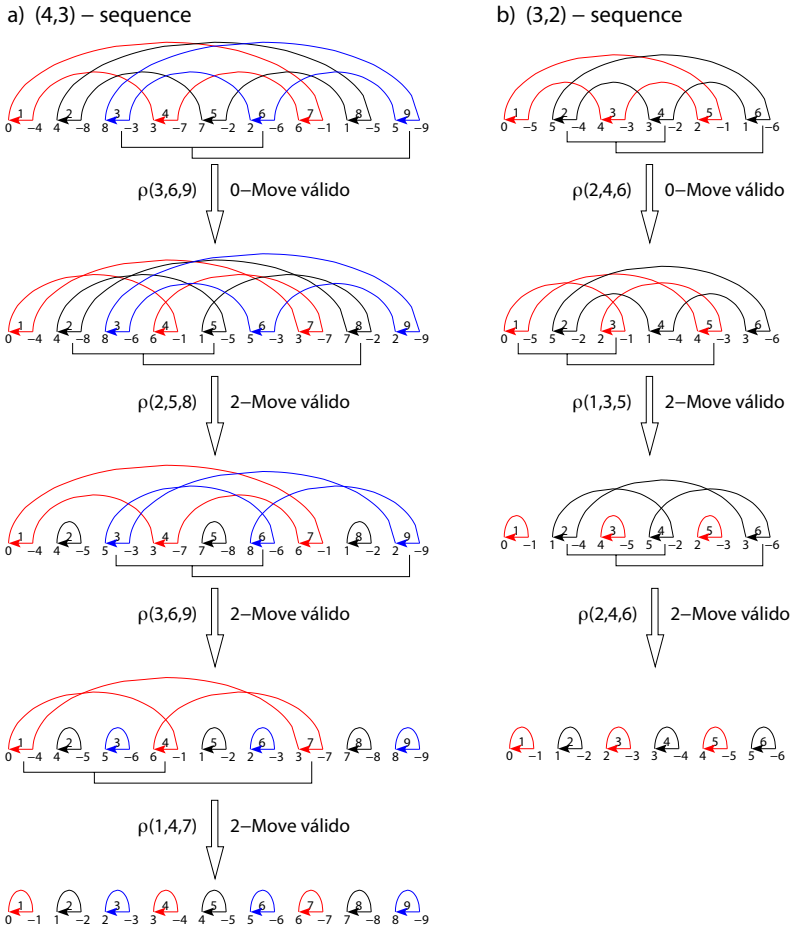


Figura 1.5: Exemplo de sequências: a) (4,3)-sequence, b) (3,2)-sequence.

Ciclos são classificados em dois grupos: orientados e não orientados. Um ciclo é orientado quando permite a aplicação de pelo menos um 2-move, caso contrário, o ciclo é não orientado. Para todo $k > 1$, um ciclo $C = (i_1, \ldots, i_k)$ é não orientado se, e somente se, $i_1, \ldots, i_k$ é uma sequência decrescente. Assim, a Figura 1.2 apresenta dois ciclos: $C_1 = (5,1,3)$ e $C_2 = (6,4,2)$. C_1 é um ciclo orientado e C_2 é um ciclo não orientado.

Duas sequências ordenadas de inteiros $V = (v_1, v_2, \ldots, v_k)$ e $W = (w_1, w_2, \ldots, w_k)$ estão entrelaçadas se $v_1 < w_1 < v_2 < w_2 < \ldots < v_k < w_k$ ou $w_1 < v_1 < w_2 < v_2 < \ldots < w_k < v_k$. Uma transposição $\rho(i, j, k)$ embaralha um tripla de arestas pretas (x, y, z) se (i, j, k) e (x, y, z) estão entrelaçados.

Por exemplo, na Figura 1.4.b, a transposição $\rho(2,4,6)$ embaralha as arestas pretas do ciclo (5,3,1) e cria o ciclo orientado (5,1,3). Generalizando, sejam (x, y, z) três arestas pretas de um ciclo não orientado com $x < y < z$, a transposição $\rho(i, j, k)$ que embaralha (x, y, z) gera um ciclo não orientado. Neste caso, após ρ, será possível aplicar o 2-move $\rho'(x', y', z')$, onde x', y' e z' são as novas posições de x, y e z. Bafna e Pevzner provam que o 2-move ρ' é válido.

Para identificar um 2-move válido, Bafna e Pevzner definem o Lema 1.9. Esse lema usa o conceito de distância entre duas arestas: sejam r e t duas arestas pretas no mesmo ciclo C, a distância $dist(r, t)$ é o número de arestas pretas em C que precisam ser atravessadas para ir de r a t.

Lema 1.9 *Dado um ciclo orientado C, se (x, y, z) é uma tripla orientada de C com no mínimo duas distâncias pares dentre $dist(x, y)$, $dist(z, x)$ ou $dist(y, z)$, então a transposição $\rho(y, z, x)$ é um 2-move válido.*

Um ciclo orientado é dito válido quando satisfaz o Lema 1.9 e é capaz de gerar um 2-move válido.

Uma classe de ciclos orientados válidos chamada de ciclos fortemente orientados foi definida por Bafna e Pevzner: seja $C = (i_1, i_2, \ldots, i_k)$ um ciclo orientado em $G(\pi)$ e seja C^* uma sequência de C em ordem decrescente. O ciclo C será fortemente orientado se existe uma transposição ρ que transforma C em C^*.

Um ciclo fortemente orientado C e um ciclo não orientado $C' = (j_1, \ldots, j_k)$ estão fortemente cruzados se o 2-move válido em C embaralha C'.

O Algoritmo 1 de aproximação 1.5 foi apresentado por Bafna e Pevzner usando os conceitos apresentados nesta seção. O algoritmo escolhe um ciclo qualquer no grafo de ciclos e usa as transposições que se aplicam a ele como. Por exemplo:

- Aplica-se dois 2-moves válidos se o ciclo é fortemente cruzado.
- Aplica-se um 2-move válido se o ciclo é fortemente orientado.
- Aplica-se um 2-move válido se o ciclo é orientado válido.
- Aplica-se uma (3,2)-sequence se o ciclo é orientado não válido.
- Aplica-se uma (3,2)-sequence se o ciclo é não orientado.

Algoritmo 1: Algoritmo de Bafna e Pevzner

```
Data: π
while π ≠ ι do
    if Existe k-ciclo C, k > 3, em G(π) then
        if C é um ciclo fortemente cruzado then
            Aplicar dois 2-moves válidos consecutivos
        end
        else
            if C é um ciclo fortemente orientado then
                Aplicar um 2-move válido
            end
            else
                if C é um ciclo orientado then
                    Aplicar um 2-move válido ou uma (3,2)-sequence
                end
                else
                    Aplicar uma (3,2)-sequence
                end
            end
        end
    end
    else
        Aplicar 0-move bom seguido de 2-move válido
    end
end
```

1.1.3 Algoritmo de Elias e Hartman

Elias e Hartman observaram que todo ciclo orientado de tamanho 3 é válido [46]. Portanto, se uma permutação π gerar um grafo $G(\pi)$ contendo apenas k-ciclos com $k \leq 3$, então a distinção entre 2-move e 2-move válido passa a ser desnecessária.

Uma permutação π é dita simples se todos os ciclos de $G(\pi)$ possuem tamanho menor ou igual a 3. O primeiro passo no algoritmo de Elias e Hartman consiste em transformar a permutação π de entrada em uma permutação π' simples, tal que π e π' possuam o mesmo limitante inferior mostrado no Lema 1.4. Vale ressaltar que a transformação não garante $d(\pi) = d(\pi')$.

O procedimento para transformar uma permutação arbitrária em uma permutação simples consiste em adicionar novos elementos de modo a quebrar k-ciclos longos, que são aqueles com $k > 3$.

- Sejam:

    1. C um k-ciclo longo em $G(\pi)$;
    2. b_1, b_2 e b_3 arestas pretas em C tais que b_2 e b_3 estão conectadas a b_1 com uma aresta cinza;

3. g uma aresta cinza conectada a b_2, mas não a b_1.

- Então, o seguinte procedimento particiona o k-ciclo C em um ciclo C_1 de tamanho 3 e em um ciclo C_2 de tamanho $k-2$. A Figura 1.6 ilustra o procedimento.

1. Remover a aresta preta b_3 e a aresta cinza g;
2. Assumir $b_3 = (v_b, w_b)$ e $g = (w_g, v_g)$;
3. Criar as arestas pretas (v_b, v) e (w, w_b);
4. Criar as arestas cinzas (w_g, w) e (v, v_g).

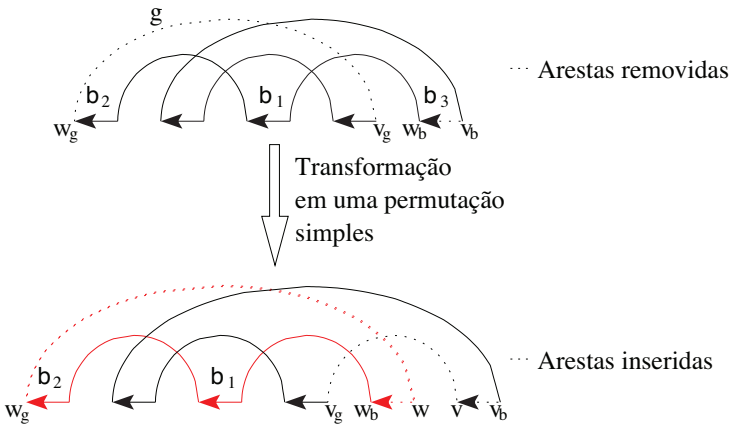


Figura 1.6: Procedimento para transformar um k-ciclo longo em dois ciclos: o primeiro ciclo terá tamanho 3 e o segundo ciclo terá tamanho $k-2$.

O procedimento acima será usado iterativamente sempre que um ciclo longo for encontrado. Os outros passos do algoritmo de Elias e Hartman consistem em aumentar o número de ciclos até $n+1$, estratégia análoga a usada por Bafna e Pevzner. A principal diferença entre ambos os algoritmos ocorrerá quando não houver ciclos orientados no grafo de ciclo. Nesses casos, Elias e Hartman provaram que, se o grafo possui mais de 8 ciclos, então é sempre possível aplicar uma sequência de transposições que garante a razão de aproximação 1.375.

Uma (x, y)-sequence garante a razão de aproximação 1.375 se $\frac{x}{y} \leq \frac{11}{8}$. A Figura 1.5.a é um exemplo de sequência que garante a razão 1.375.

Elias e Hartman baseiam a prova formal na enumeração de todas as configurações não orientadas. A enumeração foi feita com o suporte de um programa que realiza a listagem. O programa determina um conjunto finito de configurações e insere em um banco de dados. O banco de dados é gerado de modo a garantir que, para qualquer grafo com mais de 8 ciclos, sempre existe um subconjunto de ciclos no banco de dados.

1.2 Reversão

Eventos de reversão ocorrem quando um bloco é destacado do genoma e inserido no mesmo lugar, mas com a ordem e a orientação invertidas. No caso da reversão, sinais são associados aos identificadores na representação formal do genoma como uma permutação: $\pi = (\pm\pi_1 \ \pm\pi_2 \ \ldots \ \pm\pi_n)$, $\pi_i \in \mathbb{N}$, $0 < \pi_i \leq n$ e $i \neq j \leftrightarrow \pi_i \neq \pi_j$.

Um problema parecido com a reversão é chamado de reversão sem sinal. Neste problema, uma reversão inverte apenas a ordem da permutação e não a sua orientação. Neste caso, não há a necessidade de associar sinais aos identificadores e a representação formal da permutação passa a ser: $\pi = (\pi_1 \ \pi_2 \ \ldots \ \pi_n)$, $\pi_i \in \mathbb{N}$, $0 < \pi_i \leq n$ e $i \neq j \leftrightarrow \pi_i \neq \pi_j$. O problema da reversão sem sinal não possui significado biológico, mas é interessante como um problema teórico em ciência da computação.

Reversões e reversões sem sinal são as classe de rearranjo que possuem os melhores resultados conhecidos. Um estudo inicial foi apresentado em 1993 por Bafna e Pevzner [6], que criaram um algoritmo de aproximação com razão de aproximação 1.5 para reversões e 1.75 para reversões sem sinal.

O primeiro algoritmo polinomial para o problema da distância de reversão foi apresentado por Hannenhalli e Pevzner [56]. Posteriormente, a estratégia de Hannenhalli e Pevzner foi simplificada por Bergeron [12]. Atualmente, já existe um algoritmo com complexidade sub-quadrática [92] e, quando apenas a distância é necessária, um algoritmo linear pode ser usado [5].

O problema de reversão sem sinal, por outro lado, é NP-Difícil [20]. O melhor algoritmo de aproximação foi apresentado por Berman, Hannenhalli e Karpinski em 2002 [13]. Este algoritmo possui fator de aproximação 1.375.

Algumas versões mais restritas do problema da reversão já receberam tratamento algorítmico apropriado. Dentre essas versões, é possível citar o problema das reversões de prefixo [25, 53, 60], e o problema das reversões de tamanho fixo [24].

Em 2005, Ohlebusch e co-autores [82] produziram um algoritmo de ordenação para um problema inspirado em reversões. Neste problema, as reversões possuem uma simetria em relação a um ponto da permutação chamado de origem. O algoritmo produz resultado ótimo em tempo polinomial. Entretanto, o problema definido pelos autores não garante que, dados dois genomas arbitrários, seja sempre possível obter uma sequência de reversões simétricas que transforme um genoma no outro. Dessa forma, o algoritmo só pode ser usado para um grupo restrito de instâncias.

A Seção 1.2.1 apresenta uma introdução à reversões simétricas, valorizando a importância desta nova classe de reversões em genomas bacteriais.

1.2.1 Reversão Simétrica

A análise comparativa de uma série de genomas bacteriais da família *Pseudomonadaceae* e dos grupos *Xanthomonas*, *Shewanella* e *Mycobacterium* permite observar que os eventos de reversão que ocorrem nesses organismos possuem liberdade menor do que o permitido pela definição convencional de reversão. O principal indício que suporta esse argumento é obtido ao alinhar dois genomas desses grupos bacteriais em um plano cartesiano, o gráfico resultante permite observar um padrão em "X" formado pelas sequências conservadas de DNA. A característica mais importante do padrão em "X" é a simetria em relação à origem de replicação.

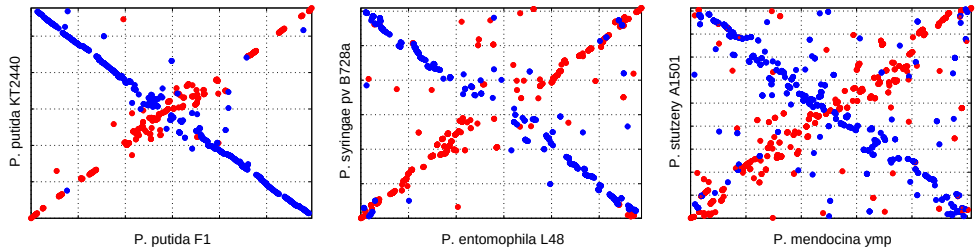
A Figura 1.7 mostra o alinhamento dos genomas de alguns dos organismos usando o programa `nucmer` do pacote MUMmer [68]. Os genomas completos foram posicionados nas ordenadas e abscissas do gráfico com as origens de replicação na coordenada (0,0). Os alinhamentos mostram com nitidez o padrão em "X" formado por regiões marcadas com a cor vermelha e regiões marcadas com a cor azul; a cor vermelha foi utilizada para identificar regiões conservadas com a mesma orientação em ambos os genomas, enquanto que a cor azul corresponde a regiões conservadas com orientações opostas.

O padrão em "X" foi primeiramente observado por Eisen e co-autores [45] ao comparar genomas completos de *Vibrio cholerae* com *Escherichia coli*, *Streptococcus pneumoniae* com *Streptococcus pyogenes* e *Mycobacterium tuberculosis* com *Mycobacterium leprae*. Eles desenvolveram um modelo estatístico para calcular a probabilidade de o padrão em "X" ocorrer ao acaso em genomas de organismos com pouca proximidade evolutiva.

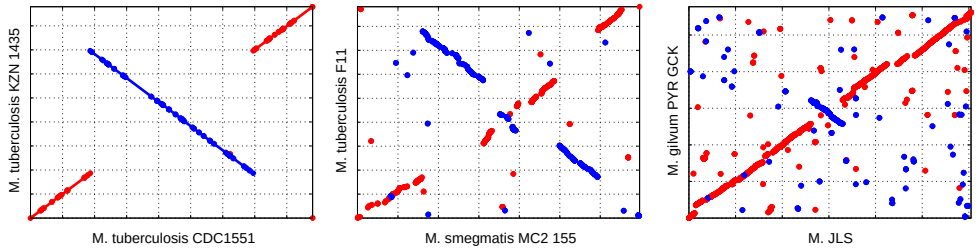
A análise estatística parte do pressuposto de que, para organismos pouco relacionados, a distribuição das regiões conservadas no gráfico cartesiano seguiria uma distribuição uniforme. Dessa forma, a densidade esperada de regiões conservadas em qualquer sub-área do gráfico cartesiano é igual à proporção desta área em relação à área total do gráfico. Assim, dado um gráfico cartesiano com um total de N pontos, a probabilidade de encontrar no mínimo m pontos em uma sub-área é dada por $Pr(N, m, p) = \sum_{x=m}^{N} \binom{N}{x} p^x (1-p)^{(N-x)}$, onde p indica a proporção da sub-área em relação à área total do gráfico.

A equação fornece um *P-value* capaz de confirmar a significância estatística do padrão em "X". Para tanto, é necessário selecionar um área onde o padrão em "X" esteja contido. Eisen e co-autores utilizaram uma área contendo 10% da área total do gráfico ao longo da diagonal $y = x$ e 10% da área total do gráfico ao longo da diagonal $y = L - x$, onde L é o tamanho do genoma.

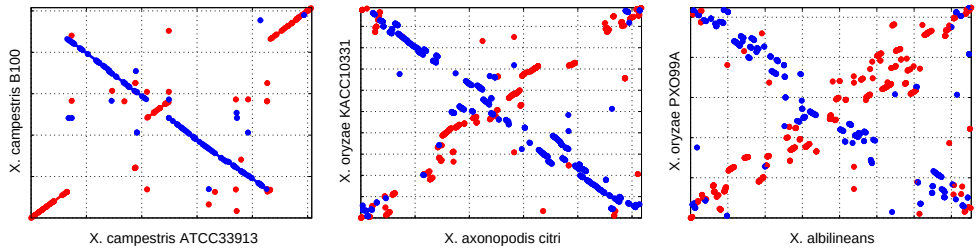
Entretanto, este método é ineficaz com os organismos das classes acima mencionadas, pois a ocorrência de deleções (principalmente as ocorridas próximo à origem de replicação) afastou o padrão em "X" da diagonal do plano. Para solucionar este problema, duas retas r e s foram calculadas com o intuito de substituírem as diagonais. As retas foram calculadas com os seguintes passos:



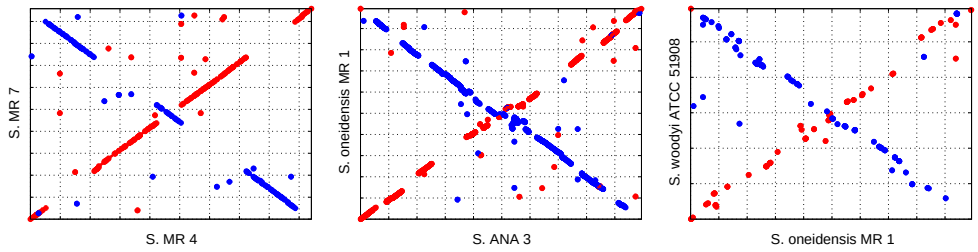
(a) Família *Pseudomonadaceae*



(b) Grupo *Mycobacterium*



(c) Grupo *Xanthomonas*



(d) Grupo *Shewanella*

Figura 1.7: Alinhamento de genomas usando a ferramenta MUMmer [68]. O alinhamento apresenta com nitidez o padrão em "X" frequentemente observado em bactérias. Neste gráfico, as regiões em vermelho correspondem a regiões conservadas (*matches*) que ocorrem na mesma direção em ambos os genomas, enquanto que as regiões em azul correspondem a regiões conservadas em direções opostas.

1. Obter a subsequência crescente mais longa dos pontos vermelhos $v_1, v_2, \ldots, v_n$ e a subsequência decrescente mais longa dos pontos azuis $a_1, a_2, \ldots, a_n$. Este passo é resolvido em tempo $O(n \log n)$, onde n é o tamanho da sequência de entrada [88].

2. Definir a reta r que passa pelos pontos $v_1, v_2, \ldots, v_n$ e a reta s que passa pelos pontos $a_1, a_2, \ldots, a_n$. Ambas as retas foram calculadas pelo método dos mínimos quadrados.

Esta estratégia garantiu, em todos os alinhamentos par-a-par, o padrão em "X" contido em uma área de aproximadamente 20% do gráfico, sendo 10% ao redor de r e 10% ao redor de s. A significância estatística foi calculada em todos os gráficos cartesianos e um sumário pode ser observado na Tabela 1.1.

	Diagonal Principal			
	Mínimo	Máximo	Média	Desvio Padrão
Pseudomonadaceae	0.0	$2 * 10^{-26}$	$1 * 10^{-28}$	$2 * 10^{-27}$
Mycobacterium	0.0	$8 * 10^{-4}$	$8 * 10^{-6}$	$7 * 10^{-5}$
Shewanella	0.0	$1 * 10^{-10}$	$5 * 10^{-13}$	$7 * 10^{-12}$
Xanthomonas	0.0	$2 * 10^{-132}$	$6 * 10^{-134}$	$6 * 10^{-133}$

Tabela 1.1: Significância Estatística

A partir desses resultados é possível concluir que o padrão em "X" não ocorre ao acaso. O grupo das *Xanthomonas* é aquele em que o padrão em "X" é mais acentuado. O segundo grupo com padrão em "X" mais acentuado é a família *Pseudomonadaceae*.

Eisen e co-autores apresentaram um modelo evolutivo capaz de explicar o padrão em "X". Esse modelo usa um tipo restrito de reversões em que os dois pontos onde o genoma é quebrado (*breakpoints*) são igualmente distantes da origem de replicação. A Figura 1.8 ilustra como uma série de reversões simétricas à origem de replicação pode gerar o padrão em "X" similar ao observado em genomas reais.

A relação entre os eventos de rearranjo e a origem de replicação tem recebido explicações baseadas na natureza da replicação do DNA em cromossomos circulares [29]. A replicação inicia quando a enzima DNA polimerase se liga à origem de replicação e duas holoenzimas copiam o cromossomo circular em direções opostas.

Cada holoenzima copia aproximadamente metade do cromossomo e, eventualmente, se encontram em uma região conhecida como término da replicação. Supõe-se que esse mecanismo favoreça a ocorrência de reversões simétricas, principalmente quando em ambos os lados do genoma existem elementos repetidos orientados em sentidos opostos [1, 62].

Dado que o padrão em "X" seria completamente destruído como resultado de apenas uma reversão fortemente assimétrica, a simples presença de padrões nítidos é um indício de que a assimetria não seja muito frequente.

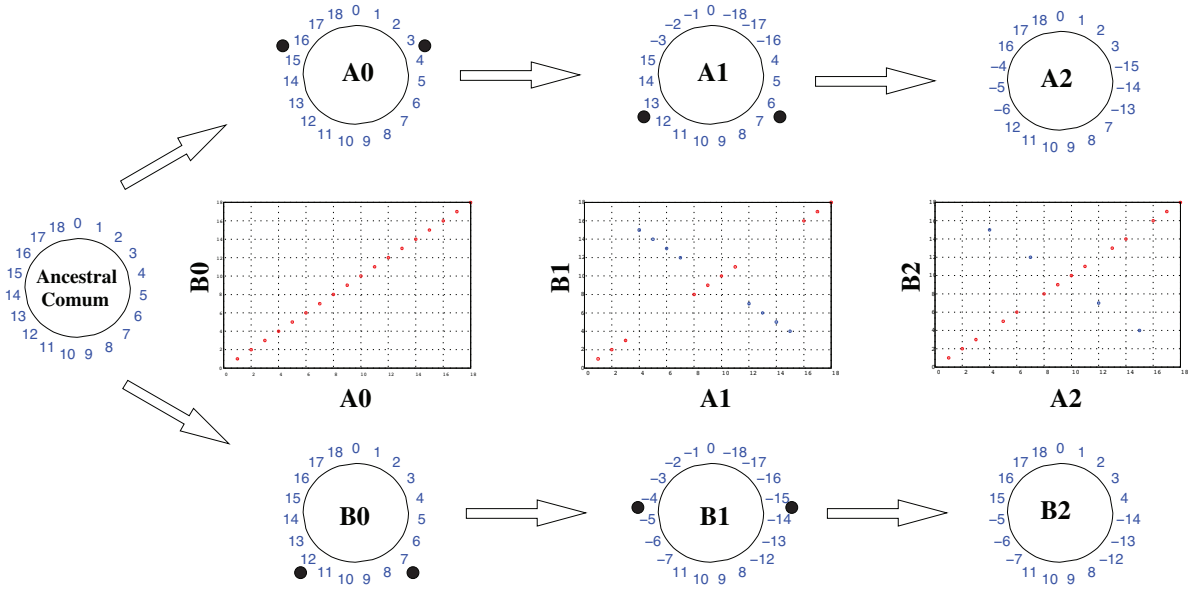


Figura 1.8: Modelo de evolução proposto por Eisen e co-autores [45] usando reversões simétricas à origem de replicação. Um gráfico do alinhamento foi criado para cada geração de modo a ilustrar a evolução dos genomas. Os números 1-18 nos cromossomos se referem a segmentos de DNA compartilhados entre os genomas.

Parte II

Transposições

Capítulo 2

Heurísticas para o Problema da Distância de Transposição

Motivação: O problema da distância de transposição é um dos mais conhecidos em rearranjo de genomas. Recentemente, Bulteau, Fertin e Rusu [16] provaram que o mesmo é NP-difícil. Vários algoritmos foram criados para o problema, a maioria com o propósito de provar algum fator de aproximação, ou apresentar nova estrutura de dados com características próprias para o problema [9, 27, 46, 58, 76, 96, 97]. Na prática, uma análise mostra que a resposta fornecida por esses algoritmos se afasta da distância real em uma grande quantidade de casos, mesmo para instâncias pequenas do problema.

Metodologia: O desenvolvimento de uma série de heurísticas baseadas no grafo de ciclos permitiu a criação de um novo algoritmo para o problema. Esse algoritmo usa a abordagem de Bafna e Pevzner [9] como ponto de partida e acopla novas regras que permitem aumentar o número de ciclos ímpares com menos transposições. Além disso, uma análise sobre o algoritmo de Elias e Hartman [46] permite inserir no algoritmo apresentado neste capítulo um passo que garante o fator de aproximação 1.375.

Resultados: O algoritmo desenvolvido neste capítulo foi comparado com vários outros. A análise mostra que, na prática, os resultados são superiores com permutações pequenas (menores que 11) e com permutações maiores (uma análise foi feita com 1000 permutações de tamanhos entre 10 e 100). Uma das versões do algoritmo ordena qualquer permutação π, para $|\pi| \leq 9$, com o número mínimo de transposições.

2.1 Introdução

O algoritmo de Bafna e Pevzner [9] apresentado na Seção 1.1.2 foi o primeiro algoritmo de aproximação para o problema da distância de transposição. O propósito dos autores era provar a razão de aproximação 1.5, havendo pouca preocupação com a qualidade dos resultados na prática.

Esta seção introduz uma série de heurísticas que permitem resultados superiores em uma análise prática. O ponto de partida para essas heurísticas é o grafo de ciclos (Seção 1.1.1), estrutura de dados do algoritmo de Bafna e Pevzner, o que possibilita a união das heurísticas com o algoritmo original de Bafna e Pevzner para gerar um novo algoritmo de melhor performance.

Posteriormente ao trabalho de Bafna e Pevzner, Elias e Hartman [46] introduziram um método para obter a razão de aproximação 1.375. Este método é também baseado no grafo de ciclos, permitindo analisá-lo juntamente com as heurísticas desenvolvidas neste capítulo.

O capítulo é estruturado da seguinte forma. A Seção 2.2 propõe as heurísticas. A Seção 2.3 cria um novo algoritmo baseado nessas heurísticas. A Seção 2.4 examina o tempo computacional da heurística menos eficiente. A Seção 2.5 compara o algoritmo proposto com outros algoritmos encontrados na literatura.

2.2 Heurísticas

O objetivo das heurísticas é diminuir rapidamente o número de ciclos ímpares no grafo de ciclos, levando a uma estratégia gulosa em que um 2-move válido é sempre o mais vantajoso, pois se aproxima da identidade tanto na quantidade de ciclos ímpares quanto na quantidade total de ciclos.

Um 0-move bom aumenta o número de ciclos ímpares, apesar de não modificar o número total de ciclos. Bafna e Pevzner priorizam 0-moves válidos em detrimento de 0-moves bons, ignorando o fato de que um 0-move bom é melhor que um 0-move válido na maioria dos casos. Bafna e Pevzner aplicam 0-moves bons apenas quando nenhuma outra transposição está disponível.

Caso não haja 2-move válido ou 0-move bom disponíveis, resta a opção de aplicar um 0-move válido. Entretanto, essa transposição deve gerar uma (4,3)-sequence para garantir o fator de aproximação 1.375.

O modo de ordenar permutações com o menor número de transposições é fazendo cada transposição responsável pelo aumento do número de ciclos ímpares e por gerar uma condição favorável para o aumento do número de ciclos ímpares na transposição seguinte. Para isso acontecer, os seguintes fatores são importantes.

- Presença de ciclos orientados válidos.
- Presença de longos ciclos pares.

Ciclos orientados válidos geram 2-moves válidos. Se uma transposição gera um ciclo orientado válido, então a transposição seguinte será um 2-move válido.

A presença de arestas pretas em ciclos pares é necessária para a ocorrência de 0-moves bons. O Lema 2.1 permite identificar 0-moves bons.

Lema 2.1 *Sejam C e D dois ciclos pares em $G(\pi)$, uma transposição é um 0-move bom se ela age na aresta preta d_l de D e nas arestas pretas c_m e c_n de C, tal que $dist(c_m, c_n)$ é ímpar.*

Prova Uma transposição agindo nas arestas pretas c_m, c_n e d_l cria dois ciclos C' e D', onde $|C'| = |C| - dist(c_m, c_n)$ e $|D'| = |D| + dist(c_m, c_n)$. Se $dist(c_m, c_n)$ é ímpar, então $|C'|$ e $|D'|$ são ímpares. Assim, a transposição é um 0-move bom. ■

Uma série de heurísticas são descritas a seguir. As heurísticas estão ordenadas de acordo com a prioridade, as mais importantes são explicadas primeiro. Essa ordem de prioridade leva aos melhores resultados na prática, fechando uma lacuna no algoritmo de Bafna e Pevzner, que, por exemplo, dado um grafo de ciclos com um ciclo orientado C e um ciclo fortemente orientado D, não era muito preciso em afirmar se é mais recomendável usar o 2-move válido em C antes de usar o 2-move válido no ciclo D.

Heurística 1 *Seja $\rho(i, j, k)$ uma transposição que aumenta o número de ciclos ímpares em $G(\pi)$, então existe 2-move válido em $\pi\rho(i, j, k)$ se (i, j, k) está entrelaçado com três arestas pretas (a, b, c) em um ciclo não orientado.*

Heurística 2 *Seja $\rho(i, j, k)$ uma transposição que aumenta o número de ciclos ímpares em $G(\pi)$, então existe 2-move válido em $\pi\rho(i, j, k)$ se (i, j, k) transforma um ciclo orientado não válido em um ciclo orientado válido.*

A Heurística 2 requer a conversão de um ciclo orientado não válido em um ciclo orientado válido. Os Lemas 2.2 e 2.3 indicam como essa conversão pode ocorrer.

Lema 2.2 *Seja $C = (x, \ldots, y, \ldots, a, z, b, \ldots)$ um ciclo orientado que não permite um 2-move válido. A transposição $\rho(i, j, k)$, tal que (i, j, k) entrelaça com (b, z, x), gera um ciclo orientado C' que permite um 2-move válido.*

Prova Se C não permite um 2-move válido, então $a < b < y < z < x$ e $dist(b, x)$ é ímpar [9], caso contrário $\rho(z, b, y)$ seria um 2-move válido, contradizendo a premissa. As situações seguintes podem ocorrer se (i, j, k) e (b, z, x) estão entrelaçados.

1. Se $i < b < j < z < k < x$, então a transposição $\rho(i, j, k)$ gera o ciclo $C' = (x', \ldots, y', \ldots, a', z',\ b', \ldots)$ com $z' < b' < x'$. Assim, a transposição $\rho(z', b', x')$ é um 2-move válido em $\pi\rho(i, j, k)$, pois (x', z', b') é uma tripla orientada e $dist(z', b')$ e $dist(b', x')$ são ímpares (Lema 1.9).

2. Se $b < i < z < j < x < k$, então a transposição $\rho(i, j, k)$ gera o ciclo $C' = (z', b', \ldots, x', \ldots, y', \ldots a')$ com $b' < x' < z'$. Assim, a transposição $\rho(b', x', z')$ é um 2-move válido em $\pi\rho(i, j, k)$, pois (z', b', x') é uma tripla orientada e $dist(z', b')$ e $dist(b', x')$ são ímpares (Lema 1.9).

■

Lema 2.3 *Seja $C = (x, \ldots, y, \ldots, a, z, b, \ldots)$ um ciclo orientado que não permite um 2-move válido. A transposição $\rho(i, j, k)$, tal que $i < a < j < b < k < x$, gera um ciclo orientado C' que permite um 2-move válido.*

Prova Dois casos são possíveis:

1. Se $k < z$, então $\rho(i, j, k)$ gera o ciclo $C' = (x', \ldots, y',\ \ldots, a', z', b', \ldots)$ com $b' < a' < z'$. Assim, a transposição $\rho(b', a', z')$ é um 2-move válido, pois (z', b', a') é uma tripla orientada e $dist(a', z')$ e $dist(b', z')$ são ímpares (Lema 1.9).

2. Se $z < k < x$, então $\rho(i, j, k)$ gera o ciclo $C' = (x', \ldots, y', \ldots, a', z', b', \ldots)$ com $b' < z' < a'$ e $y' < a' < b'$. Assim, a transposição $\rho(y', a', x')$ é um 2-move válido, pois (x', y', a') é uma tripla orientada e $dist(x', y')$ e $dist(y', a')$ são ímpares (Lema 1.9).

■

Heurística 3 *Seja $\rho(i, j, k)$ um 2-move válido, os tamanhos dos novos ciclos gerados por ρ estão no conjunto $\{dist(i, j), dist(j, k), dist(k, i)\}$, sendo que, por definição, no máximo uma das distâncias do conjunto pode ser par. Dessa forma, para garantir a criação do maior ciclo par possível, deve-se verificar qual transposição possui a maior distância par.*

A Heurística 3 permite maximizar as possibilidades de 0-moves bons para o próximo passo (Lema 2.1).

Heurística 4 *Dados a permutação de entrada π e um parâmetro t pré-definido, obtém-se todas as possíveis permutações após t transposições que podem ser 2-move válidos ou 0-moves bons. Assim, seja m o número de permutações possíveis, o conjunto de permutações é da forma:*

$$\pi^1 = \pi\rho_1^1\rho_2^1 \dots \rho_t^1$$
$$\pi^2 = \pi\rho_1^2\rho_2^2 \dots \rho_t^2$$
$$\dots$$
$$\pi^m = \pi\rho_1^m\rho_2^m \dots \rho_t^m$$

As permutações $\pi^1, \pi^2, \dots, \pi^m$ são ordenadas de acordo com a possibilidade de aplicar heurísticas. Por exemplo, uma permutação π^i que permite a aplicação da Heurística 1 é melhor que uma permutação π^j que permite apenas a aplicação da Heurística 2, dessa forma, $i < j$. Ao final desse processo, seja π^1 a melhor permutação, escolhe-se a transposição ρ_1^1.

A Heurística 4 avalia as transposições conforme as configurações possíveis após um certo número de passos. Essa heurística é uma busca em largura limitada a um certo nível t, que será chamado de *look-ahead*. Esse passo foi gerado porque, em muitos casos, é difícil prever os efeitos de uma transposição em $G(\pi)$ sem propriamente executá-la.

A Figura 2.1 ilustra o *look-ahead* aplicado na permutação $\pi = (3\ 2\ 1\ 5\ 4)$. O grafo de ciclo não contém 2-move válido. Nesse caso, a busca será executada em 0-moves bons, havendo um total de 4 disponíveis: $\rho_a(1,3,5)$, $\rho_b(1,5,7)$, $\rho_c(3,5,7)$ e $\rho_d(1,5,7)$. As transposições ρ_a e ρ_c geram permutações que permitem a Heurística 1, o que garantiria dois 2-moves válidos em sequência. Por outro lado, as permutações ρ_b e ρ_d geram permutações sem ciclos orientados, pior cenário possível por não permitir transposições que aumentam o número de ciclos ímpares.

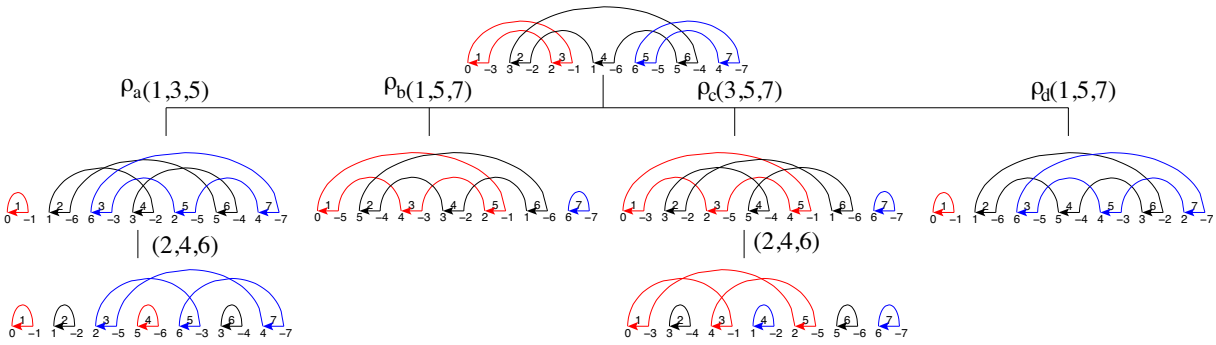


Figura 2.1: Exemplo de *look-ahead* na permutação $\pi = (3\ 2\ 1\ 5\ 4)$.

Nesse caso, usando apenas um *look-ahead*, concluiria-se que ρ_a e ρ_c são semelhantes em prioridade. Em alguns casos, um segundo *look-ahead* poderia distinguir entre as duas transposições, mas esse não é o caso da Figura 2.1.

A busca em largura é muito onerosa computacionalmente, fazendo dessa heurística uma opção apenas quando o tempo de resposta não é um fator determinante. A Seção 2.4 realiza uma análise sobre essa heurística.

Heurística 5 *Seja $E_{cd}(\pi)$ o conjunto de arestas cinzas $(+(i-1), -i)$ direcionadas para a direita em $G(\pi)$, tais que $\pi^{-1}(i-1) < \pi^{-1}(i)$, a permutação identidade é a única em que $E_{cd}(\pi) = E_c(\pi)$, onde $E_c(\pi)$ é o conjunto de todas as arestas cinzas em $G(\pi)$. Assim, escolhe-se a transposição que mais aumenta o número de arestas em E_{cd} de forma a se aproximar da identidade.*

Calcular o número de arestas cinzas em $E_{cd}(\pi')$, onde $\pi' = \pi\rho(a, b, c)$, é trivial. Sejam $E_{entrada}$ o conjunto das arestas e tais que $e \in E_{cd}(\pi'), e \notin E_{cd}(\pi)$ e E_{saida} o conjunto das arestas e tais que $e \in E_{cd}(\pi), e \notin E_{cd}(\pi')$:

- $E_{entrada} = \{+(i-1), -i | 1 \leq i \leq n+1, \pi^{-1}(i-1) > \pi^{-1}(i), a \leq \pi^{-1}(i) < b \leq \pi^{-1}(i-1) < c\}$
- $E_{saida} = \{+(i-1), -i | 1 \leq i \leq n+1, \pi^{-1}(i-1) > \pi^{-1}(i), a \leq \pi^{-1}(i-1) < b \leq \pi^{-1}(i) < c\}$

Dessa forma, para calcular a quantidade de elementos em $E_{cd}(\pi')$ usa-se a equação: $|E_{cd}(\pi')| = |E_{cd}(\pi)| + E_{entrada} - E_{saida}$.

Heurística 6 *Quando não há transposições capazes de aumentar o número de ciclos ímpares, é possível garantir uma (4,3)-sequence se no mínimo dois ciclos orientados existirem.*

A Heurística 6 atende o caso em que dois ciclos orientados não válidos existem. Os Lemas 2.4 e 2.5 fundamentam essa heurística.

Lema 2.4 *Sejam dois ciclos orientados $C = (x, \ldots, y, \ldots, a, z, b, \ldots)$ e D que não permitem um 2-move válido, se $\rho(y, z, x)$ transforma D em um ciclo orientado C' que permite um 2-move válido (Lemas 2.2 e 2.3), então ρ leva a uma (4,3)-sequence.*

Lema 2.5 *Se há dois ciclos orientados $C = (x_c, \ldots, y_c, \ldots, a_c, z_c, b_c, \ldots)$ e $D = (x_d, \ldots, y_d, \ldots, a_d, z_d, b_d \ldots)$ e o Lema 2.4 não é possível, então, sem perda de generalidade, assuma $a_c < a_d$. A transposição $\rho(a_c, a_d, z_d)$ leva a uma (4,3)-sequence.*

Prova A prova dos Lemas 2.4 e 2.5 foi realizada com o auxílio de um programa responsável por listar todas as configurações possíveis para C e D. Ao todo, 126 configurações foram encontradas pela análise. A lista completa de configurações está disponível online [34]. ■

Heurística 7 *Se $G(\pi)$ possui no máximo* 1 *ciclo orientado não válido e um conjunto de ciclos orientados ímpares, então deve-se transformar a permutação de entrada em uma permutação simples e usar o algoritmo de Elias e Hartman, que garante uma razão de aproximação* 1.375*.*

A Heurística 7 inicia com a conversão de π em uma permutação simples π'. Essa conversão gerará $G(\pi')$ com vários ciclos não orientados de tamanho 3. O próximo passo consiste em selecionar um subconjunto de 3-ciclos de $G(\pi')$ em um banco de dados contendo as transposições aplicáveis.

Esse banco de dados foi compilado a partir da prova formal de Elias e Hartman para a razão de aproximação 1.375. O banco de dados é particionado de acordo com o número de elementos presentes em cada configuração. A Tabela 2.1 mostra o número de configurações presentes no banco de dados por número de elementos.

Elementos	Configurações
9	22
12	5.681
15	53.895
18	377.874
21	1.450.662
24	1.071.555
27	1.032.969

Tabela 2.1: Número de configurações presentes no banco de dados por número de elementos.

A pesquisa no banco de dados ocorre da seguinte forma: seleciona-se 3 ciclos em $G(\pi')$ e verifica-se a presença da configuração gerada por eles no banco de dados com 9 elementos. Caso não exista a configuração, adiciona-se um novo 3-ciclo de $G(\pi')$ e retoma-se a busca no conjunto de 12 elementos. Essa busca termina quando algum subconjunto de $G(\pi')$ é encontrado no banco de dados e as transposição indicadas pelo banco são aplicadas a π'.

Vale ressaltar que todas as configurações presentes no banco de dados possuem de 9 a 27 elementos, isso ocorre porque a prova formal de Elias e Hartman mostra que esse conjunto é suficiente, pois toda configuração de 3-ciclos com mais de 27 elementos possui um subconjunto no banco de dados.

2.3 Algoritmo

As heurísticas foram agrupadas para gerar o algoritmo desta seção. A Seção 2.2 ressaltou a importância dos 2-moves válidos e dos 0-moves bons, sendo que os primeiros são preferíveis em relação aos segundos.

A Heurística 1 está relacionada com transposições entrelaçadas a ciclos não orientados. Para cada ciclo orientado válido C em $G(\pi)$, obtém-se os 2-moves válidos. Este passo pode ser obtido em $O(n^2)$. Após isso, busca-se um ciclo não orientado D com uma tripla (x, y, z) entrelaçada com algum 2-move válido listado no passo anterior. O tempo total de processamento é da ordem de $O(n^3)$.

Supondo a ausência de 2-moves válidos entrelaçados com ciclos não orientados, então a Heurística 2 sugere procurar ciclos orientados não válidos. Para cada 2-move válido $\rho(i, j, k)$, é preciso verificar se há um ciclo orientado não válido satisfazendo os Lemas 2.2 ou 2.3. Este passo pode ser realizado em $O(n^2)$.

A próxima heurística, Heurística 3, sugere gerar o maior ciclo par possível. Para tanto, basta selecionar o ciclo com maior distância par. Este passo é realizado em $O(n)$.

A Heurística 4 é de uso opcional. Nesta heurística, uma busca em largura é iniciada na tentativa de melhorar a qualidade da solução fornecida pelo algoritmo. Essa heurística é a mais onerosa computacionalmente, a complexidade de tempo é da ordem de $O(n^{3(t+1)})$, onde t é o número de *look-aheads*. Para não utilizar esta heurística, basta configurar $t = 0$.

A Heurística 5 resulta de uma observação pertinente à grande maioria das instâncias reais de tamanho menor ou igual a 10 quando as heurísticas anteriores não são possíveis. Aumentar a quantidade de arestas direcionadas da esquerda para a direita no grafo de ciclos (conjunto $E_{cd}(\pi)$) permite um menor número de transposições para ordenar as permutações. O passo é realizado em $O(n^2)$.

Caso não exista 2-moves válidos, tenta-se aplicar as Heurísticas 1, 2 e 5 com 0-moves bons. A Heurística 3 é específica para 2-moves válidos, então não pode ser usada nesta etapa.

Quando não existem transposições aumentando o número de ciclos ímpares, então a Heurística 6 sugere usar uma (4,3)-sequence se dois ciclos orientados não válidos existirem.

Por fim, caso nenhuma das heurísticas anteriores seja possível, a Heurística 7 sugere usar a estratégia de Elias e Hartman [46] para garantir a razão de aproximação 1.375.

A complexidade final do algoritmo é $O(n^{4+3t})$. O Algoritmo 2 mostra o pseudocódigo completo do novo algoritmo.

2.4 Análise do *Look-Ahead*

A Heurística 4 foi inserida no algoritmo por ser difícil prever o efeito de uma transposição. Entretanto, o efeito que a estratégia tem na qualidade da solução deve ser medido. Em essência, o *look-ahead* gera uma estratégia de busca exaustiva, o que causa impacto no tempo para obter uma solução.

Para análise, dois bancos de dados foram criados.

Algoritmo 2: Novo Algoritmo

```
Data: π,t
while π ≠ ι do
    if Existem 2-moves válidos then
        if Heurística 1 then
            Aplicar 2-move válido correspondente
        end
        else
            if Heurística 2 then
                Aplicar 2-move válido correspondente
            end
            else
                if Heurística 3 then
                    Aplicar 2-move válido correspondente
                end
                else
                    if t > 0 and Heurística 4 then
                        Aplicar 2-move válido correspondente
                    end
                    else
                        Aplicar 2-move válido correspondente à Heurística 5
                    end
                end
            end
        end
    end
    else
        if Existem 0-moves bons then
            if Heurística 1 then
                Aplicar 0-move bom correspondente
            end
            else
                if Heurística 2 then
                    Aplicar 0-move bom correspondente
                end
                else
                    if t > 0 and Heurística 4 then
                        Aplicar 0-move bom correspondente
                    end
                    else
                        Aplicar 0-move bom correspondente à Heurística 5
                    end
                end
            end
        end
        else
            if Heurística 6 then
                Aplicar (4,3)-sequence correspondente
            end
            else
                Aplicar sequência de transposições correspondente à Heurística 7
            end
        end
    end
end
```

- **dataset$_1$**: criado com todas as distâncias $d(\pi)$ para todas as instâncias π, onde $|\pi| \leq 10$. Esse banco de dados contém 4.037.913 instâncias, sendo 3.628.800 instâncias de tamanho 10, 362.880 instâncias de tamanho 9, 40.320 instâncias de tamanho 8, 5.040 instâncias de tamanho 7, 720 instâncias de tamanho 6, 120 instâncias de tamanho 5, 24 instâncias de tamanho 4, 6 instâncias de tamanho 3, 2 instâncias de tamanho 2 e 1 instância de tamanho 1.

- **dataset$_2$**: criado com 1000 instâncias aleatórias, sendo 100 instâncias de tamanho 10, 100 instâncias de tamanho 20 e assim, sucessivamente, até 100 instâncias de tamanho 100. Dada a dificuldade em se obter a distância real dessas permutações, decidiu-se usar o limitante inferior de Bafna e Pevzner [9].

A análise foi divida em duas partes. A primeira parte consiste em ordenar todas as instâncias no $dataset_1$. O resultado é mostrado na Tabela 2.2, onde se observa a quantidade de respostas não ótimas retornadas pelo algoritmo ao variar o valor t do *look-ahead* no intervalo [0,2]. Em todos os casos em que a distância calculada pelo algoritmo não foi a ótima, o valor fornecido foi uma unidade maior.

Observa-se a superioridade das versões que usam *look-ahead* sobre a que não usa. É possível ordenar todas as permutações π usando o número mínimo de transposições, para $|\pi| \leq 9$, com no mínimo $t = 1$.

Tamanho	0-LA	1-LA	2-LA
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	18	0	0
9	1.300	0	0
10	25.938	1.206	1.055

Tabela 2.2: Quantidade de respostas não ótimas fornecidas por cada algoritmo ao variar o valor do *look-ahead* no intervalo [0,2]. Na tabela, t-LA identifica o algoritmo proposto com a Heurística 4 usando o valor t para *look-ahead*.

A segunda parte da análise consiste em verificar a performance de cada algoritmo em permutações maiores. Para tanto, utilizou-se o $dataset_2$ para gerar o diagrama de Venn da Figura 2.2. O diagrama de Venn mostra com que frequência cada algoritmo fornece a melhor resposta. Neste contexto, a melhor resposta é aquela que possui o menor número

de transposições, sendo que não se pode afirmar se essa resposta é ótima ou não devido a dificuldade em se obter a distância exata para permutações grandes.

O diagrama de Venn da Figura 2.2 permite concluir que o *look-ahead* melhorou a resposta em 21,4% dos casos. Em 59.8% dos casos, os algoritmos forneceram a mesma resposta.

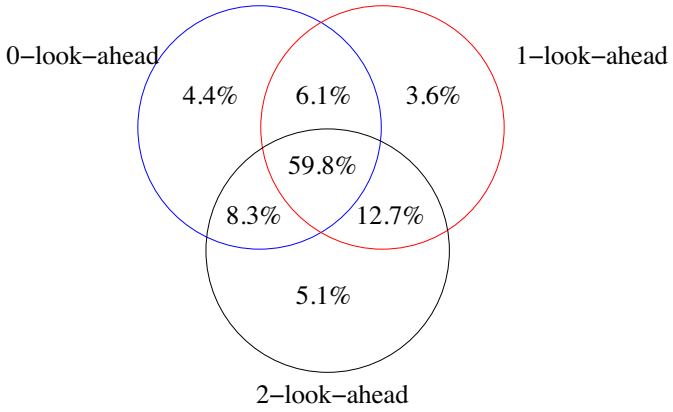


Figura 2.2: Diagrama de Venn mostrando a porcentagem de vezes em que cada versão do algoritmo fornece o melhor resultado.

A Figura 2.3 mostra o número relativo de vezes em que cada algoritmo fornece o melhor resultado. Percebe-se que 2 *look-aheads* fornecem melhores resultados, mas a vantagem é apertada. Vale observar que a versão do algoritmo com 2 *look-aheads* é superada nos testes com permutações de tamanho 20. Para permutações de tamanho 90 e 100, o algoritmo sem *look-ahead* foi melhor do que o que usa apenas 1.

Para permutações longas, calculou-se o tempo que demora para se obter uma resposta. Dessa forma, usou-se o $dataset_2$ para gerar o consumo de tempo na Tabela 2.3. A tabela expõe que para instâncias de tamanho 100, gastou-se em média apenas 0,04 minutos para obter uma resposta ao não se utilizar o *look-ahead*, enquanto que, com 1 ou 2 *look-aheads*, utilizou-se mais de 200 minutos em média. Conclui-se que os *look-aheads* devem ser utilizados quando o tempo para se obter uma resposta não é um fator determinante.

2.5 Análise Comparativa com Outros Algoritmos

A Tabela 2.4 mostra os resultados não ótimos obtidos por alguns algoritmos encontrados na literatura, de modo a facilitar a comparação com a Tabela 2.2, correspondente aos algoritmos mostrados neste capítulo.

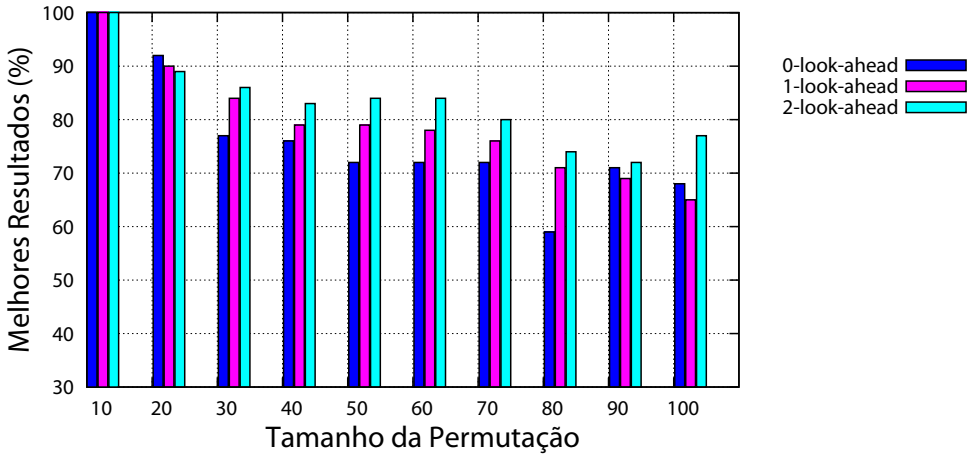


Figura 2.3: Número relativo de vezes em que cada versão do algoritmo fornece o melhor resultado. É importante ressaltar que mais de uma versão pode fornecer o melhor resultado.

Alguns dos valores mostrados na Tabela 2.4 foram publicados por terceiros e transcritos. Entretanto, dado que a análise obtida na literatura dificilmente gera resultados para permutações de tamanho 10, um o símbolo "-" foi adicionado para indicar a ausência do resultado.

Abaixo, são estabelecidos os algoritmos que aparecem nas colunas da Tabela 2.4.

- **EH/DD**: algoritmo de Elias e Hartman [46] implementado neste trabalho. **DD** se refere a "Dias e Dias", autores da implementação.

- **WDM**: algoritmo descrito por Walter, Dias e Meidanis [96]. Os próprios autores relatam os resultados transcritos na Tabela 2.4.

- **C/WCO**: algoritmo descrito por Christie [27] e implementado por Walter, Curado e Oliveira [95].

- **HS/H**: algoritmo descrito por Hartman e Sharan [58] e implementado por Honda [63].

- **BP/W**: algoritmo descrito por Bafna e Pevzner [9] e implementado por Walter e co-autores [97]. A implementação não corresponde exatamente ao algoritmo original de Bafna e Pevzner, pois novas regras foram inseridas por Walter e co-autores para melhorar a performance.

- **M**: algoritmo descrito e implementado por Mira e co-autores [76].

Tamanho	0-LA	1-LA	2-LA
10	0,00	0,00	0,01
20	0,00	0,01	0,01
30	0,00	0,09	0,08
40	0,00	0,57	0,60
50	0,00	1,78	1,86
60	0,00	5,24	5,58
70	0,01	17,12	17,37
80	0,01	40,79	53,88
90	0,02	103,50	115,26
100	0,04	244,80	234,19

Tabela 2.3: Média do tempo (em minutos) consumido para cada permutação. **0-LA** é o algoritmo sem *look-ahead*. **1-LA** é o algoritmo com 1 *look-ahead*, **2-LA** é o algoritmo com 2 *look-aheads*.

Todos os algoritmos mostrados na Tabela 2.4 exibem resultados não ótimos para permutações de tamanho 8, sendo que a quantidade de resultados não ótimos aumenta à medida que o tamanho da permutação aumenta. Dos algoritmos descritos neste capítulo, cujos resultados estão na Tabela 2.2, dois oferecem a distância correta para todas as permutações de tamanho menor ou igual a 9.

O algoritmo de Elias e Hartman obteve o pior resultado para permutações pequenas, apesar de ser o único algoritmo da Tabela 2.4 a garantir o fator de aproximação 1.375. Os algoritmos descritos na Seção 2.3 possuem a mesma razão de aproximação de Elias e Hartman e conseguem encontrar excelentes resultados na prática.

As heurísticas apresentadas neste capítulo foram implementadas usando a linguagem de programação `python`. A implementação poderá ser obtida livremente pelo link http://www.ic.unicamp.br/∼zanoni/orientacoes/doutorado/ulisses/dd2010.zip.

2.6 Publicações

Um pôster inicial sobre este capítulo foi apresentado em outubro de 2009 na conferência X-Meeting'2009 (*International Conference of the Brazilian Association for Bioinformatics and Computational Biology*) realizada em Angra dos Reis [33]. Essa versão apresentava uma visão geral do trabalho em progresso, bem como algumas das heurísticas.

A primeira versão completa das heurísticas foi apresentada no artigo "*Extending Bafna-Pevzner Algorithm*" (Ulisses Dias e Zanoni Dias) em fevereiro de 2010 na conferência ISB'2010 (*International Symposium on Biocomputing*) realizada em Calicute, Índia [35]. Este trabalho é composto de um algoritmo de aproximação na razão 1.5. O

Tamanho	WDM	C/WCO	HS/H	BP/W	M	EH/DD
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	6	0	2	0	0	0
7	72	0	108	0	0	273
8	1.167	40	1.517	34	124	2.866
9	14.327	1.182	25.425	1.007	2.977	37.393
10	-	-	-	-	69.297	449.243

Tabela 2.4: Resultados para outros algoritmos encontrados na literatura: (WDM) se refere ao algoritmo descrito e implementado por Walter, Dias e Meidanis [96], (C/WCO) se refere ao algoritmo descrito por Christie [27] e implementado por Walter, Curado and Oliveira [95], (HS/H) se refere ao algoritmo descrito por Hartman e Shamir [58] e implementado por Honda [63], (BP/W) é o algoritmo de Bafna e Pevzner [9] implementado com heurísticas por Walter e co-autores [97], (M) se refere ao algoritmo descrito e implementado por Mira e co-autores [76], (EH/DD) se refere ao algoritmo de Elias and Hartman implementado por Dias e Dias [35].

algoritmo contém as heurísticas apresentadas neste capítulo, exceto as Heurísticas 5 e 7.

O algoritmo conforme apresentado neste capítulo foi apresentado no resumo estendido "*An improved* 1.375*-approximation algorithm for the transposition distance problem*" (Ulisses Dias e Zanoni Dias) em agosto de 2010 na conferência BCB'2010 (*ACM International Conference on Bioinformatics and Computational Biology*) realizada em Niagara Falls, Estados Unidos [36]. Este resumo estendido adiciona as Heurísticas 5 e 7 para gerar um algoritmo de aproximação na razão 1.375.

Capítulo 3

Distância de Transposição usando Programação em Lógica com Restrições

Motivação: Muitos avanços foram obtidos com a abordagem clássica para o problema da distância de transposição. Entretanto, contribuições de pesquisas já consolidadas em outras áreas da computação podem propiciar avanços na obtenção de resultados, apesar de não participarem da abordagem clássico. Dentro desse contexto, os modelos baseados em Programação em Lógica com Restrições (CLP - do inglês *Constraint Logic Programming*) são promissores, pois permitem que as permutações e os eventos de transposição sejam representados naturalmente usando restrições.

Metodologia: Na abordagem de Programação em Lógica com Restrições, é possível contextualizar a distância de transposição como um Problema de Satisfação de Restrição (CSP - do inglês *Constraint Satisfaction Problem*) ou como um Problema de Otimização de Restrição (COP - do inglês *Constraint Optimization Problem*). No presente capítulo, utilizou-se ambos os paradigmas para gerar modelos implementados na plataforma `ECLiPSe` [44].

Resultados: Todos os modelos fornecem uma resposta exata para a distância de transposição, impedindo uma análise como a do Capítulo 2, baseada na quantidade de respostas não ótimas fornecidas pelos programas de ordenação. Assim, o tempo de processamento foi o único parâmetro na análise. Os resultados sugerem o paradigma CSP como o mais adequado. Uma análise final compara os modelos propostos com modelos em Programação Linear Inteira. Essa análise conclui que a Programação por Restrição é mais promissora para a distância de transposição que a Programação Linear Inteira.

3.1 Introdução

O Capítulo 2 apresentou heurísticas para a distância de transposição usando a abordagem "clássica", cuja característica principal é usar ferramentas como o grafo de ciclos e o grafo de breakpoints, com autoria de Bafna, Pevzner e Hannenhalli em trabalhos voltados para a distância de transposição e de reversão [6, 9, 56].

Distanciando-se da abordagem clássica, Meidanis e Dias [73] apresentaram um formalismo algébrico com o intuito de diminuir a dependência de representações visuais ao expor argumentos, provas e teoremas em rearranjo de genomas. Mira e co-autores [76] desenvolveram um algoritmo para o problema da distância de transposição usando o formalismo algébrico.

Para a distância de reversão, Bergeron [12] apresentou um algoritmo mais simples que o anterior de Hannenhalli e Pevzner [56]. Bergeron utiliza os conceitos de pares orientados, reversões induzidas por esses pares e intervalos fechados, conceitos estes que não haviam sido definidos anteriormente. Dessa forma, a abordagem de Bergeron também pode ser classificada como uma abordagem alternativa.

Benoît-Gagné e Hamel [11] apresentaram um algoritmo de aproximação de razão 3 para a distância de transposição. O algoritmo é baseado em uma estrutura que eles denominaram platô (*permutation plateaux*). O método é eficiente, mas os resultados são inferiores aos apresentados por outros algoritmos.

Caprara, Lancia e Ng [21, 22] lidaram com soluções práticas para o problema da distância de reversão. Eles apresentaram uma abordagem baseada no uso de Programação Linear. Em particular, eles trabalharam com uma relaxação para Programação Linear de um modelo de Programação Linear Inteira com um número exponencial de variáveis e restrições.

Dias e Souza [43] usaram Programação Linear Inteira para criar modelos para distâncias de reversão e transposição. O trabalho é uma novidade quando comparado a métodos anteriores, pois apresenta um modelo de tamanho polinomial.

Neste capítulo, seguiu-se a linha de pesquisa em abordagens alternativas para distância de transposição com a utilização de Programação em Lógica com Restrições (CLP - do inglês *Constraint Logic Programming*). Dois paradigmas foram usados: o primeiro caracteriza a distância de transposição como um Problema de Satisfação de Restrição (CSP - do inglês *Constraint Satisfaction Problem*), enquanto o segundo caracteriza a distância de transposição como um Problema de Otimização de Restrição (COP - do inglês *Constraint Optimization Problem*).

O capítulo está estruturado da seguinte forma. A Seção 3.2 apresenta uma breve introdução à programação com restrição. A Seção 3.3 apresenta os modelos de programação em lógica com restrições para o problema da distância de transposição usando os limitantes

superiores e inferiores apresentados na Seção 1.1.1. A Seção 3.4 apresenta a performance desses modelos usando o tempo de processamento como critério.

3.2 Programação por Restrição

A programação por restrição baseia-se na modelagem do comportamento de sistemas capturando as interações entre suas partes. Assim, um dado sistema é simplificado usando as entidades que o compõe e os relacionamentos dessas entidades entre si. Em geral, as entidades em um modelo de programação por restrição são chamadas de variáveis e os relacionamentos são chamados de restrições.

Definição 3.1 *Um modelo P de programação por restrição é formado por:*

- *Um conjunto de variáveis $X = \{x_1, x_2, \ldots, x_n\}$, onde cada variável possui o seu respectivo domínio $D_1, D_2, \ldots, D_n$.*
- *Um conjunto de restrições $C = \{c_1, c_2, \ldots, c_m\}$ sobre as variáveis em X.*

Uma solução para P é um conjunto $\{d_1, d_2, \ldots, d_n\}$, $d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n$, que satisfaz todas as restrições em C. P é consistente se possuir pelo menos uma solução, caso contrário, é dito inconsistente. Por exemplo, o programa P com a restrição $x > x + y$ é inconsistente se $x, y \in \mathbb{N}$, mas é consistente se $x, y \in \mathbb{I}$.

Dois paradigmas podem ser usados: satisfação de restrição (CSP - do inglês *Constraint Satisfaction Problem*) ou otimização de restrição (COP - do inglês *Constraint Optimization Problem*). Um problema no paradigma CSP é descrito conforme a Definição 3.1 acima.

Sejam um CSP $P_{csp} = \{C; x_1 \in D_1, x_2 \in D_2, \ldots, x_n \in D_n\}$ e uma função de custo $Cost = D_1 \times D_2 \times \ldots \times D_n \rightarrow R$. Um problema no paradigma COP $P_{cop} = (P_{csp}, Cost)$ requer uma solução de P_{csp} que otimiza a função $Cost$. Otimizar, neste contexto, pode significar maximar ou minimizar a função $Cost$. Em distância de transposição, otimizar deve ser entendido como minimizar o número de transposições que ordenam uma dada permutação π.

Para encontrar soluções de P em programação por restrição, a redução do espaço de busca nos domínios das variáveis é uma parte essencial. Os métodos de redução são chamados de algoritmo de propagação de restrições. Esses algoritmos reduzem o problema original em outro equivalente e mais fácil de resolver.

Definição 3.2 *Sejam P_1 e P_2 dois modelos de programação por restrição; sejam X_1 e X_2 os conjuntos de variáveis de P_1 e P_2, respectivamente. Diz-se que P_1 e P_2 são equivalentes em relação a um conjunto de variáveis X, tal que $X \subset X_1$ e $X \subset X_2$, se:*

- *Para toda solução d em P_1 existe uma solução em P_2 que coincide com d nas variáveis em X.*
- *Para toda solução e em P_2 existe uma solução em P_1 que coincide com e nas variáveis em X.*

Entretanto, apenas a redução não garante encontrar uma solução. Em geral, a solução é encontrada com uma busca no domínio das variáveis. As duas formas de busca a seguir são as mais comuns:

- **Busca Local:** o propósito é encontrar uma solução para o problema (CSP ou COP) a partir de uma atribuição inicial para todas as variáveis. Uma atribuição de valores para todas as variáveis é chamada de estado. A busca iterativamente melhora o estado corrente fazendo pequenas mudanças. Por exemplo, pode-se utilizar como parâmetro o número de restrições violadas por um estado e a busca local terminaria quando esse número se tornar 0.

  Na busca local, é necessária uma função que, a partir de um estado, encontre uma série de estados vizinhos. A execução busca iterativamente por um vizinho até que uma condição de parada seja satisfeita.

- **Busca Top-Down:** esta estratégia combina *branching* com propagação de restrição. Intuitivamente, *branching* é divisão de um problema P em dois problemas P_1 e P_2, de modo que P seja equivalente a $P_1 \cup P_2$. Dessa forma, é possível resolver os problemas P_1 e P_2 separadamente em uma árvore de busca. As folhas dessa árvore são ou uma solução para um sub-problema de P ou uma inconsistência.

  Um exemplo de *branching* é a estratégia conhecida como *labeling*. Essa estratégia consiste em particionar o domínio das variáveis em domínios unitários, propagando as restrições em cada ramo da árvore de busca. Quando uma inconsistência é encontrada na árvore de busca, a busca retorna para o nó pai desta folha (*backtrack*) e este nó continuará a busca com outro valor do domínio.

É importante ressaltar que o algoritmo de propagação de restrições atua em conjunto com a busca. Por exemplo, quando algum valor do domínio é atribuído às variáveis na estratégia *labeling*, a atribuição passa a surtir efeito no domínio de todas as outras variáveis imediatamente.

Em pacotes de desenvolvimento de programação por restrição, o algoritmo para encontrar soluções já faz parte da plataforma. O pacote `ECLiPSe` [44] foi utilizado para modelar a distância transposição e para gerar os resultados da Seção 3.4. O `ECLiPSe` utiliza a técnica de *labeling* para solucionar problemas com variáveis de domínio finito, caso

dos modelos criados para as transposições. Dois fatores relacionados a eficiência podem ser configurados:

- **Fator 1:** ordem em que as variáveis recebem valores do domínio.
- **Fator 2:** ordem em que os valores do domínio são atribuídos às variáveis.

Uma formulação do problema das n rainhas será usada para facilitar o entendimento dos fatores:

Definição 3.3 *Sejam R um conjunto de n rainhas e B um tabuleiro $n \times n$. O problema $P_{rainha}(n)$ das n rainhas consiste em posicionar as rainhas em B de modo que nenhuma rainha seja atacada por outra.*

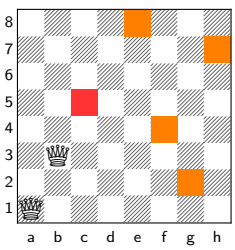
A formulação de $P_{rainha}(n)$ em programação por restrição é composta pelo conjunto de variáveis $X_{rainha} = \{x_1, x_2, \ldots, x_n\}$, $1 \leq x_i \leq n$ para todo $1 \leq i \leq n$. Nesta formulação, cada variável representa uma coluna do tabuleiro. As variáveis estão ordenadas da coluna mais à esquerda para a coluna mais à direita de B. Assim, x_1 é a coluna mais à esquerda de B e x_i é a i-ésima coluna da esquerda para a direita. O conjunto de variáveis parte do pressuposto de que duas rainha não devem ocupar a mesma coluna na solução final, o que é verdade. A linha onde se encontra a rainha da coluna i será atribuída à variável x_i.

$P_{rainha}(n)$ possui o seguinte conjunto de restrições para todo $j < i$:

- $x_i \neq x_j$ {Uma rainha por linha}
- $x_i - x_j \neq i - j$ {Uma rainha para cada diagonal ascendente}
- $x_i - x_j \neq j - i$ {Uma rainha para cada diagonal descendente}

A eficiência do modelo depende do número de *backtracks* necessários para encontrar uma solução.

Supondo a Estratégia$_1$, que consiste em atribuir valores para a variável x_i antes da variável x_j para $i < j$, sendo que o valor atribuído a x_i é o menor de seu domínio. A sequência de passos que posiciona corretamente as três primeiras colunas é dada a seguir. Um quadrado de cor vermelha no tabuleiro é usado para representar uma atribuição para a variável na busca. Um quadrado de cor laranja no tabuleiro é usado para representar o domínio de uma variável após a atribuição do quadrado de cor vermelha.

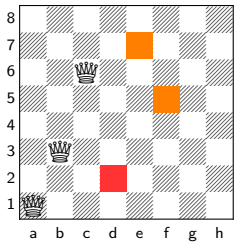


Passo 1: $x_1 = 1$, $x_2 = 3$. A variável x_3 possui domínio {5,6,7,8}. Entretanto, a atribuição $x_3 = 5$ (quadrado vermelho em `c5`) gera um efeito em cascata no domínio das outras variáveis. O domínio da variável x_6 se torna unitário e igual a {4}. Como essa é a única opção para x_6, a restrição $x_6 = 4$ (quadrado laranja em `f4`) é propagada para o domínio das demais. Após a propagação, o domínio da variável x_8 se tornará unitário e igual a {7}. O efeito em cascata de $x_8 = 7$ obriga

$x_7 = 2$ que, por conseguinte, gera o mesmo domínio unitário {8} para as variáveis x_4 e x_5. Assumindo, arbitrariamente, que $x_5 = 8$, então o domínio de x_4 torna-se nulo, evidenciando a inconsistência.

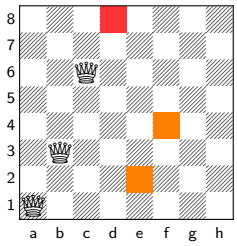
O algoritmo de propagação de restrições sobre o domínio das variáveis permite que a inconsistência seja identificada logo que a atribuição $x_3 = 5$ seja efetuada. Uma busca em profundidade convencional (sem propagação de restrição) continuaria a busca atribuindo valores para as variáveis de forma infrutífera até constatar que não há soluções.

A busca continua eliminando o valor 5 do domínio da variável x_3, que passa a ser o conjunto {6,7,8}.

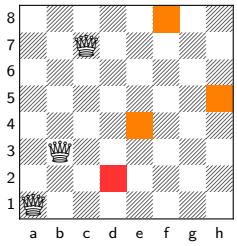


Passo 2: $x_1 = 1$, $x_2 = 3$, $x_3 = 6$. A atribuição dos três primeiros valores não esvazia o domínio das outras variáveis e nem gera domínios unitários, então a busca prossegue. A variável x_4 possui domínio {2,8}. Entretanto, a atribuição $x_4 = 2$ (quadrado vermelho em `d2`) leva a uma configuração inválida, pois o domínio de x_5 se tornaria unitário e igual a {7} (quadrado laranja em `e7`), o que forçaria $x_6 = 5$ (quadrado laranja em `f5`). Ao final da propagação, os domínios de x_7 e x_8 (colunas **g** e **h**) se tornam vazios.

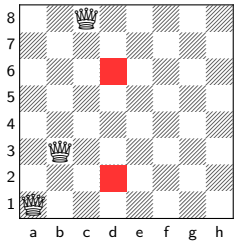
Os passos seguintes serão apresentados com menos detalhes:



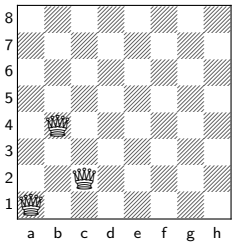
Passo 3: Ao eliminar a opção $x_4 = 2$, a variável x_4 passa a ter domínio {8}. Entretanto, $x_4 = 8$ leva a uma configuração inválida, pois o domínio das variáveis x_7 e x_8 (colunas **g** e **h**) estarão vazios após a propagação das restrições. Neste caso, o domínio da própria variável x_4 se tornou vazio, o que indica que a configuração das três primeiras rainhas não participa de nenhuma solução.



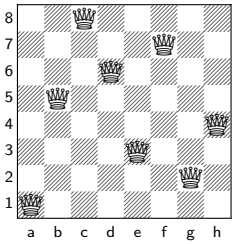
Passo 4: $x_1 = 1$, $x_2 = 3$, $x_3 = 7$. A variável x_4 possui domínio {2}, mas $x_4 = 2$ leva a uma configuração inconsistente, pois o domínio das variável x_7 (coluna **g**) torna-se vazio. Neste caso, o domínio da própria variável x_4 se tornou vazio, o que mostra que a configuração das três primeiras rainhas não participa de nenhuma solução.



Passo 5: $x_1 = 1$, $x_2 = 3$, $x_3 = 8$. A variável x_4 possui domínio {2,6}. Isso obriga a uma busca semelhante às anteriores. Essa busca mostraria que ambos os valores no domínio de x_4 levam a configurações inválidas. Nesse caso, o domínio da variável x_3 se torna nulo, pois $x_3 = 8$ era a última opção no domínio, o que indica que as duas primeiras rainhas estão mal posicionadas.



Passo 6: O posicionamento $x_1 = 1, x_2 = 4, x_3 = 2$ seria o próximo a ser verificado. Entretanto, a busca análoga aos passos anteriores mostraria que essa configuração inicial leva apenas a configurações inválidas.



Primeira Solução: $x_1 = 1,\ x_2 = 5,\ x_3 = 8, x_4 = 6, x_5 = 3, x_6 = 7, x_7 = 2, x_8 = 4$. Essa solução seria encontrada pelo método de busca usando a Estratégia$_1$.

A Estratégia$_1$ resolve $P_{rainha}(8)$ com 10 *backtracks*. $P_{rainha}(16)$ é resolvido com 542 *backtracks*. $P_{rainha}(32)$ não é resolvido antes do tempo limite estipulado (50 segundos).

A Estratégia$_2$ consiste em modificar a ordem em que as variáveis receberão valores do domínio (**Fator 1**). Nesse caso, as variáveis que representam colunas centrais recebem valores antes das variáveis que representam as colunas laterais. Por exemplo, para $P_{rainha}(8)$, a ordem de atribuição de valores é: $x_4, x_5, x_3, x_6, x_2, x_7, x_1, x_8$. Os valores recebidos pelas variáveis permanece o menor em seu domínio. A Estratégia$_2$ aproveita a observação de que uma rainha posicionada em uma coluna central gera mais restrições sobre as outras do que uma rainha posicionada em uma coluna lateral.

A Estratégia$_2$ resolve $P_{rainha}(8)$ sem *backtracks*. $P_{rainha}(16)$ é resolvido com 17 *backtracks*. $P_{rainha}(32)$ não é resolvido antes do tempo limite estipulado (50 segundos).

Quando uma variável recebe um valor, tende a diminuir o domínio das outras. Atribuir valor a uma variável pode fazer com que o domínio de outra se torne nulo, o que obrigará um *backtrack*. A Estratégia$_3$ minimiza essa dificuldade atribuindo valores primeiro para as variáveis com menor domínio. Neste caso, as variáveis são ordenadas dinamicamente conforme o progresso da busca.

A Estratégia$_3$ resolve $P_{rainha}(8)$ com 10 *backtracks*. $P_{rainha}(16)$ é resolvido com 3 *backtracks*. $P_{rainha}(32)$ é resolvido com apenas 4 *backtracks*. $P_{rainha}(75)$ é resolvido com 818 *backtracks*.

Na Estratégia$_3$, quando duas variáveis x_i, x_j possuem domínio de mesmo tamanho, a variável x_i receberá valor antes da variável x_j, para $i < j$. A Estratégia$_4$ muda essa ordenação atribuindo valores para as variáveis que representam colunas centrais (de modo análogo a Estratégia$_2$).

A Estratégia$_4$ resolve $P_{rainha}(8)$ e $P_{rainha}(16)$ sem *backtracks*. $P_{rainha}(32)$ é resolvido com apenas 1 *backtracks*. $P_{rainha}(75)$ é resolvido com 719 *backtracks*.

As Estratégias utilizaram apenas o **Fator 1** para melhorar a performance da busca, pois os valores atribuídos às variáveis foram sempre os menores no domínio. A Estratégia$_5$ altera a ordem em que os valores do domínio são atribuídos às variáveis usando a seguinte observação: colocar uma rainha no centro de uma coluna restringe mais as outras rainhas do que colocar no início. Assim, a Estratégia$_5$ utiliza a ordenação de variáveis da Estratégia$_4$, mas atribui às variáveis os valores que representam linhas centrais antes das linhas superiores e inferiores.

A Estratégia$_5$ resolve $P_{rainha}(8)$ e $P_{rainha}(16)$ com 3 *backtracks*. $P_{rainha}(32)$ é resolvido com 7 *backtracks*. $P_{rainha}(75)$ é resolvido sem *backtracks*. A Estratégia$_5$ também resolve $P_{rainha}(120)$ sem *backtracks*, sendo que este último não havia sido resolvido por nenhuma das estratégias anteriores antes do tempo limite estipulado.

O problema das n rainhas ilustra a variação de performance das diferentes estratégias de busca.

3.3 Modelos para a Distância de Transposição

Os limitantes definidos na Seção 1.1.1 geram formulações usando o paradigma CSP ou paradigma COP. Em ambos os casos, os limitantes permitem diminuir o espaço de busca, com a consequente diminuição do tempo em que uma resposta é obtida.

A notação a seguir é usada para descrever as formulações. Esta notação segue o modelo de formulação de Marriot e Stuckey [72] e se assemelha à linguagem `prolog`.

- **Termos Básicos:**
  - **Variáveis:** começam com letras maiúsculas ou com o caracter "_" para variáveis anônimas. Por exemplo, K,X,$\_Y$.
  - **Strings :** qualquer sequência de caracteres entre aspas.
  - **Átomos:** sequência de caracteres que começam com letra minúscula. Por exemplo, $joao$, $pedro$ e $jose$.
- **Termos Compostos:** formados por um átomo e uma série de argumentos. Por exemplo, $homem(joao)$, $filho(joao, pedro)$.
- **Estrutura do Programa:**
  - **Programa:** um conjunto de unidades lógicas chamadas de **predicados**.
  - **Predicado:** uma coleção de **cláusulas**.
  - **Cláusula:** pode ser uma **regra** ou um **fato**.

- **Regra:** construção da forma "*cabeca* :- *corpo*.", onde *cabeca* é um **átomo** ou um **termo composto** e *corpo* é uma sequência não vazia de **átomos** ou **termos compostos**. Uma regra deve ser vista como uma implicação lógica: se *corpo* é verdadeiro, então *cabeca* é verdadeiro.
- **Fato:** construção da forma "*cabeca*.", onde *cabeca* deve ser um **átomo** ou um **termo composto**. Um fato é uma verdade para o programa.

O exemplo abaixo ilustra a estrutura de um programa:
$homem(pedro)$.
$homem(jose)$.
$filho(joao, pedro)$.
$filho(jose, joao)$.
$pai(X, Y)$:- $filho(Y, X), homem(X)$.
$avo(X, Y)$:- $pai(X, K), pai(K, Y)$

No exemplo, o programa é formado por quatro predicados: *homem*, $filho$, *pai* e *avo*. Os predicados *homem* e $filho$ possuem duas cláusulas cada, enquanto que os predicados *pai* e *avo* possuem apenas uma cláusula. Ao todo, existem seis cláusulas no programa, sendo que as quatro primeiras correspondem a fatos e as duas últimas são regras.

Neste capítulo, a notação `prolog` será estendida de duas maneiras:

- As letras gregas π, σ e ι são adicionadas para representar permutações.
- A construção $X :: [A..B]$ indica que X (ou cada elemento em X, se X for uma lista) pertence ao intervalo $[A, B]$.

Usando a notação, uma permutação é formulada da seguinte forma.

$$\begin{aligned} &permutation(\pi, N) \text{ :-} \\ &\quad length(\pi, N), \\ &\quad \pi :: [1..N], \\ &\quad all\_different(\pi). \end{aligned} \tag{3.1}$$

A última linha de $permutation(\pi, N)$ faz uma chamada a $all\_different(\pi)$ para garantir que todos os elementos de π são diferentes. O predicado $all\_different(\pi)$ não será detalhado por ser pré-definido na maioria das plataformas de programação por restrição, inclusive na plataforma `ECLiPSe`.

De um modo geral, $permutation(\pi, N)$ define uma permutação π de tamanho N. Nesta definição, π é um vetor de elementos no intervalo $[1, N]$, em que todos os elementos deste vetor são diferentes entre si.

A cláusula $transposition(\pi, \sigma, I, J, K)$ em (3.2) define uma transposição $\rho(i, j, k)$, $0 < i < j < k \leq n$. Nesta cláusula, a permutação π é dividida em quatro sub-listas C_1, C_2, C_3, C_4, onde $C_1 = (\pi_1, \ldots, \pi_{i-1})$, $C_2 = (\pi_i, \ldots, \pi_{j-1})$, $C_3 = (\pi_j, \ldots, \pi_{k-1})$ e $C_4 = (i_k, \ldots, i_n)$. A permutação σ resultante é: $\sigma = C_1C_3C_2C_4$. Vale ressaltar que C_1 e C_4 podem ser vazios.

$$
\begin{aligned}
&transposition(\pi, \sigma, I, J, K) \text{ :-} \\
&\quad permutation(\pi, N), \\
&\quad permutation(\sigma, N), \\
&\quad 1 \leq I < J < K \leq N, \\
&\quad split(\pi, I, J, K, C_1, C_2, C_3, C_4), \\
&\quad \sigma = C_1C_3C_2C4.
\end{aligned}
\tag{3.2}
$$

Para modelar o problema usando o paradigma CSP, é necessário conhecer todas as variáveis e restrições. Entretanto, conhecer o número de variáveis e restrições implica conhecer, a priori, a distância de transposição, que é o que precisaria ser calculado. Para contornar essa dificuldade, a seguinte abordagem foi utilizada: seleciona-se um candidato T como o menor número no intervalo $[L..U]$, onde L é um limitante inferior conhecido para o problema e U é um limitante superior. Feita essa escolha, tenta-se encontrar uma sequência de transposições que ordene π com T transposições. Se uma falha ocorrer, escolhe-se um outro candidato incrementando o valor de T. O predicado *indomain* em (3.3) cria esse comportamento.

Vale ressaltar que o limitante superior U não interfere no tempo computacional, já que a estratégia de busca começa com o limitante inferior e termina logo que a distância for encontrada. Por definição, essa distância será menor ou igual a qualquer limitante superior.

$$
\begin{aligned}
&distance(\iota, 0, \_Model).\\
&distance(\pi, T, Model) \text{ :-}\\
&\quad bound(\pi, Model, LowerBound, UpperBound),\\
&\quad T :: [LowerBound..UpperBound],\\
&\quad indomain(T),\\
&\quad transposition(\pi, \sigma, \_I, \_J, \_K),\\
&\quad distance(\sigma, T-1, Model).
\end{aligned}
\tag{3.3}
$$

Todos os modelos CSP possuem a estrutura (3.3), variando apenas nos limitantes utilizados na chamada ao predicado $bound(\pi, Model, LowerBound, UpperBound)$. Nesta chamada, o parâmetro $Model$ seleciona o modelo escolhido. Os seguintes modelos foram gerados:

1. **def_csp**: modelo sem limitante inferior criado para facilitar a análise.
2. **br_csp**: modelo com limitante inferior de *breakpoints* definido no Lema 1.2.
3. **cg_csp**: modelo com limitante inferior do grafo de ciclos definido no Lema 1.4.

É possível melhorar o tempo de execução dos modelos CSP usando a permutação reduzida [27], pois ela tem no máximo o mesmo tamanho da permutação original, o que tende a diminuir o espaço de busca. Um novo predicado para cálculo de distância foi criado para usar a permutação reduzida.

$$
\begin{aligned}
&distance\_red(\pi, 0, \_Model) \text{ :- } reduce(\pi, [\,]).\\
&distance\_red(\pi, T, Model) \text{ :-}\\
&\quad reduce(\pi, \pi_{red}),\\
&\quad bound(\pi_{red}, Model, LowerBound, UpperBound),\\
&\quad T :: [LowerBound..UpperBound],\\
&\quad indomain(T),\\
&\quad transposition(\pi_{red}, \sigma, \_I, \_J, \_K),\\
&\quad distance\_red(\sigma, T-1, Model).
\end{aligned}
\tag{3.4}
$$

Em (3.4), a permutação identidade é uma lista vazia. Isso ocorre porque a operação para reduzir uma permutação, $reduce(\pi, \pi_{red})$, fará com que todos os elementos da identidade sejam eliminados. Para identificar quando a permutação reduzida é utilizada, o

sufixo *_red* é adicionado ao nome do modelo. Por exemplo, o modelo que utiliza o limitante inferior do grafo de ciclos e a operação de redução é chamado de **cg_csp_red**.

O segundo paradigma de programação consiste em analisar a distância de transposição como um COP. Um conjunto de variáveis binárias B foi criado para informar quando uma transposição é utilizada. As variáveis binárias são importantes na criação da função objetivo. O predicado *transposition_cop* precisa ser criado para se adequar às novas variáveis. Quando uma transposição $\rho(i, j, k)$ não for utilizada, diz-se que $i = j = k = 0$.

O predicado (3.5) verifica quando esse evento ocorre. O último parâmetro do predicado indica a ocorrência do evento de transposição e será, posteriormente, adicionado nas variáveis B.

$$\begin{aligned} &transposition\_cop(\iota, \iota, 0, 0, 0, 0).\\ &transposition\_cop(\pi, \sigma, I, J, K, 1) \text{ :- } transposition(\pi, \sigma, I, J, K). \end{aligned} \tag{3.5}$$

O cálculo da distância utilizando o predicado (3.5) é feito pelo predicado *distance_cop* (3.6). Esse predicado configura as variáveis B de acordo com o limitante superior. A função objetivo $Cost$ é a soma das variáveis B associadas a cada transposição ρ_k, $Cost = \sum_{k=1}^{UB} B_k$, onde UB é um limitante superior conhecido. A distância é o valor mínimo da função objetivo $d(\pi) = \min cost$.

$$\begin{aligned} &distance\_cop(\pi, N, Model) \text{ :-}\\ &\quad bound(\pi, Model, LowerBound, UpperBound),\\ &\quad length(B, UpperBound),\\ &\quad upperbound\_constraint(\pi, B, UpperBound),\\ &\quad sum(B, Cost),\\ &\quad Cost \geq LowerBound,\\ &\quad minimize(Cost, N). \end{aligned} \tag{3.6}$$

O predicado *upperbound_constraint* (3.7) recebe uma permutação π, o limitante superior $UpperBound$ e uma lista B que deve armazenar 0 ou 1 na posição k, dependendo do fato de a k-ésima transposição ser utilizada. Este predicado é o responsável por garantir que a k-ésima permutação resulte da $(k-1)$-ésima permutação pela aplicação de uma transposição ou, caso contrário, seja igual a ela. Uma última funcionalidade do predicado (3.7) é verificar se ainda é possível ordenar a permutação usando as transposições que sobram, pois isto evitaria cálculo desnecessário.

$$
\begin{aligned}
&upperbound\_constraint(\iota, [\,], \_UpperBound).\\
&upperbound\_constraint(\pi, [B|Bs], UpperBound) :-\\
&\quad transposition\_cop(\pi, \sigma, \_I, \_J, \_K, B),\\
&\quad bound(\pi, Model, LowerBound, \_UpperBound),\\
&\quad UpperBound \geq LowerBound,\\
&\quad upperbound\_constraint(\sigma, Bs, UpperBound - 1).
\end{aligned}
\tag{3.7}
$$

Todos os modelos COP possuem a estrutura acima. Os modelos COP foram usados para analisar os limitantes superiores nos Lemas 1.5 e 1.6. O modelo que usa o limitante superior do grafo de ciclos definido no Lema 1.5 é chamado de **cg_cop**. O modelo que usa o limitante superior do grafo Γ definido no Lema 1.6 é chamado de **gg_cop**. Ambos os modelos usam o limitante inferior do grafo de ciclos definido no Lema 1.4.

3.4 Análise Comparativa dos Modelos

Todos os modelos foram implementados usando a plataforma `ECLiPSe` [44]. Os experimentos foram conduzidos em uma máquina Intel® Xeon™, 3.20GHz, com 3.5 GB RAM, Ubuntu 7 e `ECLiPSe` 6.0.

A Tabela 3.1 resume os resultados. Os valores da tabela se referem à média do tempo necessário para obter a distância de todas as permutações π para $|\pi| < 7$. Dado o crescimento exponencial do espaço de busca, alguns modelos demoram um tempo considerável para fornecer uma resposta quando $7 \leq |\pi| \leq 11$. Dessa forma, foram escolhidas 1000 instâncias ao acaso e a média do tempo para obter a distância destas instâncias foi inserido na Tabela 3.1. O símbolo "−" é inserido caso o modelo demore mais de 15 horas para terminar os testes.

A Tabela 3.1 mostra que o limitante usando o grafo de ciclos é superior ao limitante usando breakpoints. Também é possível concluir que a redução gera uma substancial melhora na performance. Por exemplo, para $|\pi| = 11$, o resultado obtido pelo modelo **cg_csp_red** é quase 4.9 vezes melhor que **cg_csp**. O ponto negativo da redução é a maior dificuldade em obter a sequência mínima de transposições que ordenam a permutação de entrada. Os outros modelos necessitam de praticamente nenhuma modificação, bastando acrescentar um novo vetor de variáveis.

Os modelos COP possuem desempenho inferior aos modelos CSP. Entretanto, é preciso mencionar que não foram criadas heurísticas de buscas específicas para eles. As soluções foram obtidas usando o algoritmo genérico do pacote `ECLiPSe`, baseado em *branch and bound* e propagação de restrições.

	CSP				COP		
n	**def_csp**	**br_csp**	**cg_csp**	**cg_csp_red**	**def_cop**	**cg_cop**	**gg_cop**
4	0.006	0.010	0.005	0.005	0.134	0.019	0.039
5	0.069	0.087	0.009	0.006	2.530	0.061	0.149
6	1.887	2.367	0.022	0.013	–	1.842	4.421
7	51.69	30.707	0.045	0.024	–	3.797	39.024
8	–	–	0.233	0.104	–	–	–
9	–	–	0.946	0.313	–	–	–
10	–	–	6.816	2.016	–	–	–
11	–	–	20.603	4.212	–	–	–

Tabela 3.1: Tempo médio (em segundos) para calcular a distância de permutações de acordo com o seu tamanho. O símbolo "–" é mostrado caso a bateria de testes não seja finalizada em 15 horas.

Os resultados dos modelos CSP mostrados na Tabela 3.1 podem ser melhorados usando as seguintes técnicas:

- Quando a distância de uma permutação π for obtida, propagar $d(\pi)$ para todas as permutações σ, tais que $\sigma \in \pi_\circ^\circ$. Isso permite diminuir o número de permutações que precisam ser analisadas.

- Utilizar o grafo de ciclos de uma permutação com o intuito de ordenar o conjunto de transposições. Assim, as que possuem chance maior de pertencer a alguma sequência ótima de transposições são utilizadas primeiro. Essa estratégia de ordenação é análoga àquela usada no problema das n rainhas para melhorar a performance (Seção 3.2).

A primeira técnica utiliza a equivalência toroidal definida na Seção 1.1.1. Se duas permutações pertencem à mesma classe toroidal, então os grafos de ciclos associados a elas são isomórficos [59], implicando que ambas possuem a mesma distância.

Por exemplo, as sete permutações a seguir estão na mesma classe toroidal: (1 6 2 4 5 3), (1 6 3 4 2 5), (2 3 1 5 6 4), (3 5 6 4 1 2), (4 5 3 6 1 2), (5 1 3 4 2 6), (5 2 3 1 4 6). Quando uma destas permutações for pesquisada, a distância de transposição para o grupo inteiro passará a ser conhecida. Os modelos mostrados na Tabela 3.1 recalculam a distância caso seja requerida para as outras permutações dentro da mesma classe toroidal.

A segunda técnica propõe ordenar o espaço de busca, iniciando-se com as transposições "mais promissoras". Os modelos apresentados até o momento ainda não utilizam este critério e aplicam as transposições com os menores valores do domínio das variáveis I, J e K. Em outras palavras, as transposições são pesquisadas na seguinte ordem: $\rho(1,2,3)$, $\rho(1,2,4)$, $\rho(1,2,5)$ e assim por diante.

Bafna e Pevzner [9] subdividem as transposições em: 2-moves válidos, 2-moves, 0-moves bons, 0-moves válidos e 0-moves. Esses subconjuntos são uma ordenação das transposições conforme a chance de pertencerem à solução. Por exemplo, se existe um 2-move válido para uma permutação π, então ele possui maior chance de estar na sequência ótima de transposições do que um 0-move. Como essa classificação depende apenas do grafo de ciclos, é possível utilizá-la em modelos que usam o limitante inferior de Bafna e Pevzner sem a necessidade de estruturas adicionais.

Assim, os modelos **cg_csp** e **cg_csp_red** são os melhores candidatos para se beneficiarem das novas técnicas. Entretanto, o modelo **cg_csp_red** dificulta o processo de encontrar as permutações equivalentes devido a presença de permutações de tamanhos diferentes no espaço de busca (permutações de tamanhos diferentes são resultado da redução de algumas delas). Dessa forma, **cg_csp** foi o único modelo utilizado com as melhorias.

A Tabela 3.2 mostra os resultados obtidos mantendo o mesmo conjunto de dados e a mesma plataforma computacional utilizados na Tabela 3.1. Para ressaltar a melhora de performance, o tempo médio para se obter respostas para permutações de tamanho 11 foi de 0.034 segundos, em contrapartida, 20.603 segundos eram necessários antes das melhorias.

n	**cg_csp**
6	0.004
7	0.003
8	0.016
9	0.008
10	0.198
11	0.034

Tabela 3.2: Tempo médio (em segundos) do modelo **cg_csp** quando são aplicadas as técnicas para melhorar a performance.

A Tabela 3.2 evidencia um fato interessante, o valor médio para ordenar permutações de tamanho 10 foi maior do que o valor médio para ordenar permutações de tamanho 11. Isso ocorre porque algumas permutações foram particularmente difíceis de ordenar usando as novas técnicas. Uma linha de pesquisa consiste em identificar os fatores que criam essa dificuldade em certas permutações.

Uma comparação final pode ser feita entre os modelos propostos neste capítulo e os modelos usando Programação Linear Inteira de Dias e Souza [43]. Eles apresentam um modelo que encontra a solução para $|\pi| = 9$ com 143.5 segundos, em média. Além disso, para $\pi = 10$, o modelo se tornou proibitivo.

Esses resultados mostram que os modelos usando Programação por Restrição são mais promissores que os modelos usando Programação Linear Inteira. Entretanto, não é

possível fazer uma análise mais aprofundada, porque as instâncias e o ambiente computacional utilizado por Dias e Souza são diferentes dos utilizados neste capítulo.

3.5 Publicações

O presente capítulo foi apresentado no artigo "*Constraint programming models for transposition distance problem*" (Ulisses Dias e Zanoni Dias) em julho de 2009 na conferência BSB'2009 (*Brazilian Symposium on Bioinformatics*) realizada em Porto Alegre [32]. Este artigo contém todos os modelos apresentados neste capítulo. Entretanto, as técnicas para melhorar a busca dos modelos CSP ainda não haviam sido implementadas e os resultado presente na Tabela 3.2 ainda não haviam sido obtidos (anunciados como trabalhos futuros). Dessa forma, o presente capítulo completa o artigo apresentado no congresso.

Parte III

Reversões

Capítulo 4

Uma Ferramenta de Simulação para o Estudo de Reversões Simétricas em Genomas Bacteriais

Motivação: O alinhamento par-a-par de genomas bacteriais de espécies da família *Pseudomonadaceae* e dos gêneros *Mycobacterium*, *Shewanella* e *Xanthomonas* evidencia um padrão em "X" formado pelas sequências conservadas de DNA. Um modelo evolutivo capaz de explicar o padrão em "X" prega que reversões simétricas são dominantes em relação aos outros tipos de rearranjo [45]. Uma ferramenta de simulação para esse modelo evolutivo pode auxiliar no entendimento dessas reversões. Tal ferramenta também é útil para criar bases de dados que permitem avaliar programas de reconstrução de genomas ancestrais quando não existem ancestrais reais disponíveis.

Metodologia: Duas características comuns em genomas bacteriais foram utilizadas para gerar o modelo de simulação: a primeira delas são as reversões simétricas acima mencionadas e a segunda se refere aos blocos sintênicos, blocos de DNA com grande tendência em se manter conservados durante o processo evolutivo. Os parâmetros para a ferramenta de simulação foram configurados empiricamente, observando alinhamentos de genomas da família *Pseudomonadaceae*.

Resultados: Foram desenvolvidas medidas baseadas na dispersão de pontos nos *dotplots* dos alinhamentos. Tais medidas são uma comparação quantitativa entre alinhamentos de genomas reais e de genomas simulados e formam um arcabouço para geração de árvores filogenéticas usando cenários simulados. Essas árvores foram comparadas com árvores de referência e mostram que a ferramenta realiza um bom trabalho de simulação em termos de reversões simétricas.

4.1 Introdução

Em bactérias, reversões são os eventos de rearranjo mais frequentes. Em especial, reversões simétricas à origem de replicação são propostas por Eisen e co-autores [45] como o mecanismo principal que explica o padrão em "X" observado quando dois cromossomos circulares são alinhados [45]. Um estudo recente usando genomas de *Yersinias* encontrou evidências de que reversões simétricas são mais frequentes do que o esperado ao acaso [29].

A Seção 1.2.1 analisa o padrão em "X" na família *Pseudomonadaceae* e nos gêneros *Xanthomonas*, *Shewanella* e *Mycobacterium*, mostrando que há uma baixa probabilidade desse padrão ter ocorrido ao acaso. A Seção 1.2.1 também exibe o modelo de evolução que usa reversões simétricas à origem de replicação.

Uma área emergente em análise computacional de fenômenos evolucionários é a reconstrução de genomas ancestrais. Vários programas já foram propostos como, por exemplo, BPAnalysis [87], GRAPPA [78], BADGER [70] e MGR [15]. Dado que geralmente não há genomas ancestrais disponíveis para validar as ferramentas de reconstrução (as espécies ancestrais deixaram de existir), ferramentas de simulação são essenciais. Propõe-se neste capítulo, uma ferramenta de simulação específica para genomas bacteriais. Essa ferramenta recebeu o nome de SIB, acrônimo para *Symmetric Inversions in Bacteria*.

Uma ferramenta de simulação precisa modelar a evolução de genomas bacteriais reais. Essa modelagem é difícil por duas razões:

- Apesar da aparente predominância de reversões simétricas, genomas bacteriais sofrem outros tipos de rearranjo, assim como mutações pontuais.
- Alinhamentos de genomas de diferentes grupos de bactérias revela diferentes padrões de rearranjos.

Para contornar essas dificuldades, decidiu-se criar uma ferramenta de simulação cujo objetivo é reproduzir o padrão em "X" mais claramente observados em alinhamentos de certos grupos bacteriais: família *Pseudomonadaceae* e gêneros *Mycobacterium*, *Shewanella* e *Xanthomonas*. Um segundo critério para seleção desses grupos é a disponibilidade de genomas completos distintos em cada grupo.

O capítulo é estruturado da seguinte forma. A Seção 4.2 descreve os parâmetros do SIB e as medidas criadas para facilitar a configuração do simulador baseado em genomas reais. A Seção 4.3 relata um método de geração de árvores filogenéticas usando o SIB. Essas árvores, quando comparadas a árvores de referência encontradas na literatura, promovem uma validação qualitativa da ferramenta de simulação. A Seção 4.4 analisa o programa MGR, que reconstrói genomas ancestrais assumindo que apenas eventos de reversões ocorreram durante o processo evolutivo.

4.2 Parâmetros do SIB

Uma característica de genomas bacteriais é o fato de a pressão evolutiva desfavorecer a separação de determinados blocos de DNA, conhecidos como blocos sintênicos. A razão para um bloco se conservar está diretamente ligada às características funcionais por ele desempenhadas. Caso o bloco corresponda a um gene, um evento de rearranjo provavelmente impedirá o desempenho da função na célula, o que poderia ocasionar a morte do indivíduo dependendo da importância dessa função. Em outros casos, um bloco contém um grupo de genes relacionados pela função que desempenham, sendo que tal função seria impossibilitada caso um evento de larga escala afastasse um gene do outro, mesmo que nenhum deles seja diretamente afetado.

O SIB utiliza reversões simétricas e blocos sintênicos como características principais. As reversões simétricas são definidas como reversões $\rho(i, j)$, onde i e j são igualmente distantes da origem. Os blocos sintênicos inspiraram a definição dos blocos inquebráveis no modelo de simulação. Quando os pontos i ou j são localizados em um bloco inquebrável, uma assimetria é permitida de modo a incluir o bloco. A Figura 4.1 mostra a assimetria criada caso qualquer um dos lados da inversão ocorra dentro de um bloco inquebrável. O ponto `A` da reversão foi deslocado de modo a incluir o bloco inquebrável.

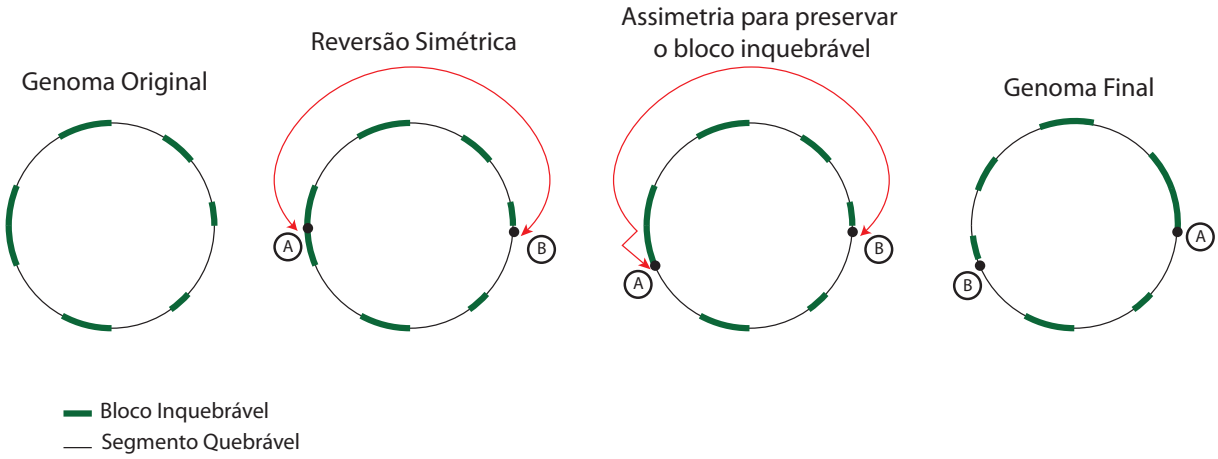


Figura 4.1: Uma assimetria ocorre caso qualquer um dos lados da inversão ocorra dentro de um bloco inquebrável. O ponto `A` da reversão foi deslocado de modo a incluir o bloco inquebrável na reversão simétrica.

Dessa forma, existe uma relação entre o tamanho dos blocos e o perfil das reversões simétricas, o que pode levar tanto a efeitos desejáveis como o gradativo afastamento dos blocos conservados da diagonal principal, quanto a efeitos indesejáveis como uma alta concentração de reversões simétricas ao redor de blocos sintênicos muito grandes.

Os parâmetros da ferramenta SIB são configurados de forma a diminuir os efeitos indesejáveis. Os parâmetros foram derivados de um estudo sobre a família *Pseudomonadaceae*. Vale lembrar que novos valores poderão ser atribuídos a esses parâmetros para se adaptar a cada necessidade. No estudo, utilizou-se alinhamentos par-a-par de todos os genomas da família *Pseudomonadaceae* como ponto de partida. A Tabela 4.1 apresenta os organismos utilizados.

Organismo	Referência	Tamanho (bp)
Azotobacter vinelandii	NC_012560	5.365.318
Pseudomonas aeruginosa PAO1	NC_002516	6.264.404
Pseudomonas aeruginosa LESB58	NC_011770	6.601.757
Pseudomonas aeruginosa PA7	NC_009656	6.588.339
Pseudomonas aeruginosa UCBPP PA14	NC_008463	6.537.648
Pseudomonas entomophila L48	NC_008027	5.888.780
Pseudomonas fluorescens Pf-5	NC_004129	7.074.893
Pseudomonas fluorescens Pf01	NC_007492	6.438.405
Pseudomonas fluorescens SBW25	NC_010501	6.722.539
Pseudomonas mendocina ymp	NC_009439	5.072.807
Pseudomonas putida F1	NC_009512	5.959.964
Pseudomonas putida GB 1	NC_010322	6.078.430
Pseudomonas putida KT2440	NC_002947	6.181.863
Pseudomonas putida W619	NC_010501	5.774.330
Pseudomonas stutzeri A1501	NC_009434	4.567.418
Pseudomonas syringae pv *phaseolicola* 1448A	NC_005773	5.928.787
Pseudomonas syringae B728a	NC_007005	6.093.698
Pseudomonas syringae pv *tomato* DC3000	NC_004578	6.397.126

Tabela 4.1: Genomas da família *Pseudomonadaceae*.

A família *Pseudomonadaceae* possui 18 organismos disponíveis para estudo. O alinhamento par-a-par permite a geração de $\binom{18}{2} = 153$ gráficos. Para utilizar as informações contidas nestes gráficos, utiliza-se as regiões conservadas em ambos os genomas. As regiões conservadas podem ocorrer com a mesma orientação (regiões vermelhas) ou com orientações opostas (regiões azuis).

As regiões conservadas foram "quebradas" em vários segmentos de 1kbp (tamanho médio de um gene) e cada um desses segmentos se tornou um ponto. No decorrer deste texto, o termo *dotplot* denota gráficos que já passaram por essa fase de pré-processamento. A Figura 4.2 ilustra um *dotplot* formado pelo alinhamento de dois genomas da família *Pseudomonadaceae*.

Uma característica chave dos *dotplots* é que a maioria dos pontos vermelhos ocorrem próximos à diagonal principal ($y = x$) e que a maioria dos pontos azuis ocorrem próximos

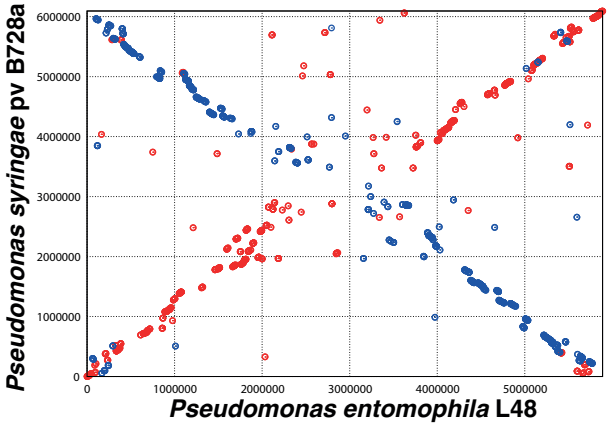


Figura 4.2: *Dotplot* formado ao alinhar genomas da família *Pseudomonadaceae*. O dotplot é composto de regiões conservadas que ocorrem com a mesma orientação (regiões vermelhas) e regiões conservadas que ocorrem com orientações opostas (regiões azuis).

à diagonal secundária ($y = L - x$, onde L é o tamanho do genoma). De acordo com o modelo evolutivo, um padrão em "X" nítido, indica proximidade entre os genomas.

Para caracterizar quantitativamente o padrão em "X", os *dotplots* foram particionado em áreas, de modo análogo ao procedimento usado na Seção 1.2.1. Em primeiro lugar, duas retas r e s são calculadas:

1. A subsequência crescente mais longa dos pontos vermelhos $v_1, v_2, \ldots, v_n$ e a subsequência decrescente mais longa dos pontos azuis $a_1, a_2, \ldots, a_n$ são obtidas. Este passo é resolvido em tempo $O(n \log n)$, onde n é o tamanho da sequência de entrada [88].

2. A reta r passa pelos pontos $v_1, v_2, \ldots, v_n$ e a reta s passa pelos pontos $a_1, a_2, \ldots, a_n$. Ambas as retas foram calculadas pelo método dos mínimos quadrados.

Delimita-se uma região R_1 ao redor de r e uma região R_2 ao redor de s. R_1 e R_2 possuem 10% da área total do dotplot. A região R_3 corresponde à área no *dotplot* que não pertence nem a R_1 e nem a R_2. Essa é importante para calcular a densidade de pontos em cada uma das áreas. As medidas de densidade d_1, d_2 e d_3 foram geradas, onde d_1 é a densidade dos pontos vermelhos em R_1, d_2 é densidade dos pontos azuis em R_2 e d_3 é a densidade de pontos (independente de cor) em R_3.

As densidades d_1, d_2 e d_3 são um forma de medir o quão próximos são dois *dotplots*. Para tanto, usa-se a distância euclidiana em um espaço tri-dimensional: sejam A e B dois *dotplots* e sejam $\{d_{A1}, d_{A2}, d_{A3}\}$ e $\{d_{B1}, d_{B2}, d_{B3}\}$ os conjuntos de densidades para A e B, respectivamente, a distância entre A e B é dada por:

$$dist(A,B) = \sqrt{(d_{A1} - d_{B1})^2 + (d_{A2} - d_{B2})^2 + (d_{A3} - d_{B3})^2}$$

A distância euclidiana permite medir a similaridade entre os *dotplots* obtidos a partir de genomas reais e *dotplots* obtidos a partir da ferramenta de simulação. A configuração de parâmetros que maximizou a similaridade modela reversões simétricas e blocos sintênicos seguindo distribuições normais.

Para reversões simétricas $\rho(i,j)$, o SIB tem como parâmetro a distância entre os pontos i e j e a origem de replicação, o que denominou-se "largura da reversão". A largura é caracterizada pela distribuição normal $N_\ell(\mu_\ell, \sigma_\ell)$, onde μ_ℓ é a média das larguras e σ_ℓ é o desvio padrão. Nas simulação cuja avaliação estatística é apresentada na Seção 4.3, utilizou-se os valores $\mu_\ell = n/3$ e $\sigma_\ell = n/5$, onde n é o número de pontos na simulação.

Os blocos sintênicos foram modelados seguindo uma distribuição normal $N_{\text{un}}(\mu_{\text{un}}, \sigma_{\text{un}})$, onde μ_{un} é o tamanho médio de um bloco sintênico e σ_{un} é o seu desvio padrão. A configuração padrão que gera cenários próximos aos observados em *Pseudomonadaceae* usa valores $\mu_{\text{un}} = 3$ e $\sigma_{\text{un}} = 10$.

Não foram modelados eventos de inserção e remoção, o que resulta no fato de que todos os genomas de um mesmo cenário possuem o mesmo tamanho. Os testes utilizam genomas de tamanho 2500, a média do número de pontos obtidos em genomas reais de *pseudomonas*.

4.3 Validação do SIB

Nesta seção, é apresentado um método indireto para avaliar a capacidade do SIB em gerar cenários que se aproximam da realidade. Este método gera árvores filogenéticas utilizando os modelos simulados e as compara a árvores de referências encontradas na literatura. Os seguintes passos mostram como uma árvore filogenética pode ser obtida usando o SIB.

1. Gerar duas ramificações de cenários simulados: $A = \{A_0, A_1, A_2, \ldots, A_{2500}\}$ e $B = \{B_0, B_1, B_2, \ldots, B_{2500}\}$. Nesses ramos, o i-ésimo elemento, $1 < i \leq 2500$, resulta do elemento $i-1$ após a aplicação de uma operação de reversão simétrica. Os elementos A_0 e B_0 correspondem à identidade. A Figura 1.8 ilustra as duas ramificações geradas para esse cenário.

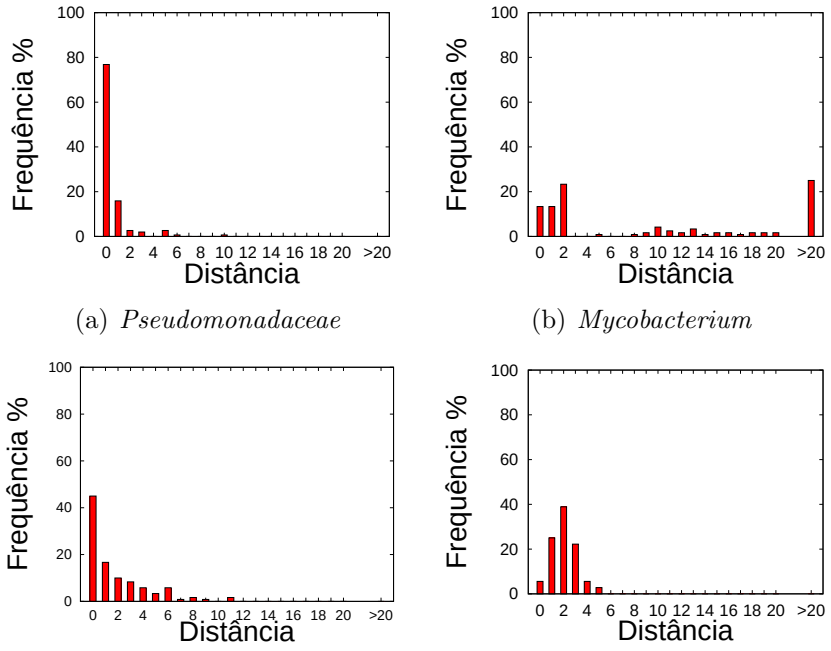
2. Gerar *dotplots* entre os genomas A_i e B_i, $1 \leq i \leq 2500$. Estes *dotplots* possuem $2 \times i$ reversões simétrica de distância.

3. Obter, para cada *dotplot* real, o *dotplot* simulado mais próximo usando a distância euclidiana.

4. Como os *dotplots* gerados pelo SIB possuem o número de reversões simétricas entre os genomas que os formam, associa-se esse número aos *dotplots* reais como uma medida de distância. Assim, cada *dotplot* real possuirá um valor que consiste no número de reversões simétricas utilizadas para gerar um *dotplot* semelhante a ele usando o SIB.

5. Gerar uma matriz de distância de *dotplots* reais com os valores obtidos no item anterior. Após isso, gerar uma árvore filogenética com a ferramenta `neighbor` do pacote PHYLIP [48].

Uma importante medida de qualidade da associação entre um *dotplot* real e um *dotplot* simulado é a própria distância euclidiana entre eles, pois é desejável que essa distância seja próxima de zero. A Figura 4.3 mostra um histograma com distâncias euclidianas para cada um dos quatro grupos bacteriais. O histograma da família *Pseudomonadaceae* é o melhor, já que 76% das distâncias são menores do que 1. O histograma para *Shewanella* e para *Xanthomonas* mostram que 61% e 30% das distâncias são menores ou iguais a 1, respectivamente. O histograma para o grupo *Mycobacterium* mostra que esse grupo possui as associações mais problemáticas.



(a) *Pseudomonadaceae* (b) *Mycobacterium*

(c) *Shewanella* (d) *Xanthomonas*

Figura 4.3: Histograma das distâncias euclidianas entre *dotplots* obtidos de organismos reais e o *dotplot* mais próximo obtido na simulação

A geração de árvores filogenéticas é uma maneira indireta de avaliar se os cenários gerados pelo SIB correspondem à realidade. Para isso, as árvores filogenéticas obtidas usando o SIB são comparadas com árvores filogenéticas confiáveis encontradas na literatura. As árvores filogenéticas obtidas pelo SIB são mostradas nas Figuras 4.4, 4.5, 4.6 e 4.7. Nestas figuras, são também mostradas as árvores de referências e os artigos onde foram publicadas.

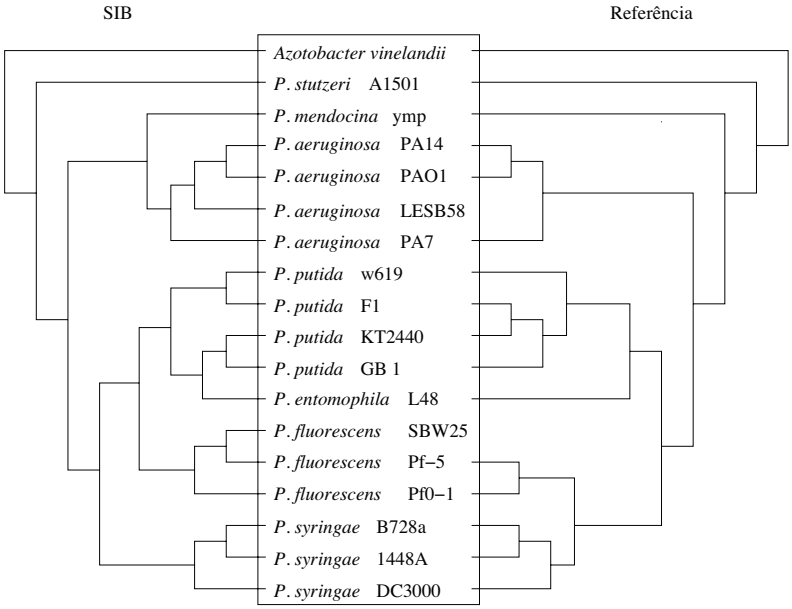


Figura 4.4: Comparação entre a árvore da família *Pseudomonadaceae* produzida pelo SIB e a árvore proposta por Setubal et al [90].

A similaridade entre as árvores obtidas pelo SIB e as árvores de referência encontradas na literatura foi medida usando o método apresentado por Vienne et al [94], que fornece um `p-value` para a similaridade entre duas árvores de entrada. No caso da família *Pseudomonadaceae*, o `p-value` é 7×10^{-6}; o `p-value` para os gêneros *Shewanella* e *Xanthomonas* é $1{,}3 \times 10^{-2}$; o `p-value` para o gênero *Mycobacterium* é $7{,}5 \times 10^{-5}$. Dessa forma, é possível concluir que a família *Pseudomonadaceae* e o grupo *Mycobacterium* obtiveram os melhores resultados, o que está de acordo com uma inspeção visual pelas Figuras 4.4, 4.5, 4.6 e 4.7.

A análise toma como verdade a hipótese de que as árvores de referência encontradas na literatura estão corretas, mas é possível que mesmo essas árvores contenham erros. Por exemplo, é possível encontrar um exemplo em que o SIB agrupa corretamente organismos (em termos de distância de rearranjos), mas a árvore de referência mostra um diferente agrupamento. Este exemplo é ilustrado na Figura 4.8. O método de máxima verossimilhança da árvore de referência agrupa *X. campestris vesicatoria* com *X. oryzae* PXO99A,

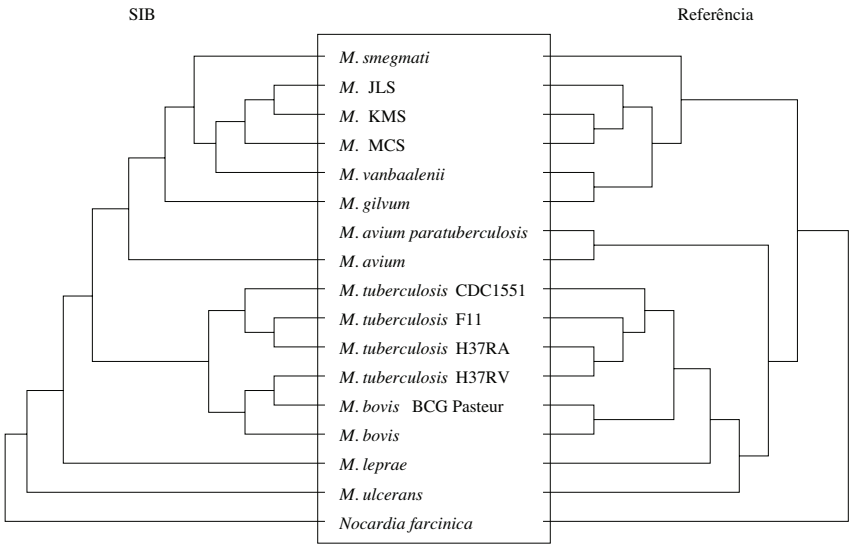


Figura 4.5: Comparação entre a árvore do grupo *Mycobacterium* produzida pelo SIB e a árvore proposta por Alam et al [2].

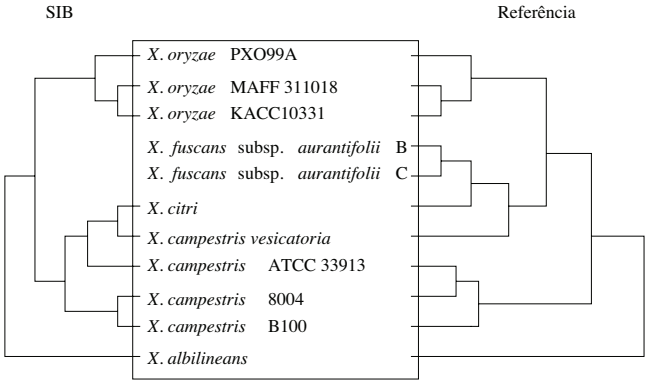


Figura 4.6: Comparação entre a árvore do grupo *Xanthomonas* produzida pelo SIB e a árvore proposta por Moreira et al [77].

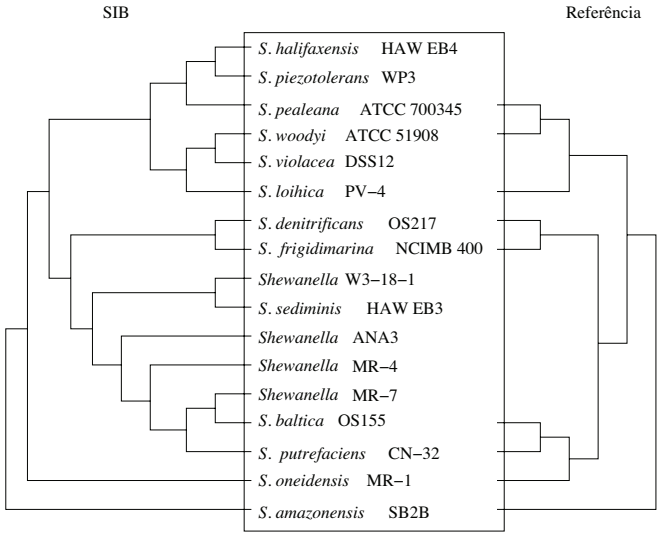


Figura 4.7: Comparação entre a árvore do grupo *Shewanella* produzida pelo SIB e a árvore proposta por Williams et al [99]

enquanto que o SIB agrupa *X. campestris vesicatoria* com as outras espécies do grupo *campestris*, em especial a espécie *X. campestris* ATCC 33913. Uma inspeção visual pelos *dotplots* leva à mesma conclusão do SIB: a quantidade de rearranjos entre *X. campestris vesicatoria* e *X. campestris* ATCC 33913 é menor que a quantidade de rearranjo entre ambas e *X. oryzae* PXO99A. Essa conclusão mostra que o SIB agrupa corretamente organismos quando é levado em conta apenas os eventos de rearranjo, mesmo que isso gere uma incongruência com outros métodos.

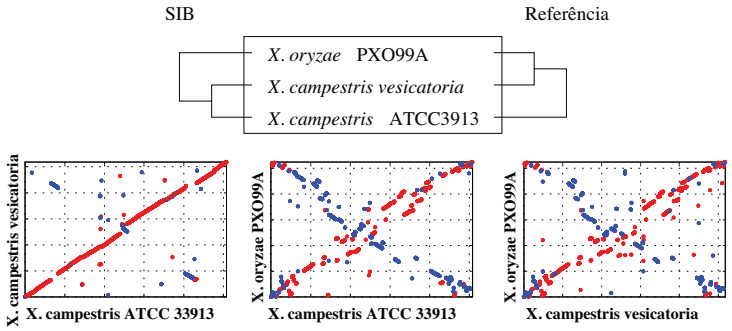


Figura 4.8: Exemplo de incongruência entre a árvore gerada pelo SIB e árvore de máxima verossimilhança. O agrupamento obtido pelo SIB é mais consistente com os *dotplots*.

Baseado nos resultados acima, é possível concluir que o SIB realiza um bom trabalho (apesar de não ser perfeito) em simular a evolução de eventos de reversões simétricas.

4.4 Considerações Finais sobre os Programas de Reconstrução

Nesta seção, será apresentada uma avaliação da reconstrução do genoma ancestral pelo programa MGR [15] em um cenário simulado criado pelo SIB. O programa MGR reconstrói o genoma ancestral assumindo que o único evento possível é a reversão. O critério da máxima parcimônia é usado da seguinte maneira: dados três genomas A, B e C, o ancestral comum *root* deve minimizar a soma do número de inversões entre *root* e A, B e C. O próposito desta seção é reforçar a necessidade de algoritmos que levem em consideração reversões simétricas, mostrando que o algoritmo de reversão não é capaz de gerar resultados condizentes com os observados em genomas reais.

O cenário evolutivo criado é descrito pela Figura 4.9. Existem neste cenário: 150 reversões simétricas entre *root* e A, 150 reversões entre *root* e B, 300 reversões entre *root* e C. A geração de C utilizou mais reversões que os demais por se tratar do *outgroup*.

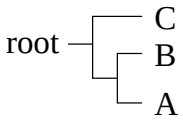


Figura 4.9: Cenário evolutivo criado para avaliar a reconstrução do genoma ancestral assumindo que o único evento possível é a reversão. A reconstrução foi feita com o programa MGR.

Criou-se 10 cenários diferentes para o caso descrito na Figura 4.9. Em média, a reconstrução do MGR requer um total de 380 reversões, em contraponto com as 600 reversões realmente necessárias para criar cada cenário. Isso já é uma primeira sugestão de que o evento de reversão irrestrito pode não ser um bom critério para cenários onde as reversões são majoritariamente simétricas.

A segunda sugestão de que o evento de reversão não é um bom critério é mais conclusiva que a primeira. A Figura 4.10 mostra dois grupos de *dotplots*. No primeiro grupo, são mostrados alinhamentos entre o ancestral obtido pelo MGR e os genomas A, B e C. É possível perceber que o padrão em "X" está deformado, sendo que o novo padrão mostrado não encontra paralelos em observações de genomas reais. Na mesma figura, abaixo do primeiro grupo de *dotplots*, estão os *dotplots* que deveriam ter sido obtidos pela reconstrução. A Figura 4.10 é representativa para os 10 cenários gerados.

Esses resultados mostram que MGR falha em reconstruir corretamente o ancestral da simulação. Partindo-se do pressuposto de que o SIB é um modelo fiel ao que ocorre na natureza, é possível concluir que MGR também não reconstruiria genomas ancestrais

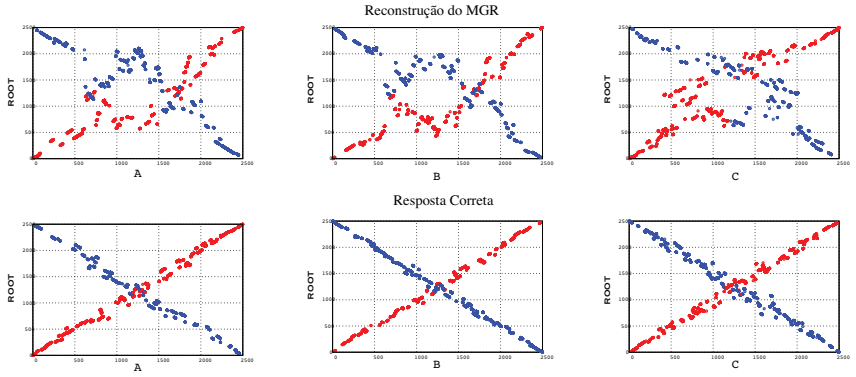


Figura 4.10: Na parte de cima estão os *dotplots* que resultam do alinhamento do ancestral reconstruído e dos três descendentes A, B and C. Na parte de baixo da figura são mostrados *dotplots* corretos, que deviam ter sido obtidos pela ferramenta de reconstrução.

reais com uma presença nítida do padrão em "X". Esse resultado mostra que o modelo embutido na ferramenta MGR (reversões irrestritas) não é adequado neste caso.

O SIB está disponível em http://www.ic.unicamp.br/∼zanoni/sib.

4.5 Publicações

O simulador SIB foi apresentado no artigo "*A simulation tool for the study of symmetric inversions in bacterial genomes*" (Ulisses Dias, Zanoni Dias e João C. Setubal) em outubro de 2010 na conferência RECOMB-CG'2010 (*RECOMB Satellite Workshop on Comparative Genomics*) realizada em Ottawa, Canadá [38]. Este artigo apresenta os resultados para as bactérias da família *Pseudomonadaceae* e dos gêneros *Xanthomonas* e *Shewanella*. A diferença principal entre o presente capítulo e o artigo é a adição das bactérias do gênero *Mycobacterium* neste capítulo.

Capítulo 5

Distância de Reversão Quase-Simétrica

Motivação: A Seção 4.4 do capítulo anterior mostra que algoritmos de reversão produzem resultados incorretos quando usados para calcular o genoma ancestral nos cenários em que o evento de reversão é predominantemente simétrico. Esses resultados reforçam a necessidade de se desenvolver algoritmos específicos para o problema da reversão simétrica.

Metodologia: Formalizou-se o conceito de reversões quase-simétricas. Para tanto, uma série de definições e funções importantes são formuladas com o intuito de fornecer um entendimento maior sobre o problema. A estratégia inicial consiste em isolar cada elemento da permutação e calcular o número de operações necessárias para posicioná-lo adequadamente. Essa abordagem permite identificar um limitante inferior e criar conjecturas sobre um limitante superior.

Resultados: Apesar de o problema das reversões simétricas continuar em aberto, o primeiro tratamento algorítmico foi apresentado. As noções apresentadas neste capítulo são a base para trabalhos futuros no contexto de algoritmos de aproximação e heurísticas. Um algoritmo guloso fornece uma sequência de reversões quase-simétricas para ordenar qualquer permutação de entrada. Além disso, algoritmos exatos foram apresentados para várias famílias de permutações.

5.1 Introdução

As reversões simétricas são o mecanismo mais provável para explicar o padrão em "X", comum ao alinhar dois genomas de organismos relacionados. O Capítulo 1 introduz as reversões simétricas e o modelo evolutivo capaz de explicar o padrão em "X".

O Capítulo 4 apresenta uma ferramenta de simulação para reversões simétricas. Na Seção 4.4, essa ferramenta foi usada para mostrar a necessidade de algoritmos específicos para reversões simétricas, mostrando que o algoritmo de reversão não é capaz de gerar resultados condizentes com os observados em genomas reais.

Recentemente, Ohlebusch e co-autores [82] produziram um algoritmo de ordenação para um problema inspirado em reversões simétricas. O algoritmo produz resultado em tempo polinomial, o que permite a utilização dessa abordagem no cálculo do genoma ancestral, como proposto originalmente pelos autores. Entretanto, o problema definido pelos autores não garante que, dados dois genomas arbitrários, seja sempre possível obter uma sequência de reversões simétricas que transforme um genoma no outro. Dessa forma, o algoritmo só pode ser usado para um grupo restrito de instâncias.

No presente capítulo, é proposto um novo problema baseado em reversões simétricas sem a limitação do trabalho de Ohlebusch e co-autores. Para este problema, são propostos limitantes inferiores e superiores, heurísticas para ordenação e alguns resultados para famílias de permutações.

O capítulo é estruturado da seguinte forma: a Seção 5.2 apresenta a definição do novo problema; a Seção 5.3 apresenta um limitante inferior e define conjecturas sobre o limitante superior; a Seção 5.4 apresenta os primeiros algoritmos para resolver o problema, tais algoritmos fornecem resposta para qualquer permutação, mas não há fator de aproximação que garanta uma solução próxima da distância correta; a Seção 5.5 apresenta algoritmos exatos para algumas classes e famílias de permutações.

5.2 Definições

Um genoma é representado como uma permutação: $\pi = (\pm\pi_1 \ \pm\pi_2 \ \ldots \ \pm\pi_n)$, $\pi_i \in \mathbb{N}$, $1 \leq \pi_i \leq n$ e $i \neq j \leftrightarrow \pi_i \neq \pi_j$. Uma permutação pode ser utilizada para representar tanto genomas lineares quanto genomas circulares. No presente capítulo, os genomas alvo são bacteriais circulares. Por convenção, π_1 é o elemento observado após a origem de replicação.

Três funções podem ser aplicadas a qualquer permutação π:

- **Position:** $p(\pi, i) = k \leftrightarrow |\pi[k]| = i$, $p(\pi, i) \in \{1,2,\ldots,n\}$.
- **Sign:** $s(\pi, i) = k \leftrightarrow \pi[p(\pi, i)] = ki$, $s(\pi, i) \in \{-1, +1\}$.

- **Slice:** $slice(\pi, i) = min\{p(\pi, i), n - p(\pi, i) + 1\}$, $slice(\pi, i) \in \{1,2,\ldots,\lceil \frac{n}{2} \rceil\}$.

A Figura 5.1 mostra duas permutações circulares, a origem de replicação é representada pelo número "0" e a ordem de leitura está no sentido horário. A figura indica os valores de **Slice** para os elementos das permutações.

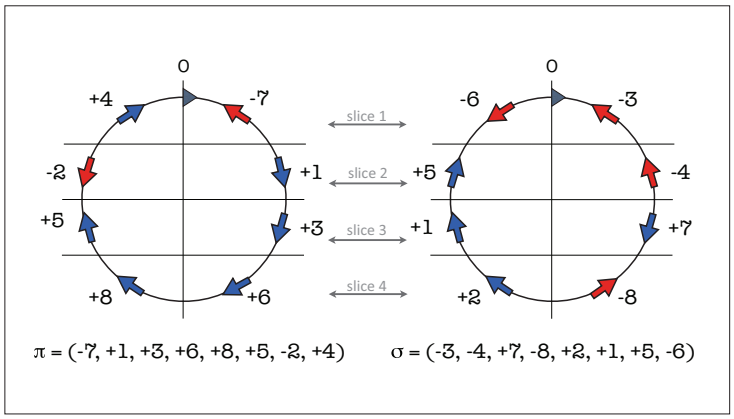
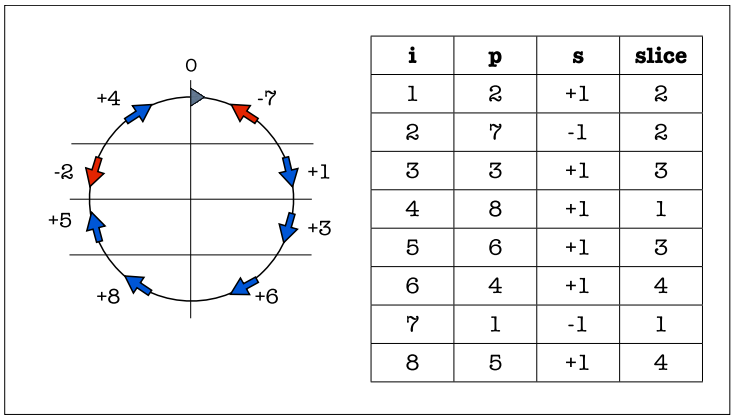


Figura 5.1: Representação de genomas. A origem de replicação é representada pelo número "0" e a ordem de leitura está no sentido horário.

A Figura 5.2 mostra um exemplo das funções Position (p), Sign (s) e Slice ($slice$) para a permutação $(-7\ +1\ +3\ +6\ +8\ +5\ -2\ +4)$.



i	p	s	slice
1	2	+1	2
2	7	-1	2
3	3	+1	3
4	8	+1	1
5	6	+1	3
6	4	+1	4
7	1	-1	1
8	5	+1	4

Figura 5.2: Exemplo das funções Position (p), Sign (s) e Slice ($slice$) para a permutação $(-7\ +1\ +3\ +6\ +8\ +5\ -2\ +4)$.

Seja $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ um genoma arbitrário, uma reversão ρ é uma operação em que a ordem de um segmento da permutação é invertida e a orientação de todos os elementos

é modificada. Dessa forma, ao se aplicar a reversão $\rho(i, j)$, $1 \leq i < j \leq n$, obtém-se: $\pi\rho = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_{i+1}, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$. Uma reversão simétrica $\ddot{\rho}$ é uma reversão em que $\ddot{\rho}(i) = \rho(i+1, n-i)$, para todo $0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor$ (Figura 5.3).

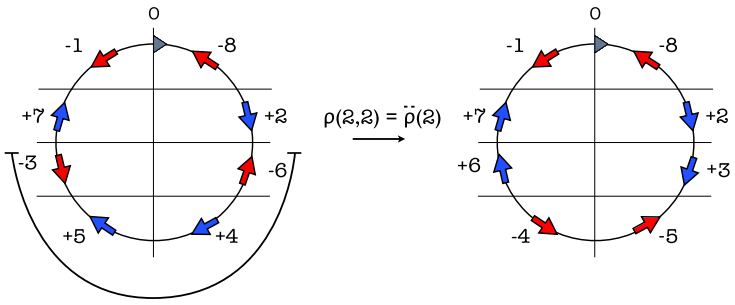


Figura 5.3: Reversão simétrica

Dadas duas permutações π e σ de mesmo tamanho n, é possível encontrar uma sequência de reversões simétricas que transforma π em σ se, e somente se, $slice(\pi, i) = slice(\sigma, i)$, para todo $1 \leq i \leq n$. Da mesma forma, uma permutação arbitrária π pode ser ordenada se, e somente se, $slice(\pi, i) = slice(\iota, i)$, onde $\iota = (1\ 2\ \ldots\ n)$.

O Algoritmo 3 verifica se uma permutação é ordenável por reversões simétricas e, caso seja verdade, a distância de reversão simétrica é calculada. Para esse cálculo, o algoritmo acessa todos os slices uma única vez e verifica se os elementos estão na posição correta.

Algoritmo 3: Reversões Simétricas

```
Data: π
k ← 0
for i ← 1 to ⌈n/2⌉ do
    if π[i] = −(n − i + 1) and π[n − i + 1] = −i then
        k ← k + 1
    end
    else
        if π[i] ≠ +i or π[n − i + 1] ≠ +(n − i + 1) then
            return ERRO
        end
    end
end
return k
```

A definição do problema das reversões simétricas apresenta duas limitações:

1. O número de permutações com sinal é $2^n n!$. Entretanto, o número de permutações ordenáveis usando reversões simétricas é $2^{\lceil \frac{n}{2} \rceil}$.

2. Espera-se que as reversões não sejam perfeitamente simétricas em genomas reais.

Essas limitações motivam a definição do problema proposto neste capítulo, denominado **reversões quase-simétricas**.

Definição 5.1 *Reversões quase-simétricas:* $\overline{\rho}(i,j) = \rho(i+1, n-j)$, $0 \leq i,j \leq n-1$, $|i-j| \leq 1$.

As reversões quase-simétricas são um caso especial das reversões k-assimétricas:

Definição 5.2 *Reversões k-assimétricas:* $\hat{\rho}(i,j) = \rho(i+1, n-j)$, $0 \leq i,j \leq n-1$, $|i-j| \leq k$, $k \geq 1$.

Todas as permutações possíveis podem ser ordenadas usando reversões k-assimétricas e reversões quase-simétricas.

5.3 Limitantes

O limitante inferior para o problema das reversões quase-simétricas leva em consideração o número de operações necessárias para se posicionar um único elemento. Para tanto, é contabilizado o número mínimo de slices que cada elemento percorre até a posição correta.

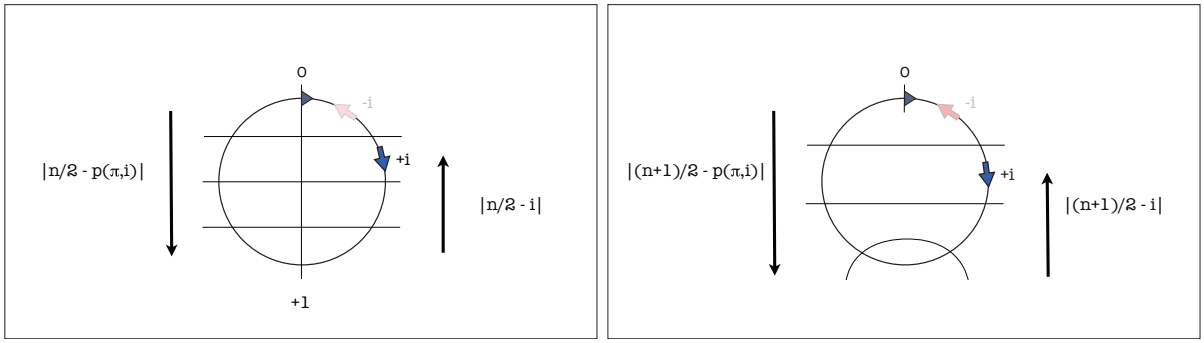
Definição 5.3 *Diz-se que uma reversão quase-simétrica* $\overline{\rho}(i,j)$ ***age*** *no elemento* k *de* π *se* $i \leq p(\pi,k) \leq j$.

Definição 5.4 *Define-se* $d_\pi[i]$ *como o número mínimo de reversões quase-simétricas que devem agir no elemento* i *para ordenar a permutação* π.

Lema 5.1 *Seja* $\overline{\rho}$ *uma reversão quase-simétrica. Se* $slice(\pi, j) = k$, *então* $slice(\pi\overline{\rho}, j) \in \{k-1, k, k+1\}$.

Lema 5.2 *Se* $slice(\pi, i) = j$ *e* $slice(\iota, i) = k$, *então um total de* $d_\pi[i] \geq |j-k|$ *reversões quase-simétricas devem agir no elemento* i *para posicioná-lo adequadamente.*

O Lema 5.2 permite calcular o número mínimo de movimentos necessários para movimentar um elemento até a sua posição. Entretanto, esse lema não considera o sinal que o elemento terá ao ser posicionado. Quando um elemento está com o sinal negativo na posição correta, é preciso movimentá-lo até a região diametralmente oposta à origem da permutação e depois trazê-lo de volta. Dessa forma, a melhor estratégia é verificar o sinal que o elemento terá ao final do posicionamento antes de realizar as reversões. Os Lemas 5.3 e 5.4 fornecem uma equação para o número de operações necessárias para realizar todo esse procedimento em permutações de tamanhos pares e ímpares, respectivamente. A Figura 5.4 ilustra ambos os lemas.



(a) Permutações de tamanho par (b) Permutações de tamanho ímpar

Figura 5.4: Número de reversões simétricas necessárias para posicionar um elemento quando o caminho direto resultaria em um posicionamento com o sinal negativo. Dois casos são ilustrados, um para π de tamanho par e outro para π de tamanho ímpar.

Lema 5.3 *Seja π um genoma com número par de genes ($n = 2k$), $d_\pi[i] = |\frac{n}{2} - p(\pi, i)| + |\frac{n}{2} - i| + 1$ se um dos itens a seguir for verdade.*

- $i \leq \frac{n}{2}$, $p(\pi, i) \leq \frac{n}{2}$ *e* $s(\pi, i) = -1$
- $i \leq \frac{n}{2}$, $p(\pi, i) > \frac{n}{2}$ *e* $s(\pi, i) = +1$
- $i > \frac{n}{2}$, $p(\pi, i) \leq \frac{n}{2}$ *e* $s(\pi, i) = +1$
- $i > \frac{n}{2}$, $p(\pi, i) > \frac{n}{2}$ *e* $s(\pi, i) = -1$

Lema 5.4 *Seja π um genoma com um número ímpar de genes ($n = 2k + 1$), $d_\pi[i] = |\frac{(n+1)}{2} - p(\pi, i)| + |\frac{(n+1)}{2} - i|$ se um dos itens a seguir for verdade.*

- $i \leq \frac{n+1}{2}$, $p(\pi, i) \leq \frac{n+1}{2}$ *e* $s(\pi, i) = -1$
- $i \leq \frac{n+1}{2}$, $p(\pi, i) > \frac{n+1}{2}$ *e* $s(\pi, i) = +1$
- $i > \frac{n+1}{2}$, $p(\pi, i) \leq \frac{n+1}{2}$ *e* $s(\pi, i) = +1$
- $i > \frac{n+1}{2}$, $p(\pi, i) > \frac{n+1}{2}$ *e* $s(\pi, i) = -1$

Os Lemas 5.3 e 5.4 permitem calcular a distância para ordenar um dado elemento sem levar em conta os demais. Entretanto, para ordenar a permutação, uma reversão quase-simétrica necessária para posicionar um dado elemento j pode afetar um outro elemento i sem contribuir para a ordenação deste. O Lema 5.5 lida com a situação em que o elemento i já está corretamente posicionado.

Lema 5.5 *Seja i um elemento com sinal positivo em sua posição correta ($\pi_i = i$). Se existe um elemento j tal que $slice(\pi, j) < slice(\pi, i)$ e $d_\pi[j] > 0$, então no mínimo duas reversões quase-simétricas devem agir em i para ordenar π, o que significa que $d_\pi[i] \geq 2$.*

As funções $Lower(\pi)$ e $Upper(\pi)$ nas Definições 5.5 e 5.6 usam a distância $d_\pi[i]$.

Definição 5.5 $Lower(\pi) = max_{1 \leq i \leq n} d_\pi[i]$.

Definição 5.6 $Upper(\pi) = \sum_{1 \leq i \leq n} d_\pi[i]$.

A função *Lower* é um limitante inferior para o problema da distância de reversões quase-simétricas. Por outro lado, nada se pode afirmar sobre a função *Upper*, mas uma busca exaustiva usando todos as permutações π, para $|\pi| \leq 9$, permite conjecturar que *Upper* seja uma limitante superior.

Lema 5.6 $d(\pi) \geq Lower(\pi)$

Conjectura 1 $d(\pi) \leq Upper(\pi)$

A Figura 5.5 ilustra os valores da função *Lower* e *Upper* para a permutação $\pi = (-7\ -1\ +3\ +6\ +8\ -5\ -2\ +4)$.

Conjectura 2 *Seja π um genoma com n genes, então $1 \leq d(\pi)/Lower(\pi) \leq n$.*

Conjectura 3 *Seja π um genoma com n genes, então $1 \leq Upper(\pi)/d(\pi) \leq n$.*

As Conjecturas 2, 3 são válidas para todos os genomas de tamanho $n \leq 9$.

Para avaliar se *Lower* e *Upper* são próximos da distância real de uma dada permutação, calculou-se os valores médios de $Lower(\pi)$, $Upper(\pi)$ e $d(\pi)$ para todas as permutações π com tamanho menor ou igual a 9. O diâmetro $D(n)$ foi calculado para $n \leq 10$. Os valores são mostrados na Tabela 5.1. Observa-se que em média tanto *Lower* quanto *Upper* se afastam da distância exata à medida que n aumenta. A distribuição das distâncias exatas para todas as permutações com 10 ou menos elementos é mostrada na Figura 5.6.

5.4 Algoritmos de Ordenação

O Algoritmo 4 fornece uma lista de reversões quase-simétricas que ordenam qualquer permutação de entrada. Esse algoritmo posiciona gradativamente os elementos dependendo do slice. Assim, sendo i e j dois elementos de π com $slice(\pi, i) < slice(\pi, j)$, o elemento i será posicionado antes do elemento j.

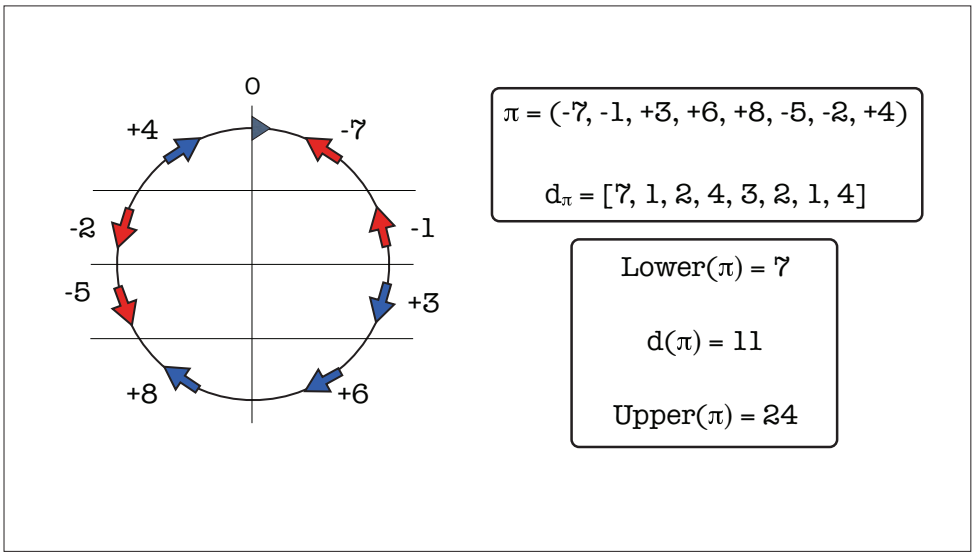


Figura 5.5: Funções *Lower* e *Upper* aplicadas à permutação $\pi = (-7\ -1\ +3\ +6\ +8\ -5\ -2\ +4)$.

			Média		
N	**Permutações**	**Diam**	**Lower**	**Dist**	**Upper**
2	8	3	1,25	1,50	2,00
3	48	5	2,13	2,96	4,46
4	384	6	2,90	4,26	7,48
5	3.840	8	3,71	5,68	11,09
6	46.080	10	4,52	7,02	15,33
7	645.120	11	5,36	8,41	20,21
8	10.321.920	13	6,20	9,76	25,75
9	185.794.560	15	7,06	11,15	31,94
10	3.715.891.200	16	–	12,51	–

Tabela 5.1: Valores médios de $Lower(\pi)$, $d(\pi)$ e $Upper(\pi)$ para todas as permutações π com tamanho menor ou igual a 10. A coluna **Diam** mostra o valor do diâmetro para um dado tamanho de permutação.

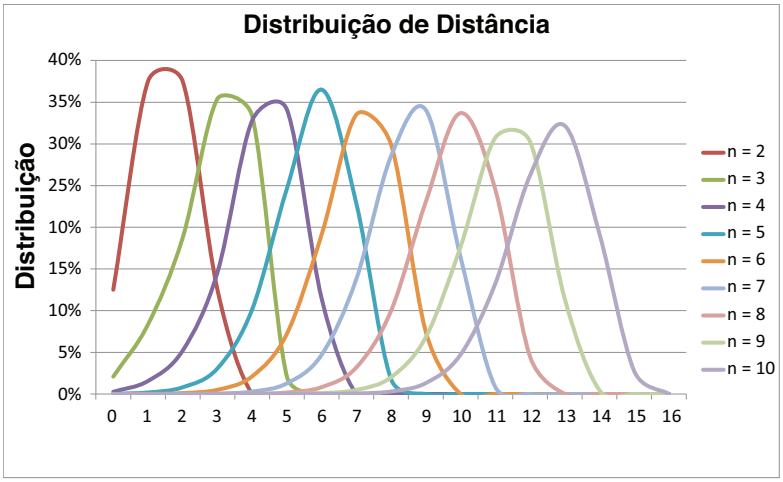


Figura 5.6: Distribuição das distâncias exatas para todas as permutações com 10 ou menos elementos

Algoritmo 4: Algoritmo Básico

Data: π
for $i \leftarrow 1$ **to** $\lceil \frac{n}{2} \rceil$ **do**
    Coloque o elemento i em sua posição correta
    Coloque o elemento $n - i + 1$ em sua posição correta
end

Em uma permutação arbitrária π de tamanho n, os elementos i e $n-i+1$ compartilham o mesmo slice para $1 \leq i \leq \lceil \frac{n}{2} \rceil$. O algoritmo básico segue, por convenção, a estratégia de ordenar o elemento i antes do elemento $n - i + 1$. Entretanto, escolher a ordem em que os elementos do mesmo nível serão ordenados diminui o número total de operações para ordenar a permutação, o que sugere uma simples melhoria para o Algoritmo 4: verificar os dois caminhos e selecionar aquele com o menor número de operações. Essa estratégia é implementada no Algoritmo 5.

Algoritmo 5: Algoritmo Básico com Otimização de Slice

Data: π
for $i \leftarrow 1$ **to** $\lceil \frac{n}{2} \rceil$ **do**
  Escolha a melhor opção dentre (menor número de reversões):
  - Coloque o elemento i em sua posição correta
    + Coloque o elemento $n - i + 1$ em sua posição correta
  - Coloque o elemento $n - i + 1$ em sua posição correta
    + Coloque o elemento i em sua posição correta

end

A complexidade de tempo dos Algoritmos 4 e 5 é $O(n^3)$, pois é necessário um tempo $O(n)$ para posicionar cada elemento, sendo que ao todo são necessárias $O(n^2)$ reversões simétricas de acordo com o limitante superior (Definição 5.6). A verificação adicional realizada pelo Algoritmo 5 afeta apenas as constantes de proporcionalidade.

Os Algoritmos 4 e 5 foram executados para todas as permutações π com $|\pi| \leq 9$. Os resultados são mostrados nas Tabelas 5.2 e 5.3. Ambas as tabelas mostram o valor máximo e médio das distâncias absolutas fornecidas pelos algoritmos. Conclui-se pelas tabelas que não houve mudança de um algoritmo para o outro no valor máximo de operações usadas, apesar de em média a otimização de slice ser positiva.

Adicionalmente, calculou-se a razão $\frac{d_{alg}(\pi)}{d(\pi)}$, onde $d_{alg}(\pi)$ é a distância fornecida pelo algoritmo e $d(\pi)$ é a distância real, para todas as permutações π. O valor médio e o valor máximo dessa razão foram adicionados na tabela. Percebe-se que os resultados obtidos com a otimização de slice são mais próximos da distância real. A porcentagem de permutações em que os algoritmos retornam a distância exata são mostrados na coluna **Exatos**. Conclui-se que o Algoritmo 5.3 fornece regularmente resultados melhores.

Uma estratégia gulosa foi projetada para o problema da ordenação por reversões quase-simétricas. A estratégia se baseia na minimização da função $h(n) = \min\{d[i], d[n-i-1]\} + d[i] + d[n-i+1]$, para toda $\overline{\rho}$ aplicável a π (Algoritmo 6).

Algoritmo 6: Algoritmo Guloso Básico

Data: π
$k \leftarrow 0$
while $\pi \neq \iota$ **do**
  $\overline{\rho} \leftarrow \overline{\rho}'$ que minimiza $\min\{d[i], d[n-i-1]\} + d[i] + d[n-i+1]$, para $i \leq i \leq \lceil \frac{n}{2} \rceil$, em $\overline{\rho}'\pi$ para todo $\overline{\rho}'$ aplicável a π
  $\pi \leftarrow \overline{\rho}\pi$
  $k \leftarrow k + 1$
end
return k

	Algoritmo Básico				
	Operações		Razão		Exatos
N	MAX	AVG	MAX	AVG	
2	3	1,50	1,00	1,00	100,00%
3	7	3,50	2,50	1,18	66,67%
4	10	5,33	3,00	1,25	48,96%
5	17	8,50	4,50	1,50	16,98%
6	21	11,23	5,00	1,60	8,20%
7	31	15,50	6,50	1,84	1,87%
8	36	19,16	7,00	1,96	0,64%
9	49	24,50	8,50	2,20	0,12%

Tabela 5.2: Performance do Algoritmo 4. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata

O algoritmo guloso possui uma complexidade de tempo superior aos algoritmos básicos. Para calcular a função a ser minimizada, é necessário tempo $O(n)$ para todo $i \leq i \leq \lceil \frac{n}{2} \rceil$. Dessa forma, para cada reversão, utiliza-se um tempo $O(n^3)$. Como o número de reversões é da ordem $O(n^2)$, o tempo de execução do algoritmo inteiro é da ordem de $O(n^5)$.

O algoritmo guloso produz resultados superiores aos dois algoritmos básicos. A Tabela 5.4 apresenta esses resultados para todas as permutações π, para $|\pi| \leq 9$.

Para avaliar o comportamento dos algoritmos com permutações maiores, 100 permutações ao acaso foram escolhidas para cada n no conjunto $\{5, 10, 15, \ldots, 100\}$. As distâncias médias obtidas com cada um dos algoritmos implementados, bem como a média dos valores de *Upper* e *Lower* são mostrados na Figura 5.7.

5.5 Famílias e Classes de Permutações

Apesar de ainda não haver um algoritmo exato para o problema da distância de ordenação por reversões quase-simétricas para permutações genéricas, é possível definir uma série de classes e famílias com o intuito de elaborar algoritmos específicos.

Uma família é um grupo de permutações que compartilham uma mesma regra, sendo que para todo n pertencente ao conjunto de números naturais, existe uma permutação com n elementos pertencente à família.

As famílias foram selecionadas por possuírem características que as tornam difíceis de ordenar usando os algoritmos da Seção 5.4, sendo que a solução dessas famílias fornece

	Algoritmo Básico com Otimização de Slice				
	Operações		Razão		Exatos
N	**MAX**	**AVG**	**MAX**	**AVG**	
2	3	1,50	1,00	1,00	100,00%
3	7	3,33	2,00	1,13	75,00%
4	10	5,08	2,33	1,19	56,77%
5	17	7,86	2,83	1,38	23,36%
6	21	10,55	3,50	1,50	11,78%
7	31	14,22	4,50	1,69	3,12%
8	36	17,87	5,00	1,83	1,13%
9	49	22,42	6,25	2,01	0,23%

Tabela 5.3: Performance do Algoritmo 5. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata

novas informações sobre o problema com um todo.

Uma classe é um grupo mais abrangente e pode conter mais de uma permutação de mesmo tamanho n. Algumas classes são generalizações de várias famílias.

5.5.1 Famílias de permutações

A Conjectura 4 agrega limitantes para 10 famílias de permutação.

Conjectura 4 *Para cada família $F_k(n)$, conjectura-se que a distância $d(F_k(n))$ corresponde à distância exata.*

Família de Permutação	***Distância***
$F_1(n) = [-1, +2, +3, \ldots, +n]$	$d(F_1(n)) = 2\lfloor\frac{n-1}{2}\rfloor + 1$, *para* $n \geq 1$.
$F_2(n) = [+(n-1), \ldots, +2, +1, -n]$	$d(F_2(n)) = 2\lfloor\frac{n-1}{2}\rfloor + 1$, *para* $n \geq 2$.
$F_3(n) = [+n, -1, -2, \ldots, -(n-1)]$	$d(F_3(n)) = 2\lceil\frac{n-1}{2}\rceil$, *para* $n \geq 3$.
$F_4(n) = [-1, -2, \ldots, -(n-1), -n]$	$d(F_4(n)) = 2\lceil\frac{n}{2}\rceil$, *para* $n \geq 2$.
$F_5(n) = [+n, +(n-1), \ldots, +2, +1]$	$d(F_5(n)) = 2\lfloor\frac{n}{2}\rfloor + 1$, *para* $n \geq 2$.
$F_6(n) = [+1, -2, \ldots, -(n-1), -n]$	$d(F_6(n)) = 2\lceil\frac{n}{2}\rceil + 1$, *para* $n \geq 5$.
$F_7(n) = [+n, +(n-1), \ldots, +2, -1]$	$d(F_7(n)) = 2\lfloor\frac{n}{2}\rfloor + 2$, *para* $n \geq 5$.
$F_8(n) = [-1, +2, \ldots, +(n-1), -n]$	$d(F_8(n)) = n + 2$, *para* $n \geq 5$.
$F_9(n) = [+n, -1, +(n-2), -3, \ldots, +2, -(n-1)]$ *(n par)*	$d(F_9(n)) = \lceil\frac{3n}{2}\rceil - 1$, *para* $n \geq 4$.
$F_9(n) = [+n, -2, +(n-2), -4, \ldots, -(n-1), +1]$ *(n ímpar)*	$d(F_9(n)) = \lceil\frac{3n}{2}\rceil - 1$, *para* $n \geq 4$.
$F_{10}(n) = [-1, +2, -3, +4, \ldots, (-1)^n n]$	$d(F_{10}(n)) = \lceil\frac{3n}{2}\rceil$, *para* $n \geq 5$.

As conjecturas referentes às famílias F_1 a F_{10} são verdadeiras para todos as permutações π, tais que $|\pi| \leq 10$. Os valores conjecturados, que coincidem com os valores

	Algoritmo Guloso				
	Operações		Razão		Exatos
N	**MAX**	**AVG**	**MAX**	**AVG**	
2	3	1,50	1,00	1,00	100,00%
3	5	3,08	1,25	1,04	87,50%
4	8	4,59	1,75	1,08	76,30%
5	12	6,85	2,75	1,21	41,43%
6	15	8,89	2,75	1,27	25,88%
7	21	11,70	3,80	1,39	10,26%
8	25	14,10	3,60	1,44	5,18%
9	32	17,41	4,75	1,57	1,62%

Tabela 5.4: Performance do Algoritmo 6. A tabela mostra o valor máximo (**Operações-Max**) e o médio (**Operações-AVG**) das distâncias fornecidas pelo algoritmo, bem como o valor máximo (**Razão-Max**) e o médio (**Razão-AVG**) da razão de aproximação desses valores em relação à distância exata. A última coluna fornece a porcentagem de permutações em que o algoritmo retorna a distância exata

reais de distância, são mostrados na Tabela 5.5. A última coluna mostra o valor do diâmetro para cada tamanho N, o que permite observar que as famílias se aproximam gradativamente do diâmetro.

Para cada família foi gerado um algoritmo de ordenação. Entretanto, dado a semelhança com os outros algoritmos mostrados, os algoritmos para as famílias F_6, F_7, F_8 e F_{10} não são mostrados.

Algoritmo 7: Ordena família 1

Data: π, n
$bool \leftarrow true$
for $i \leftarrow 1$ **to** $\lfloor \frac{n-1}{2} \rfloor$ **do**
  if *bool* **then** $\pi \leftarrow \rho(1, n-1)\pi$ **else** $\pi \leftarrow \rho(2, n)\pi$
  $bool \leftarrow not(bool)$
end
if $n \mod 2 = 1$ **then** $\pi \leftarrow \rho(\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil)\pi$ **else**
  if $\lfloor \frac{n-1}{2} \rfloor \mod 2 = 1$ **then** $\pi \leftarrow \rho(\frac{n}{2}+1, \frac{n}{2}+1)\pi$ **else** $\pi \leftarrow \rho(\frac{n}{2}, \frac{n}{2})\pi$
end
for $i \leftarrow 1$ **to** $\lfloor \frac{n-1}{2} \rfloor$ **do**
  if *bool* **then** $\pi \leftarrow \rho(2, n)\pi$ **else** $\pi \leftarrow \rho(1, n-1)\pi$
  $bool \leftarrow not(bool)$
end
return $\pi, 2\lfloor \frac{n-1}{2} \rfloor + 1$

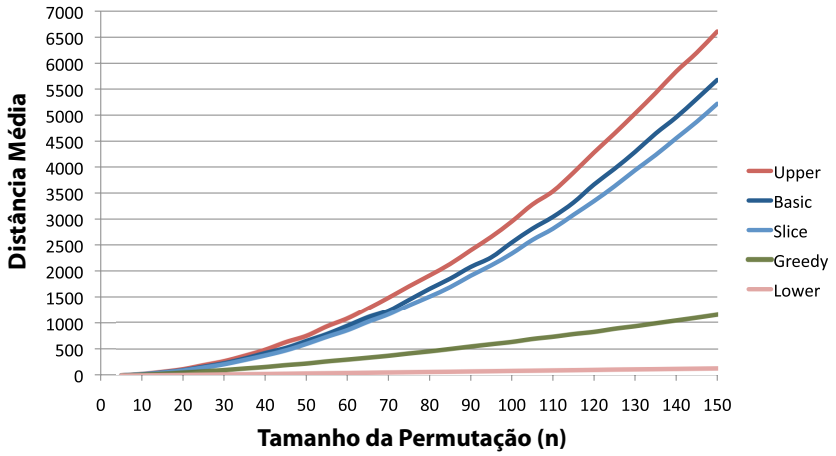


Figura 5.7: Distâncias médias (e limitantes) para permutações de tamanho menor ou igual a 100. Nos testes foram geradas 100 permutações aleatórias para cada valor n no conjunto $\{5, 10, 15, 20, \ldots, 100\}$.

Algoritmo 8: Ordena família 2

Data: π, n
if $n \mod 2 = 0$ **then**
  $\pi \leftarrow \rho(2, n)\pi$
end
else
  $\pi \leftarrow \rho(1, n)\pi$
end
for $i \leftarrow 1$ **to** $\lfloor \frac{n-1}{2} \rfloor$ **do**
  $\pi \leftarrow \rho(1, n-1)\pi$
  $\pi \leftarrow \rho(2, n)\pi$
end
return $\pi, 2\lfloor \frac{n-1}{2} \rfloor + 1$

N	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**D**
1	1	1	0	1								**1**
2	1	1	2	2	3							**3**
3	3	3	2	4	3			5				**5**
4	3	3	4	4	5	5		6	5			**6**
5	5	5	4	6	5	7	6	7	7	8	7	**8**
6	5	5	6	6	7	7	8	8	8	9	10	**10**
7	7	7	6	8	7	9	8	9	10	11	10	**11**
8	7	7	8	8	9	9	10	10	11	12	13	**13**
9	9	9	8	10	9	11	10	11	13	14	13	**15**
10	9	9	10	10	11	11	12	12	14	14	16	**16**

Tabela 5.5: Lista de distâncias fornecidas pelas conjecturas sobre as famílias F_1 a F_{10}. Todos os valores conjecturados na tabela coincidem com os valores exatos de distância para $N \leq 10$. A última coluna mostra o valor do diâmetro para cada valor de N.

Algoritmo 9: Ordena família 3

Data: π, n
if $n \mod 2 = 0$ **then**
  $\pi \leftarrow \rho(1, n)\pi$
end
else
  $\pi \leftarrow \rho(1, n-1)\pi$
end
$\pi \leftarrow \rho(2, n)\pi$
for $i \leftarrow 1$ **to** $\lceil \frac{n-1}{2} \rceil - 1$ **do**
  $\pi \leftarrow \rho(1, n-1)\pi$
  $\pi \leftarrow \rho(2, n)\pi$
end
return $\pi, 2\lceil \frac{n-1}{2} \rceil$

Algoritmo 10: Ordena família 4

Data: π, n
if $n \mod 2 = 0$ **then**
  $\pi \leftarrow \rho(1, n-1)\pi$
end
else
  $\pi \leftarrow \rho(1, n)\pi$
end
$\pi \leftarrow \rho(2, n)\pi$
for $i \leftarrow 1$ **to** $\lceil \frac{n}{2} \rceil - 1$ **do**
  $\pi \leftarrow \rho(1, n-1)\pi$
  $\pi \leftarrow \rho(2, n)\pi$
end
return $\pi, 2\lceil \frac{n}{2} \rceil$

Algoritmo 11: Ordena família 5

Data: π, n
if $n \mod 2 = 0$ **then**
  $\pi \leftarrow \rho(1, n)\pi$
end
else
  $\pi \leftarrow \rho(2, n)\pi$
end
for $i \leftarrow 1$ **to** $\lfloor \frac{n}{2} \rfloor$ **do**
  $\pi \leftarrow \rho(1, n-1)\pi$
  $\pi \leftarrow \rho(2, n)\pi$
end
return $\pi, 2\lfloor \frac{n}{2} \rfloor + 1$

Algoritmo 12: Ordena família 9

Data: π, n
if $n \mod 2 = 0$ **then**
  $\pi \leftarrow \rho(1, n)\pi$
  for $i \leftarrow 0$ **to** $\lfloor \frac{n}{2} \rfloor - 2$ **do**
    $\pi \leftarrow \rho(\lfloor \frac{n}{2} \rfloor - i, \lfloor \frac{n}{2} \rfloor + i)\pi$
  end
  $[\pi, d] \leftarrow OrdenaFamilia2(\pi, n)$
end
else
  for $i \leftarrow 0$ **to** $\lfloor \frac{n}{2} \rfloor - 1$ **do**
    $\pi \leftarrow \rho(\lceil \frac{n}{2} \rceil - i, \lceil \frac{n}{2} \rceil + i)\pi$
  end
  $[\pi, d] \leftarrow OrdenaFamilia5(\pi, n)$
end
return $\pi, d + \lfloor \frac{n}{2} \rfloor$

Os algoritmos propostos para as família ordenam a permutação com o número conjecturado de reversões quase-simétricas. Para avaliar numericamente os benefícios trazidos por esses algoritmos sobre o algoritmo básico e o algoritmo guloso, as seguintes métricas foram calculadas:

- **Basic**: resultado fornecido pelo algoritmo básico (Algoritmo 4).
- **Greedy**: resultado fornecido pelo algoritmo guloso (Algoritmo 6).
- **Lower**: limitante inferior ($Lower(F_k(n))$).
- **Upper**: limitante superior ($Upper(F_k(n))$).
- **Conjecture**: resultado obtido pelo algoritmo específico para cada família. Cada algoritmo ordena usando o número de reversões quase-simétricas indicado pela conjectura sobre a distância da família $F_K(n)$.

A Figura 5.8 mostra os gráficos desses valores para todas as famílias com $N \leq 40$. Nesses gráficos, percebe-se a melhoria obtida pelas conjecturas sobre o algoritmo básico e o algoritmo guloso. Na maioria dos casos, a conjectura se iguala ao limitante inferior.

5.5.2 Classes de Permutações

A Conjectura 5 apresenta os resultados referentes às classes de permutação.

Conjectura 5 *Sejam*

- $C_1(n,k) = \iota \cdot \rho(1,k)$, *para* $1 \leq k \leq n$.
- $C_2(n,i,j) = \iota \cdot \rho(i,j)$, *para* $1 \leq i \leq j \leq n$.
- $C_3(n,i,j) = \iota \cdot \rho(1,k) \cdot \rho(k+1,n) \cdot \rho(1,n)$, *para* $1 \leq k < n$.
- $C_4(n) = \{\iota \cdot \rho(i_1,i_1) \cdot \rho(i_2,i_2) \cdot \ldots \cdot \rho(i_r,i_r) | 1 \leq i_k \leq n,\ 1 \leq k \leq r,\ i_p \neq i_q,\ 1 \leq p < q \leq r, 1 \leq r \leq n\}$.
- $C_5(n) = \{\iota \cdot \rho(i_1,j_1) \cdot \rho(i_2,j_2) \cdot \ldots \cdot \rho(i_r,j_r) | 1 \leq i_k \leq j_k \leq n, 1 \leq k \leq r, [i_p,j_p] \cap [i_q,j_q] = \emptyset,\ 1 \leq p < q \leq r, 1 \leq r \leq n\}$.

classes de permutação. Então,

- $d(C_1(n,k)) = 2\lfloor \frac{n-k}{2} \rfloor + 1$, *para* $1 \leq k \leq n$.
- $d(C_2(n,i,j)) = 2\lfloor \frac{|n-i-j+1|}{2} \rfloor + 1$, *para* $1 \leq i \leq j \leq n$.

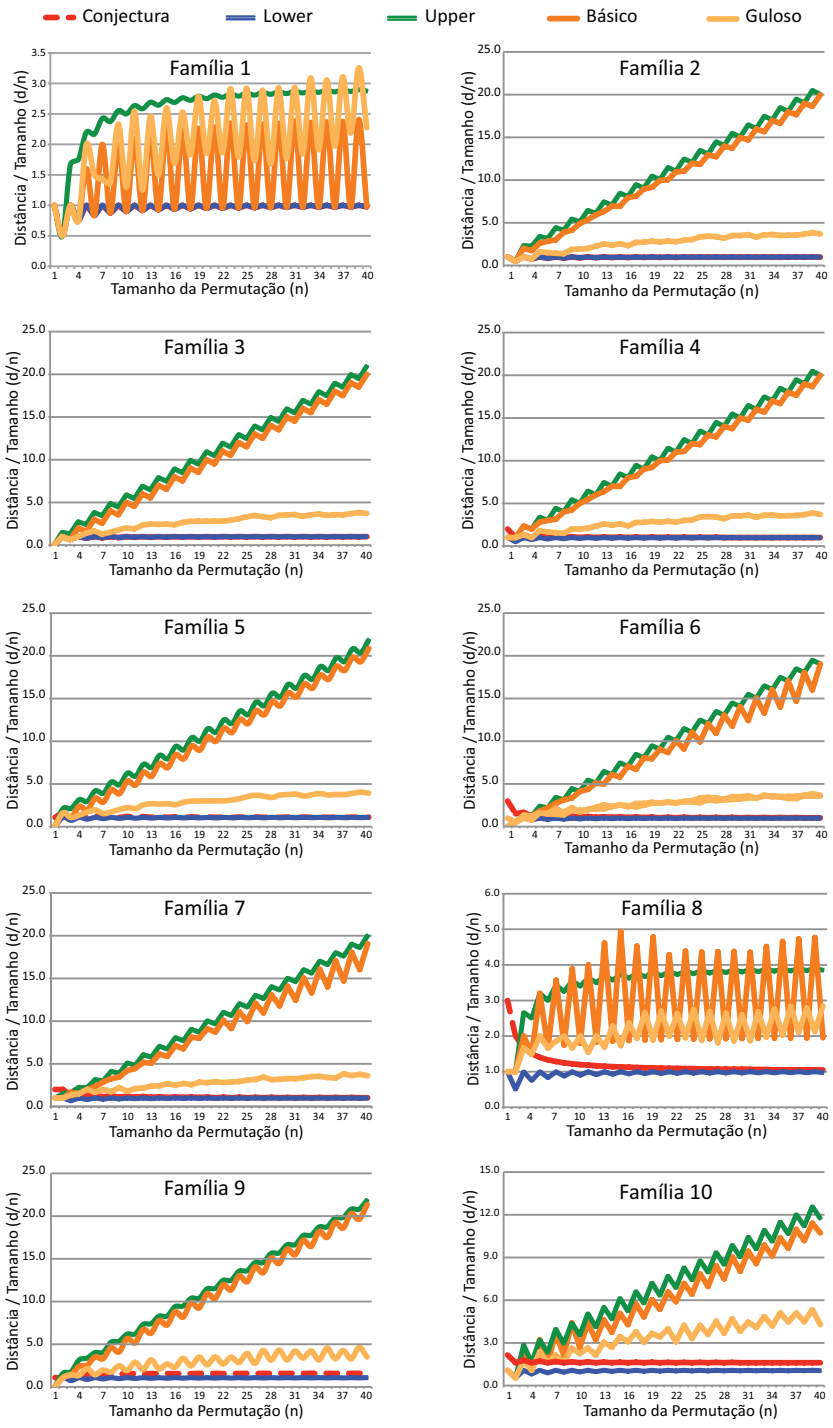


Figura 5.8: Gráficos das Famílias de Permutações. Os gráficos mostram os resultados obtidos usando os algoritmos específicos para cada família, os valores dos limitantes e os resultados obtidos pelos algoritmos básico e guloso.

- $d(C_3(n,k)) \leq 2\lfloor\frac{n-k}{2}\rfloor + 2\lfloor\frac{k}{2}\rfloor + 3$, *para* $1 \leq k < n$.
- $d(C_4(n)) \leq 2n$.
- $d(C_5(n)) \leq 3n$.

Todas as conjecturas relacionadas às classes são verdadeiras para permutações com 10 elementos ou menos. Foram implementados algoritmos para cada uma dessas classes e todos satisfazem o número conjecturado de reversões quase-simétricas.

As classes C_4 e C_5 são particularmente interessantes por fornecerem uma generalização para algumas das famílias de permutações. Por exemplo, $C_4(n) \supseteq \{F_1(n), F_4(n), F_6(n), F_8(n), F_{10}(n)\}$ e $C_5(n) \supseteq \{F_1(n), F_4(n), F_6(n), F_8(n), F_{10}(n), F_{12}(n), F_{13}(n)\}$. Além disso, $C_5(n) \supseteq C_4(n)$.

5.6 Publicações

Os resultados deste capítulo serão apresentados no artigo "*Sorting genomes using almost-symmetric inversions*" (Zanoni Dias, Ulisses Dias, João C. Setubal e Lenwood S. Heath) em março de 2012 na conferência SAC-BIO'2012 (*Symposium on Applied Computing - Conference Track on Bioinformatics and Computational Systems Biology*) a ser realizada em Riva del Garda (Trento), Itália [41].

Capítulo 6

Geração de Scaffolds usando Assinaturas de Reversão

Motivação: Com a diminuição nos custos de sequenciamento de DNA, tornou-se comum sequenciar apenas parcialmente genomas de organismos procariotos. Neste caso, o resultado obtido é composto de um conjunto de subsequências de DNA, sendo que a informação sobre a ordem e a orientação dessas subsequências não é conhecida. Neste contexto, construir um *scaffold* consiste em determinar a ordem dessas sequências e definir a orientação de cada uma delas.

Metodologia: Uma técnica popular de construção de *scaffolds* propõe mapear as subsequências em um genoma de referência. Entretanto, rearranjos podem existir entre o genoma alvo e o genoma de referência, o que resulta em resultados incorretos caso os rearranjos não sejam identificados. Em organismos procariotos, reversões são o evento mais comum de rearranjo. No presente capítulo, assinaturas para os eventos de reversão são definidas e um algoritmo para construção de *scaffolds* que faz uso dessas assinaturas é apresentado. O programa gerado com a implementação deste algoritmo recebeu o nome de SIS, acrônimo para *Scaffold from Inversion Signatures*.

Resultados: O algoritmo foi comparado a sete outros programas de construção de *scaffold* que usam genoma de referência. Os resultados evidenciam que SIS apresenta a melhor performance. Os testes foram realizados com a utilização de genomas de organismos procariotos.

6.1 Introdução

Sequenciamento de genomas é o processo laboratorial para determinar a sequência de DNA de um organismo. Idealmente, o resultado final deveria ser uma única sequência. Entretanto, o processo torna-se mais barato caso seja interrompido em um estágio conhecido como *draft*. Neste caso, o resultado é composto de um conjunto de subsequências de DNA chamadas de *contigs*, onde nem a ordem e nem a orientação relativa destes contigs são conhecidas.

Até 07 de dezembro de 2011, 2324 genomas microbiais incompletos foram submetidos ao NCBI, enquanto que apenas 1813 genomas microbiais completos estavam disponíveis. A Figura 6.1 mostra que a quantidade de genomas microbiais sequenciados até o estágio *draft* superou a quantidade de genomas microbiais completos durante o ano de 2011, o que gera uma necessidade por ferramentas que forneçam informações de ordem e orientações de contigs em um genoma incompleto. Em um caso ideal, a ferramenta deveria receber o conjunto de contigs e ordená-los, fornecendo uma orientação relativa para cada um deles, um processo conhecido como construção de *scaffold*.

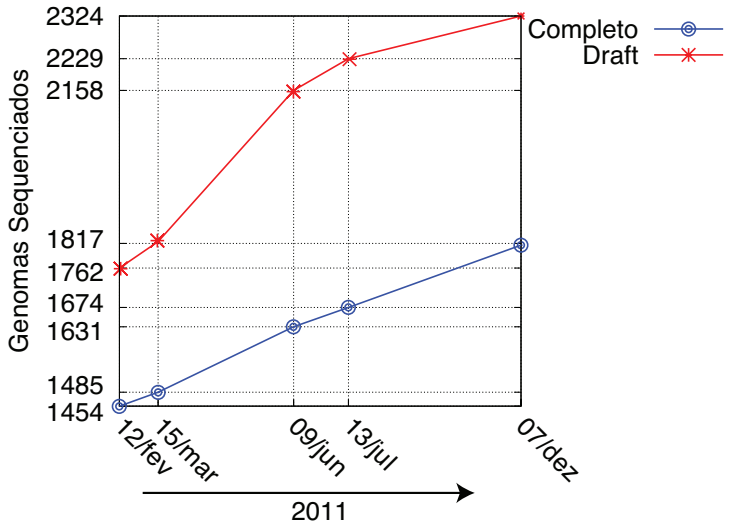


Figura 6.1: Número de genomas microbiais sequenciados até o estágio *draft* e genomas microbiais completamente sequenciados.

Várias técnicas existem para auxiliar na construção do *scaffold*. A maioria delas utiliza informações extraídas do processo de montagem do genoma como, por exemplo, *paired-end reads* [80]; outras utilizam dados de mapas físicos [98] ou óticos [81, 93]. A combinação de algumas dessas técnicas tem sido usada desde que o primeiro genoma bacterial foi sequenciado em 1995 [50], sendo que alguns pacotes de construção já incluem a produção

de *scaffolds* baseados em *paired-end reads* [10, 14, 17, 30, 64, 66, 71, 83].

Uma técnica que se tornou popular nos últimos anos consiste em utilizar um genoma completo como referência. Nesse caso, assume-se que o organismo A, que gerou o genoma incompleto, possui um irmão B, que já possui o genoma completamente sequenciado. O genoma de B pode ser usado como guia para construir o *scaffold* do genoma incompleto de A. A qualidade do *scaffold* construído depende da proximidade de A com B na escala evolutiva. Essa técnica é usada por programas como ABACAS [4], fillScaffold [79], Mauve Aligner [85], OSLay [84], Projector 2 [61], r2cat [65], PGA4genomics [102] e CONTIGuator [52].

Ao usar o genoma de B como referência para a construção do *scaffold* de A, um problema que surge decorre do fato de que rearranjos de genomas podem ter ocorrido no histórico evolutivo entre A, B e o ancestral comum. Em procariotos, os eventos de rearranjo mais comuns são as reversões e as reversões simétricas [29].

Neste capítulo, será apresentado um algoritmo para obtenção de *scaffolds* que leva em consideração a presença de reversões. A idéia fundamental do algoritmo consiste em procurar "assinaturas" de reversões nos contigs, uma estratégia que não é usada pelos outros programas de reconstrução, apesar de alguns deles possuírem estratégias próprias para lidar, pelo menos indiretamente, com eventos de rearranjo. Neste contexto, o Mauve Aligner [85], baseado no programa de alinhamento múltiplo Mauve [28] merece uma distinção especial por ter sido implementado para lidar com eventos de rearranjo.

O programa SIS é uma implementação do algoritmo apresentado neste capítulo. O SIS foi usado em uma análise comparativa com os outros programas de reconstrução de *scaffold* que utilizam um genoma B como referência. Nos testes, o SIS obteve a melhor performance por uma margem bastante significativa.

O capítulo é estruturado da seguinte forma. A Seção 6.2 apresenta o algoritmo para construção de *scaffolds*. A Seção 6.3 e a Seção 6.4 apresentam as análises, que são divididas em duas etapas, dependendo dos dados de entrada. A primeira etapa (Seção 6.3) analisa o algoritmo usando genomas simulados, enquanto que a segunda etapa (Seção 6.4) analisa o algoritmo usando genomas reais. Em ambas as etapas, foi feita uma comparação entre o algoritmo desenvolvido neste capítulo e outros programas de construção de *scaffold*.

6.2 Algoritmos

Genomas são representado por permutações com sinais, onde os números representam os blocos conservados entre dois genomas e os sinais indicam as orientações dos blocos. O genoma de referência é representado pela permutação identidade $\iota = (+1\ +2\ +3\ \ldots\ +n)$, enquanto um genoma arbitrário é representado pela permutação $\pi = (\pm\pi_1\ \pm\pi_2\ \ldots\ \pm\pi_n)$, onde $1 \leq \pi_i \leq n$ e $\pi_i \neq \pi_j$ se $i \neq j$.

Uma reversão $\rho(i, j)$ é um evento de rearranjo que inverte a ordem (e os sinais) de um trecho consecutivo de blocos de um genoma, ou seja, $\pi\rho = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_{i+1}, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$, tal que $1 \leq i < j \leq n$.

Um par de elementos consecutivos (π_i, π_{i+1}) é chamado de breakpoint se $\pi_i \neq \pi_{i+1} - 1$, e, caso contrário, é chamado de adjacência. Uma *strip* é uma subsequência maximal $[\pi_i, \pi_{i+1}, \ldots, \pi_j]$ tal que todo par (π_k, π_{k+1}) é uma adjacência, para $i \leq k < j$. Diz-se que uma reversão $\rho(r, s)$ age em um *strip* $[\pi_i, \pi_{i+1}, \ldots, \pi_j]$ se $i < r \leq s < j$.

Uma assinatura de reversão (IS, de *Inversion Signature*) é um breakpoint (π_i, π_{i+1}) tal que π_i e π_{i+1} possuem sinais distintos. Um par de assinaturas (π_i, π_{i+1}) e (π_j, π_{j+1}) formam um IS-Pair se $\pi_i = -\pi_j + 1$ e $\pi_{i+1} = -\pi_{j+1} + 1$, como mostra a Figura 6.2.

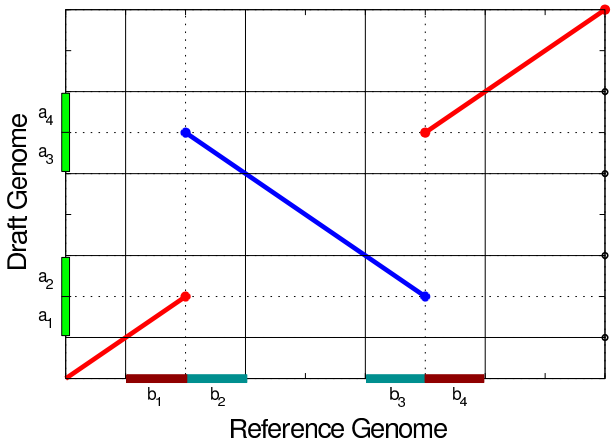


Figura 6.2: Duas regiões com assinaturas de reversão: a região a_1, a_2 no genoma incompleto ("draft") foi pareada com a região b_1, b_3 do genoma de referência. O fato de b_1 e b_3 não serem consecutivos no genoma de referência caracteriza o que chamamos de breakpoint e o fato de o alinhamento entre a_1 e b_1 possuir uma orientação diferente do alinhamento de a_2 e b_3 caracteriza o que chamamos de assinatura de reversão. A mesma situação explicada acima ocorre entre a assinatura de reversão correspondente às regiões $(a3, a4)$ de A e $(b2, b4)$ de B. Ambas as assinaturas se relacionam por terem sido formadas pela ação de uma única reversão e, neste caso, denominamos estas assinaturas de IS-Pairs.

Uma reversão $\rho(i, j)$ é dita **simétrica** se $n = i + j - 1$ (Figura 6.3). Uma reversão $\rho(i_1, j_1)$ é dita **aninhada** com respeito a uma reversão $\rho(i_2, j_2)$, se $i_2 < i_1 \leq j_1 < j_2$ (Figura 6.4). Uma reversão $\rho(i, j)$ é dita **segura** com respeito a π, se $\rho(i, j)$ age em uma *strip* $[\pi_r, \pi_{r+1}, \ldots, \pi_s]$ de π (Figura 6.5). Uma reversão $\rho(i, j)$ sem qualquer restrição nos valores de i e j será chamada de reversão **genérica** (Figura 6.6).

Sejam $P_1(n)$, $P_2(n)$, $P_3(n)$ e $P_4(n)$ os conjuntos de todas as permutações que podem ser obtidas a partir da identidade (ι) aplicando-se, respectivamente, uma série de reversões

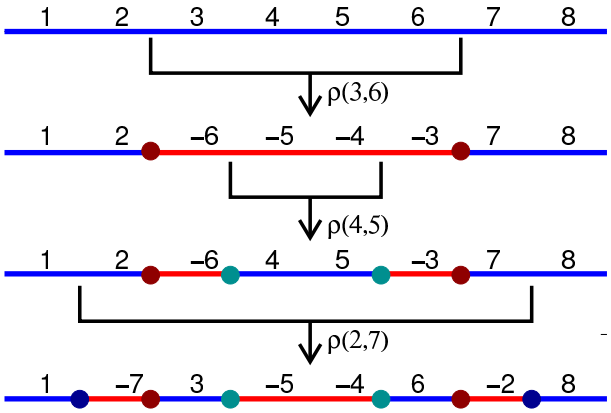


Figura 6.3: Três reversões simétricas (ρ(3,6), ρ(4,5) e ρ(2,7)) aplicadas na permutação identidade. Note que o resultado final não é alterado caso as reversões simétricas fossem aplicadas em outra ordem. Os círculos colorido representam IS-Pairs criados pelas reversões simétricas.

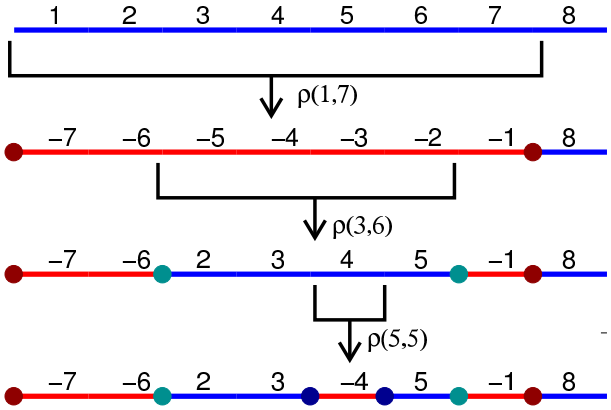


Figura 6.4: Três reversões aninhadas (ρ(1,7), ρ(3,6) e ρ(5,5)) aplicadas na permutação identidade. Note que o resultado final é alterado caso essas reversões sejam aplicadas em outra ordem. Qualquer sequência de reversões simétricas pode ser ordenada de forma aninhada, com o mesmo resultado original. Os círculos colorido representam IS-Pairs criados pelas reversões aninhadas.

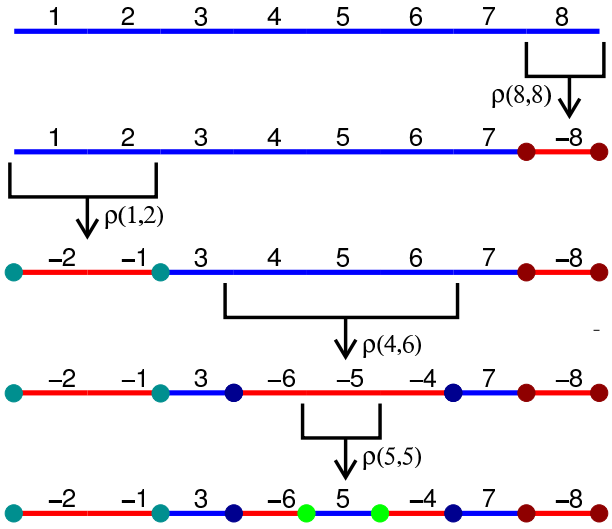


Figura 6.5: Quatro reversões seguras ($\rho(8,8)$, $\rho(1,2)$, $\rho(4,6)$ e $\rho(5,5)$) aplicadas na permutação identidade. Toda sequência de reversões aninhadas é uma sequência de reversões seguras (o inverso não é verdade). Os círculos colorido representam IS-Pairs criados pelas reversões seguras.

simétricas, uma série de reversões aninhadas, uma série de reversões seguras e uma série de reversões quaisquer. Claramente, $P_1(n) \subset P_2(n) \subset P_3(n) \subset P_4(n)$.

Uma coleção de m contigs de um genoma π é representada por uma coleção de subsequências $\{C_1, C_2, \ldots, C_m\}$, tal que, para $1 \leq k \leq m$, $C_k = [\pi_{i_k}, \ldots, \pi_{j_k}]$ ou $C_k = [-\pi_{j_k}, \ldots, -\pi_{i_k}]$ e os intervalos mapeados em π por dois contigs quaisquer C_{k_1} e C_{k_2} não podem se sobrepor, ou seja, $[i_{k_1}..j_{k_1}] \cap [i_{k_2}..j_{k_2}] = \emptyset$, para $1 \leq k_1 < k_2 \leq m$.

6.2.1 Algoritmo Básico

Foram desenvolvidos dois algoritmos, o primeiro deles, chamado de algoritmo básico, gera scaffolds corretos na presença de reversões simétricas, aninhadas e seguras. O segundo algoritmo é uma generalização do primeiro e é capaz de produzir scaffolds na presença de reversões genéricas. O segundo algoritmo não garante que o scaffold produzido esteja correto e uma análise de sua performance é mostrada na Seção 6.3, a análise também compara o nosso algoritmo com outros previamente publicados.

Quando algumas premissas são satisfeitas, o algoritmo básico é capaz de gerar um scaffold perfeito. Entretanto, o scaffold gerado é muito bom mesmo quando algumas dessas premissas não são satisfeitas, como será mostrado pelas análises da Seção 6.3 e da Seção 6.4.

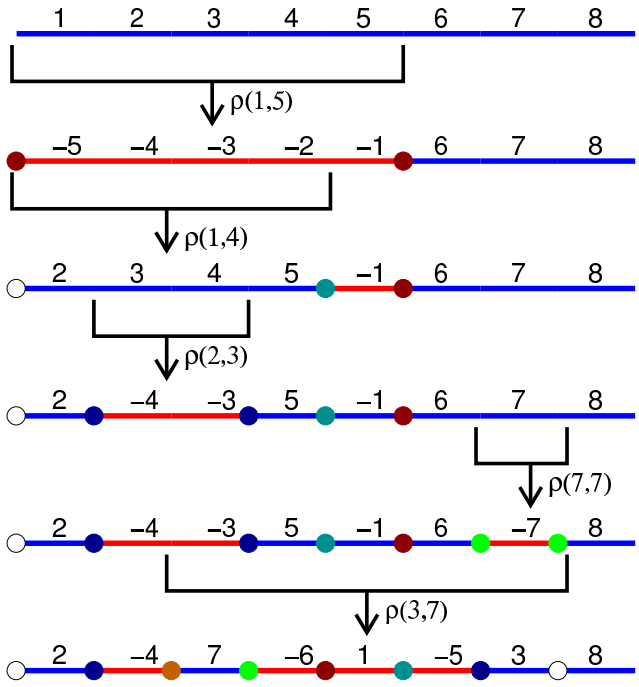


Figura 6.6: Cinco reversões genéricas ($\rho(1,5)$, $\rho(1,4)$, $\rho(2,3)$, $\rho(7,7)$ e $\rho(3,7)$) aplicadas na permutação identidade. Os círculos brancos representam os ISs destruídos por reversões não seguras.

1. Não há blocos conservados repetidos (duplicações) em nenhum dos genomas;
2. Todos os contigs estão corretamente montados;
3. Ambos os genomas são compostos por um único cromossomo;
4. Em termo dos blocos conservados, a diferença entre os dois genomas pode ser representada como uma série de reversões seguras;
5. O bloco conservado identificado com o número 01 está na primeira posição (com sinal positivo) ou na última posição (com sinal negativo) de algum contig.

A premissa (1) significa que não há blocos conservados contendo uma sequência repetida. Essa premissa é comum em estudos de rearranjo de genomas, sendo que a premissa (2) depende dela, já que a qualidade da montagem do genoma é altamente dependente da ausência de repetições.

A premissa (3) permite uma simplicidade de exposição do algoritmo, pois, em problemas reais, quando um organismo possui mais de um cromossomo, é possível gerar o *scaffold* de cada cromossomo independentemente.

A premissa (4) cria uma restrição para os genomas de entrada, de modo que seja possível para o algoritmo básico fornecer uma resposta, vale ressaltar que essa restrição possui uma motivação biológica forte, pois o alinhamento par-a-par de genomas completos mostra que a maioria das reversões que ocorrem na natureza apresentam uma certa simetria em relação a origem de replicação [45].

A premissa (5) simplifica a exposição do algoritmo, evitando que ele tenha que explicitamente tratar os genomas como circulares. Caso o bloco 1 não atenda a restrição do item (5), um remapeamento dos blocos conservados pode ser feito em tempo $O(n)$, onde n é o número de blocos conservados, de forma a garantir tal propriedade.

O algoritmo recebe como entrada uma lista de contigs representados pelas sequências dos blocos conservados em relação ao genoma de referência. A numeração dos blocos conservados em cada contig pode ser facilmente obtida com programas que fazem o alinhamento de genomas completos, como por exemplo, cross_match (www.phrap.org) ou MUMmer [68].

Sejam A e B um par de genomas com 40 blocos conservados. O genoma A está incompleto e possui 9 contigs a serem ordenados, enquanto que o genoma B já fora completamente sequenciado. Cada bloco conservado do genoma B recebeu uma numeração no intervalo $[1, 40]$, sendo que o bloco $i + 1$ é o sucessor do bloco i no genoma B. Dado que o genoma B é circular, o bloco 1 é o sucessor do bloco 40. Ao mapear os blocos conservados nos contigs do genoma A, obtém-se a seguinte listagem:

Contig 1: $[-23, -22, +19, +20]$
Contig 2: $[-15, -14, +12, +13, -11]$
Contig 3: $[+34, +35]$
Contig 4: $[+36, +37, +38, +39, +40]$
Contig 5: $[-03, -02, -01]$
Contig 6: $[+05, +06, +07, +08]$
Contig 7: $[+31, +32, +33, -30, -29]$
Contig 8: $[-28, -27, +26, -25, -24, +04]$
Contig 9: $[+21, -18, -17, -16, +09, +10]$

A ordem correta dos contigs no genoma A é dada a seguir:

$$\begin{aligned}\pi = \; & [+01, +02, +03, -23, -22, +19, +20, +21,\\ & -18, -17, -16, +09, +10, +11, -13, -12,\\ & +14, +15, -08, -07, -06, -05, -04, +24,\\ & +25, -26, +27, +28, -35, -34, +31, +32,\\ & +33, -30, -29, +36, +37, +38, +39, +40]\end{aligned}$$

Observa-se que 7 reversões seguras agiram na diferenciação dos genomas. Neste casos os IS-Pairs são:

IS-Pair 1: $[+03, -23] \times [-04, +24]$
IS-Pair 2: $[-22, +19] \times [+21, -18]$
IS-Pair 3: $[-16, +09] \times [+15, -08]$
IS-Pair 4: $[+11, -13] \times [-12, +14]$
IS-Pair 5: $[+25, -26] \times [-26, +27]$
IS-Pair 6: $[+28, -35] \times [-29, +36]$
IS-Pair 7: $[-34, +31] \times [+33, -30]$

Entretanto, apenas os seguintes IS estão presentes em algum contig:

IS-Pair 1: $\times [-04, +24]$
IS-Pair 2: $[-22, +19] \times [+21, -18]$
IS-Pair 3: $[-16, +09] \times$
IS-Pair 4: $[+11, -13] \times [-12, +14]$
IS-Pair 5: $[+25, -26] \times [-26, +27]$
IS-Pair 6: $\times$
IS-Pair 7: $\times [+33, -30]$

O algoritmo básico construirá o scaffold incrementalmente, escolhendo qual o contig deve ser posicionado ao lado do último previamente posicionado no scaffold. Os blocos conservados são procurados na ordem imposta pelos contigs e, toda vez que um bloco é encontrado em uma ordem diferente da esperada, as assinaturas de reversão (ISs) são usadas para inferir o contig correto a ser posicionado.

O algoritmo inicia posicionando o contig que contém o bloco conservado +01. No exemplo da Figura 6.7, este bloco é encontrado com o sinal negativo na última posição do Contig 5. Assim, o Contig 5 é inserido invertido na primeira posição do scaffold.

A seguir, o algoritmo procura o bloco conservado consecutivo ao último bloco do primeiro contig do scaffold, ou seja, o bloco +04. Este bloco é encontrado no Contig 8, porém em uma posição incorreta (deveria ser o primeiro bloco, já que seu o sinal é positivo). O algoritmo então supõe que a alteração da ordem foi decorrente de uma reversão e usa a assinatura de reversão $(-24, +04)$ para procurar o contig que contém o bloco −23, que deve estar na outra ponta da reversão que criou aquela assinatura. O algoritmo então encontra o bloco −23 na primeira posição do Contig 1, posicionando este contig no scaffold.

A seguir, o bloco +21 é localizado no começo do Contig 9. Após posicionar o Contig 9 na terceira posição do scaffold, o algoritmo busca o bloco +11 encontrado com sinal negativo na última posição do Contig 2.

Após inserir o Contig 2 invertido no *scaffold*, o algoritmo procura o bloco +16 encontrado invertido no meio do Contig 9. O algoritmo usa a assinatura de reversão $(-16, +09)$ para procurar o bloco −08 que substitui o bloco +16 na sequências dos blocos conservados do genoma que está sendo reconstruído. O bloco −08 é encontrado invertido na última

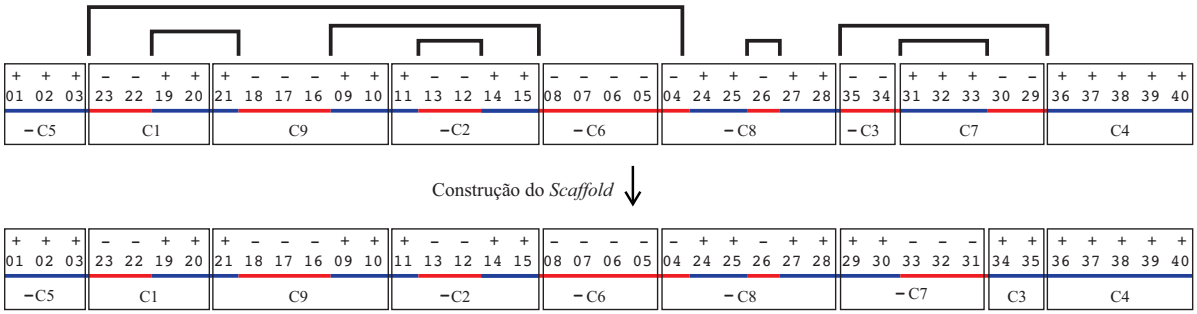


Figura 6.7: Exemplo de um genoma com 40 blocos conservados em relação ao um genoma de referência. Os blocos conservados estão divididos em 9 contigs. A diferenciação dos genomas pode ser explicada por 7 reversões seguras. Das 14 assinaturas de reversões, apenas 9 podem ser identificadas nos contigs. Ambas as assinaturas do IS-Pair gerado pela reversão que agiu nos contigs 3 e 7 foram completamente perdidas por falta de cobertura dos contigs. Na primeira linha de contigs exibimos a reconstrução perfeita, enquanto na segunda linha, a reconstrução realizada pelo nosso algoritmo.

posição do Contig 6.

O algoritmo segue até reconstruir o scaffold (-5 -1 $+9$ -2 -6 -8 -7 $+3$ $+4$). Observa-se que o scaffold reconstruído não foi perfeito, já que a ordem do genoma seria (-5 -1 $+9$ -2 -6 -8 -3 $+7$ $+4$). O algoritmo não conseguiu reconstruir a ordem correta pois não há informação nos contigs que permitam deduzir que houve uma reversão que inverteu o bloco de contigs $[-7, +3]$. Note que não há vestígios de nenhuma das pontas do IS-Pair 6 nos contigs. Neste caso o algoritmo acertou 7 das 9 adjacências do contig.

O Algoritmo 1 mostra como implementar o método previamente descrito. Duas estruturas de dados auxiliares são usadas: a estrutura `contigs` armazena os contigs e a estrutura `inContig` armazena, para cada bloco conservado, em que contig, posição e sinal ele se encontra na estrutura `contigs`.

Se `Contig[`c`][`p`] =` x, então `inContig[`$|x|$`].contig =` c, `inContig[`$|x|$`].pos =` p e `inContig[`$|x|$`].sign =` s, de tal forma que $s \times |x| = x$. Por exemplo, `contig[7]` $= [+31, +32, +33, -30, -29]$, `contig[7][4]` $= -30$, `inContig[`$|-30|$`].contig` $= 7$, `inContig[`$|-30|$`].pos` $= 4$ e `inContig[`-30`].sign` $= -1$.

O loop principal das Linhas 2-16 posiciona, um a um, os contigs no scaffold. A Linha 3 recupera o registro que armazena a informação de contig, posição e sinal do bloco que está sendo posicionado. As Linhas 3-9 verificam se o bloco está na posição correta e em caso negativo usa (implicitamente) as informações das assinaturas de reversão para localizar o bloco correto. Na Linha 10, as informações do novo bloco são recuperadas. A Linha 11 posiciona o contig correto no scaffold. Por fim, as Linhas 12-16 determinam o novo bloco

Algoritmo 13: Algoritmo Básico

```
   Data: contigs, inContig, m
   Result: scaffold
1  next ← 1
2  for i ← 1 to m do
       // Procura elemento nos contigs
3      e ← inContig[|next|]
4      if sign(next) = e.sign then
5          if e.pos ≠ 1 then
6              next ← contigs[e.contig][e.pos − 1] + 1
7      else
8          if e.pos ≠ length(contigs[e.contig]) then
9              next ← −contigs[e.contig][e.pos + 1] + 1
10     e ← inContig[|next|]
       // Posicionar contig no scaffold
11     scaffold[i] ← sign(next) × e.sign × e.contig
       // Determinar o próximo elemento na busca
12     if sign(next) = e.sign then
13         lastIndex ← length(contigs[e.contig])
14         next ← contigs[e.contig][lastIndex] + 1
15     else
16         next ← −contigs[e.contig][1] + 1
17 return scaffold
```

a ser procurado.

Seja n o número de blocos conservados entre os genomas e m o número de contigs, a estrutura de dados `inContig` precisa de tempo $O(n)$ para ser construída. A complexidade para posicionar os blocos é $O(m)$. Dado que $n > m$, a complexidade geral do algoritmo é $O(n)$.

O algoritmo é capaz de reconstruir o scaffold correto caso pelo menos uma assinatura de reversão de cada IS-Pair esteja presente nos contigs. Caso as duas ISs de um IS-Pair estejam ausentes, então o algoritmo irá posicionar uma sequência de contigs incorretamente invertida (preservando as demais relações de adjacência).

6.2.2 Algoritmo para Reversões Genéricas

O algoritmo da seção anterior é capaz de construir *scaffolds* com alta fidelidade, mesmo na ausência de algumas assinaturas de reversão, porém considerando apenas reversões seguras. Na prática, não há garantia de que as diferenças entre dois genomas possam ser explicada apenas por reversões seguras. Assim, modificou-se o algoritmo básico para que ele seja capaz de lidar com reversões genéricas.

O algoritmo permanece usando as assinaturas de reversão para posicionar os contigs, caso os blocos apareçam em uma ordem diferente da esperada. Eventualmente, uma série de assinaturas de reversões pode ser usada para localizar o contig correto. No entanto, algumas assinaturas podem ter sido destruídas por reversões não seguras.

Sendo assim, devido a ausência de informações confiáveis que indiquem qual reversão ocorreu, o algoritmo pode não ser capaz de encontrar um contig válido, e, neste caso, o algoritmo modificado faz com que ele escolha um contig ainda não posicionado no scaffold.

A alteração faz com que o algoritmo modificado sempre retorne um scaffold válido (não necessariamente perfeito). O algoritmo modificado mantém o comportamento original caso todas as reversões sejam seguras.

O Algoritmo 14 mostra o pseudocódigo do novo algoritmo para lidar com reversões genéricas. O novo algoritmo usa dois novos vetores auxiliares: o vetor `used` indica quais contigs já foram posicionados no scaffold e o vetor `searched` indica todos os blocos conservados que já foram procurados até o momento. A variável `nextC` indica o contig de menor identificador ainda não posicionado no scaffold.

As Linhas 1-6 inicializam as variáveis auxiliares. O loop principal das Linhas 7-39 é executado até que todos os contigs estejam posicionados no scaffold. A Linha 8 inicializa a variável `ok` utilizada para indicar se o algoritmo achou um contig candidato a ser inserido no scaffold. A Linha 9 garante que o algoritmo não está procurando um bloco conservado inválido. O bloco das Linhas 10-20 é similar ao bloco das Linhas 3-9 do Algoritmo 1, que verifica se o bloco está na posição correta e, em caso negativo, usa (implicitamente) as informações das assinaturas de reversão para localizar o bloco correto. As Linhas 21-24 testam se o algoritmo está procurando um bloco conservado já pesquisado.

Se o algoritmo decidir inserir um novo contig no scaffold (teste da Linha 25), então executa o bloco das Linhas 26-39. Se o algoritmo não foi capaz de encontrar um contig válido, o bloco das Linhas 26-28 faz que o algoritmo considere o primeiro contig ainda não inserido no scaffold. As Linhas 29-30 posicionam o contig no scaffold. As Linhas 31-34 atualizam o vetor `used` e as variáveis `nextC` e `i`. O bloco das Linhas 35-39 é idêntico ao bloco das Linhas 12-16 do Algoritmo 1, e servem para determinar o próximo bloco conservado a ser procurado.

A complexidade da inicialização das variáveis (Linhas 1-6) é $O(n+m)$. O loop principal é executado $O(n + m)$ vezes, já que, a cada iteração, ou uma nova posição do vetor `searched` é marcada ou um contig é posicionado no scaffold.

A única operação dentro do loop principal que não tem custo constante é o loop das Linhas 33-34, mas como a variável `nextC` pode ser incrementada no máximo m vezes ao longo de toda a execução do algoritmo, então seu custo amortizado é $O(1)$. Sendo assim, o algoritmo modificado tem a mesma complexidade do algoritmo original, $O(n + m)$, incluindo o custo de construção da estrutura `inContig`.

Algoritmo 14: Algoritmo para Inversões Genéricas

```
   Data: contigs, inContig, m, n
   Result: scaffold
 1 for i ← 1 to m do used[i] ← false
 2 for i ← −n to +n do searched[i] ← false
 3 searched[n+1] ← true
 4 nextC ← 1
 5 next ← 1
 6 i ← 0
   // Enquanto houver contigs não posicionados
 7 while i < m do
 8     ok ← false
 9     if next ≠ n + 1 then
           // Procurar pelo elemento nos contigs
10         e ← inContig[—next—]
11         if sign(next) = e.sign then
12             if e.pos ≠ 1 then
13                 next ← contigs[e.contig][e.pos−1] + 1
14             else
15                 ok ← true
16         else
17             if e.pos ≠ len(contigs[e.contig]) then
18                 next ← −contigs[e.contig][e.pos+1] + 1
19             else
20                 ok ← true
21     if not(searched[next]) then
22         searched[next] ← true
23     else
24         ok ← true
           // Posicionar o contig no Scaffold
25     if ok then
           // Se o contig já está posicionado no scaffold
26         if next = n + 1 or used[e.contig] then
27             next ← contigs[nextC][1]
28             e ← inContig[—next—]
29         contigSign ← sign(next) × e.sign
30         scaffold[i] ← contigSign × e.contig
31         i ← i + 1
32         used[e.contig] ← true
33         while nextC ≤ m and used[nextC] do
34             nextC ← nextC + 1
           // Determinar próximo elemento na busca
35         if sign(next) = e.sign then
36             lastIndex ← len(contigs[e.contig])
37             next ← contigs[e.contig][lastIndex] + 1
38         else
39             next ← −contigs[e.contig][1] + 1
40 return scaffold
```

6.3 Análise dos Algoritmos com Genomas Incompletos Simulados

Dois testes foram realizados. No primeiro teste, apresentado na Seção 6.3.1, dados simulados foram gerados com o intuito de avaliar o impacto da falta de ISs na performance do algoritmo.

No segundo teste apresentado na Seção 6.3.2, genomas reais foram utilizados, mas os contigs foram simulados. A simulação ocorreu para avaliar o SIS, programa que implementa o algoritmo para reversões genéricas apresentado na Seção 6.2.2, e os principais programas de reconstrução da literatura em um cenário simplificado.

A qualidade dos scaffold será medida em termos do número de adjacências corretas. Uma adjacência é dita correta se dois contigs aparecem consecutivos tanto no scaffold produzido quanto na ordem correta dos contigs. Um scaffold é dito perfeito se todas suas adjacências são corretas. No caso de genomas circulares, todo scaffold perfeito de m contigs possui exatamente m adjacências corretas.

6.3.1 Testes Simulados

Foram geradas 5000 permutações aleatórias de tamanho $n = 1000$ pertencentes aos conjuntos $P_3(r)$ (reversões seguras) e $P_4(r)$ (reversões genéricas), para $1 \leq r \leq 200$, totalizando 2 milhões de casos de testes (1 milhão para cada tipo de reversão). Os conjuntos $P_3(r)$ e $P_4(r)$ foram previamente definidos na Seção 6.2.

Para cada genoma, foram utilizados 99 pontos aleatórios distintos que serviram para gerar 100 contigs. Nenhuma restrição extra foi imposta ao particionamento dos contigs, de forma que, eventualmente, alguns breakpoints (ISs, no caso de reversões seguras) não podiam ser identificados nos contigs, apesar de presentes no genoma original. Os contigs foram aleatoriamente embaralhados e revertidos, antes de serem fornecidos para o Algoritmo 14, capaz de lidar com os casos de reversões seguras e genéricas. A estratégia de construção deste teste foi similar a ilustrada pela Figura 6.7.

Após a execução do algoritmo, calculou-se para cada instância de teste o número de adjacências corretas e o número de breakpoints (ISs, no caso de reversões seguras) que, apesar de presentes no genoma, não puderam ser identificados nos contigs.

A Figura 6.8 mostra o gráfico com o número médio de adjacências corretas em relação ao número de reversões de cada conjunto. Por exemplo, SIS acerta, em média, mais de 96% das adjacências para o caso de reversões seguras. Para o caso de reversões genéricas, SIS acerta, em média, quase 97% das adjacências, quando 100 ou menos reversões são aplicadas. Em média, o algoritmo acerta mais de 92% das adjacências quando os genomas diferem por 200 ou menos reversões genéricas.

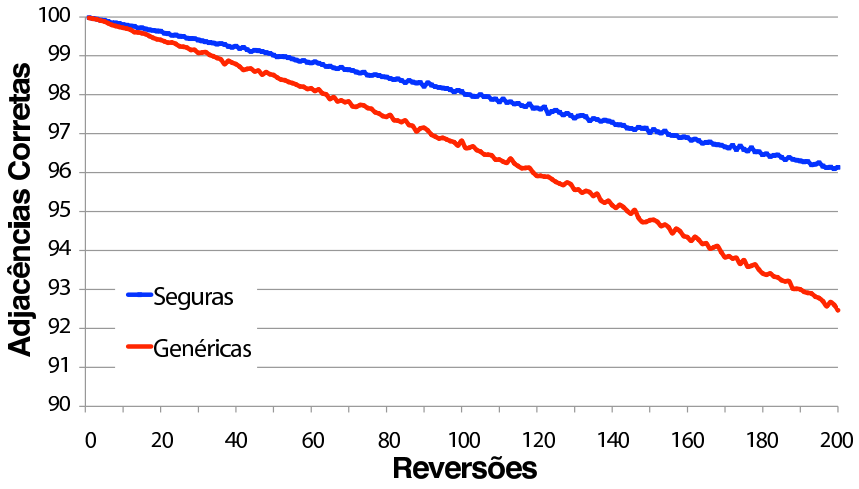


Figura 6.8: Gráfico com o número médio de adjacências corretas em relação ao número de reversões de cada conjunto.

A Figura 6.9 mostra o gráfico com o número médio de breakpoints perdidos em relação ao número de reversões de cada conjunto. Por exemplo, no caso de reversões seguras, o número médio de breakpoints perdidos (IS, neste caso) não chega a 40. No caso de reversões genéricas, o número médio de breakpoints perdidos, não chega a 31. O crescimento dos breakpoint perdidos no caso de reversões seguras é linear, enquanto no caso de reversões genéricas o gráfico indica um crescimento mais lento, já que um breakpoint pode ser afetado por várias reversões.

A Figura 6.10 mostra o gráfico com a porcentagem de casos de testes com número de adjacências corretas acima de um dado valor. Por exemplo, no caso de reversões seguras, em mais de 70% dos casos de testes o algoritmo acertou mais de 98% das adjacências. Já para reversões genéricas, em mais de 70% dos casos de testes o algoritmo acertou mais de 95% das adjacências.

A Figura 6.11 mostra a porcentagem de scaffolds perfeitos (100% de adjacências corretas) em relação ao número de reversões de cada conjunto. Por exemplo, no caso de reversões seguras, o número de scaffolds perfeitos com 50 reversões é superior a 63%. No caso de reversões genéricas, é acima de 60%.

Esses resultados mostram que o SIS realiza um bom trabalho mesmo nos casos em que um número substancial de IS estão faltando. Entretanto, um teste real deve ser feito em comparação com outros programas de reconstrução e esse é o propósito das próximas seções.

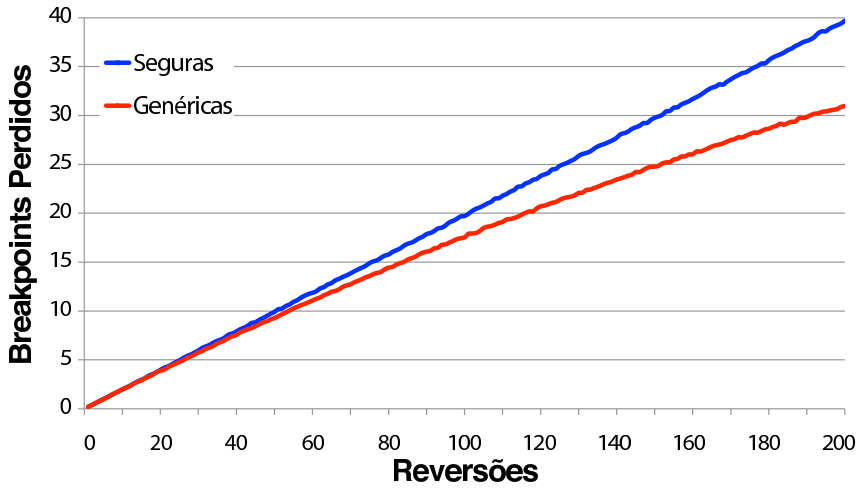


Figura 6.9: Gráfico com o número médio de breakpoints perdidos em relação ao número de reversões de cada conjunto.

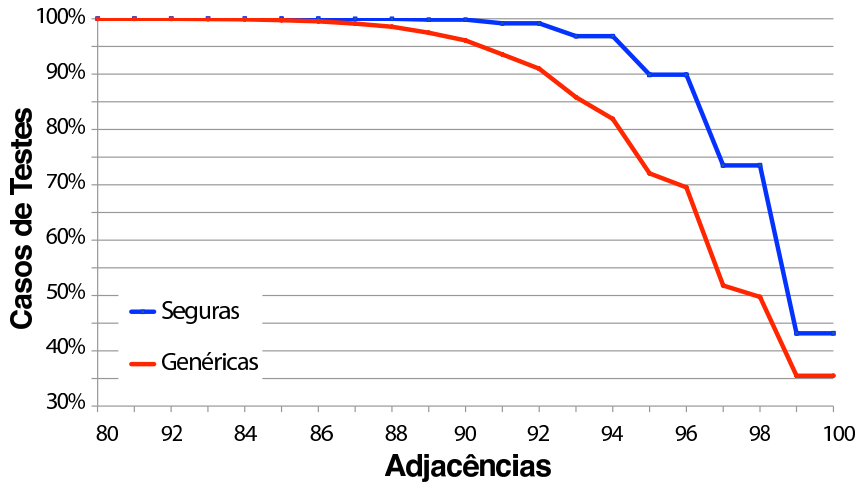


Figura 6.10: Porcentagem de casos de testes com número de adjacências corretas acima de um dado valor no eixo X.

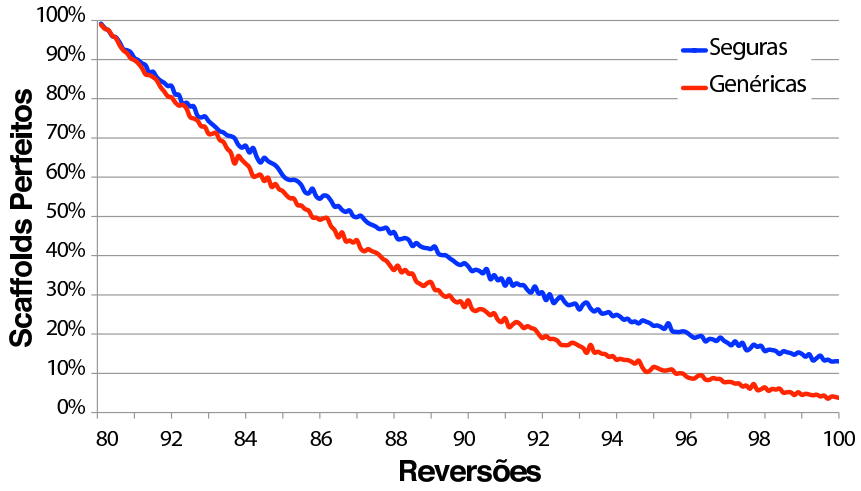


Figura 6.11: Porcentagem de scaffold perfeitos em relação ao número de reversões de cada conjunto.

6.3.2 Testes Simulados com Genomas Reais

Para este grupo de testes, utilizou-se os genomas completos de 21 organismos do gênero *Mycobacterium*, 18 organismos da família *Pseudomonadaceae*, 20 organismos do gênero *Shewanella* e 9 organismos do gênero *Xanthomonas*. Os organismos possuem apenas um cromossomo principal, mas alguns deles possuem pequenos plasmídeos que não foram considerados na análise. Foram utilizados todos os genomas disponíveis no NCBI para esses grupos de organismos em novembro de 2010.

Os grupos de contigs foram simulados dividindo os genomas completos em 100 subsequências de aproximadamente o mesmo tamanho e, após isso, foram removidos 2.5% do início e 2.5% do final de cada uma dessas subsequências, com o intuito de simular o fato de que genomas incompletos normalmente não cobrem todo o genoma do organismo. Após isso, as 100 subsequências foram embaralhadas e invertidas (com 50% de probabilidade).

Seja $R = \{r_1, r_2, \ldots, r_n\}$ o conjunto dos genomas originais e $Q = \{q_1, q_2, \ldots, q_n\}$ o conjunto dos genomas incompletos obtidos pelo processo acima, onde q_i é o genoma incompleto obtido a partir de r_i, foi realizada uma série de $n(n-1)$ testes com cada um dos algoritmos de construção de scaffold, onde, em cada teste, o elemento r_i foi fornecido como referência e o elemento q_j foi fornecido como alvo da construção do scaffold, sendo $i \neq j$.

As figuras 6.12, 6.13, 6.14 e 6.15 mostram os resultados considerando todos os pares de instâncias de testes para cada programa. É possível observar que o SIS é avaliado duas vezes. Isso ocorre porque o SIS pode usar qualquer programa de identificação de blocos conservados. Nos testes, foram usados dois programas: `nucmer` e `promer`, ambos estão no

pacote MUMmer [68]. O primeiro identifica blocos usando a sequência de nucleotídeos, enquanto que o segundo realiza a comparação usando a tradução da sequência de DNA em uma sequência de aminoácidos.

No decorrer deste capítulo, a notação "SIS (nucmer)" será usada para identificar a utilização do SIS com o programa `nucmer` e a notação "SIS (promer)" será usada para identificar a utilização de SIS com o programa `promer`.

O SIS (promer) apresentou os melhores resultados em todos os organismos, como pode ser observado nas figuras. Na família *Pseudomonadaceae* (Figura 6.13), o pior resultado individual do SIS (promer) é melhor do que 81% dos resultados dos outros programas.

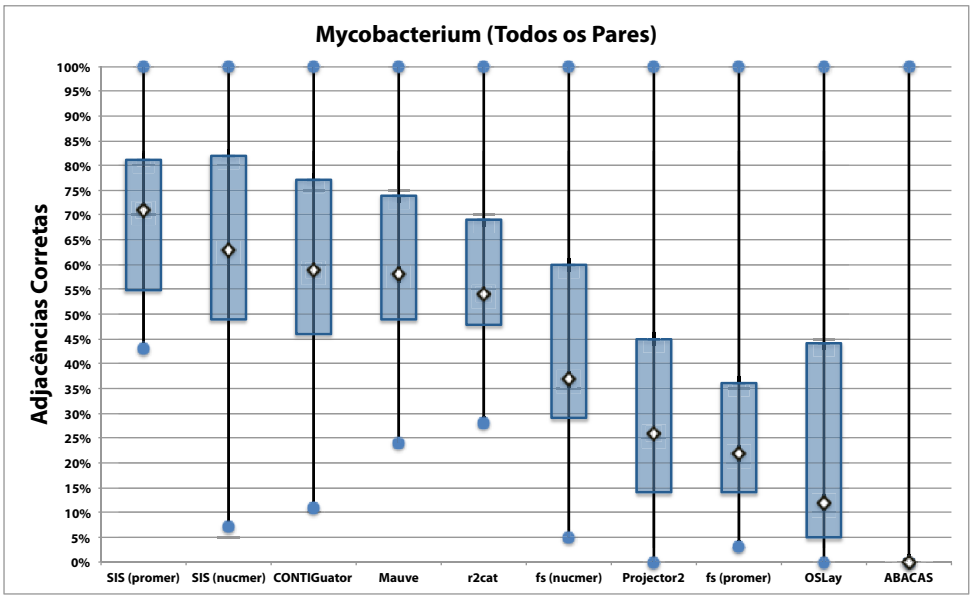


Figura 6.12: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Mycobacterium*. Nesse gráfico, foram considerados todos os 420 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds.

Na prática, a escolha de um genoma de referência será guiada pela distância filogenética, pois o genoma completo mais próximo de um genoma incompleto é, intuitivamente, aquele que fornecerá os melhores resultados. Assim, computou-se os pares de genomas mais próximos usando a distância NUCMi [37, 39] (a distância NUCMi será apresentada no Capítulo 7) e resultados específicos para esses pares foram tabulados.

As figuras 6.16, 6.17, 6.18 e 6.19 mostram que todos os programas obtiveram uma considerável melhora de performance nesses pares, mas o SIS continua sendo o melhor programa em todos os casos.

A diferença entre os resultados apresentados para todos os pares e os resultados para

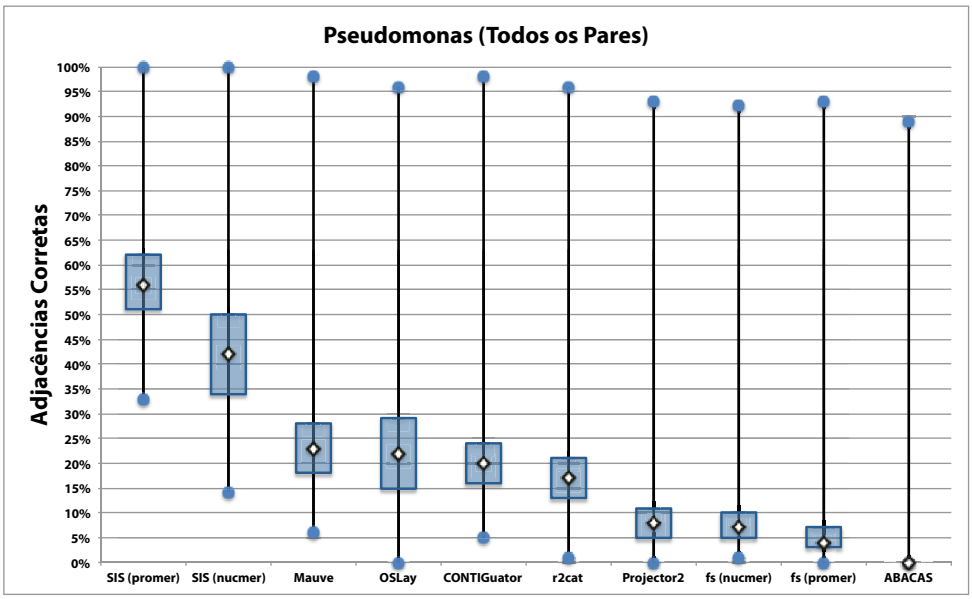


Figura 6.13: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Pseudomonadaceae*. Nesse gráfico, foram considerados todos os 306 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds.

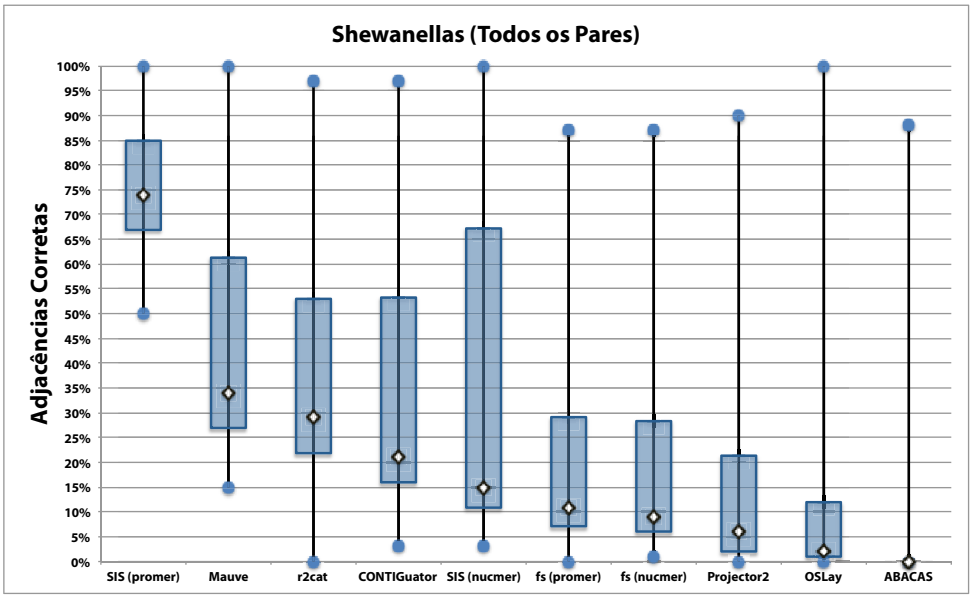


Figura 6.14: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Shewanella*. Nesse gráfico, foram considerados todos os 380 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds.

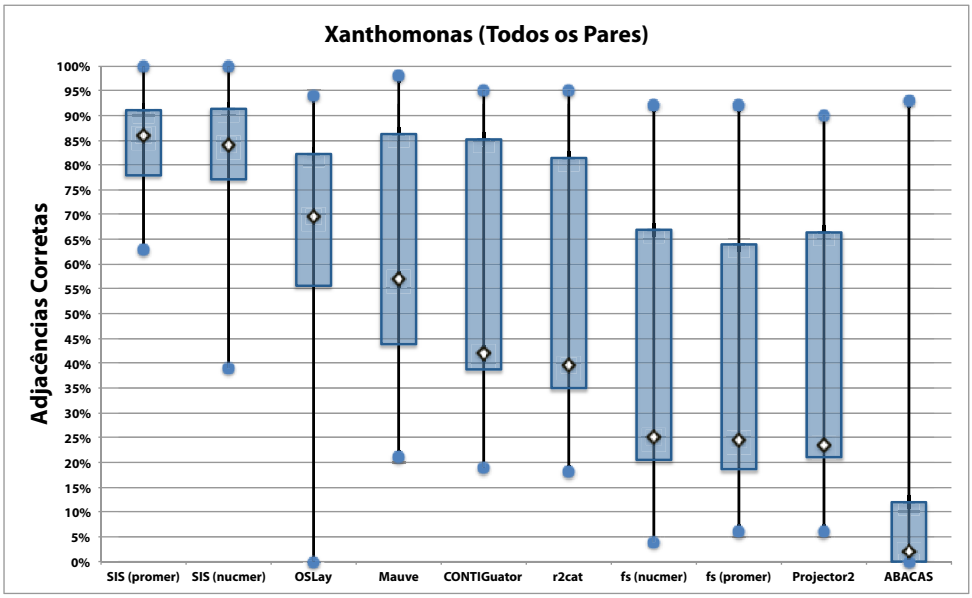


Figura 6.15: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Xanthomonas*. Nesse gráfico, foram considerados todos os 72 pares de instâncias de teste. O termo **fs** foi usado para referenciar o programa fillScaffolds.

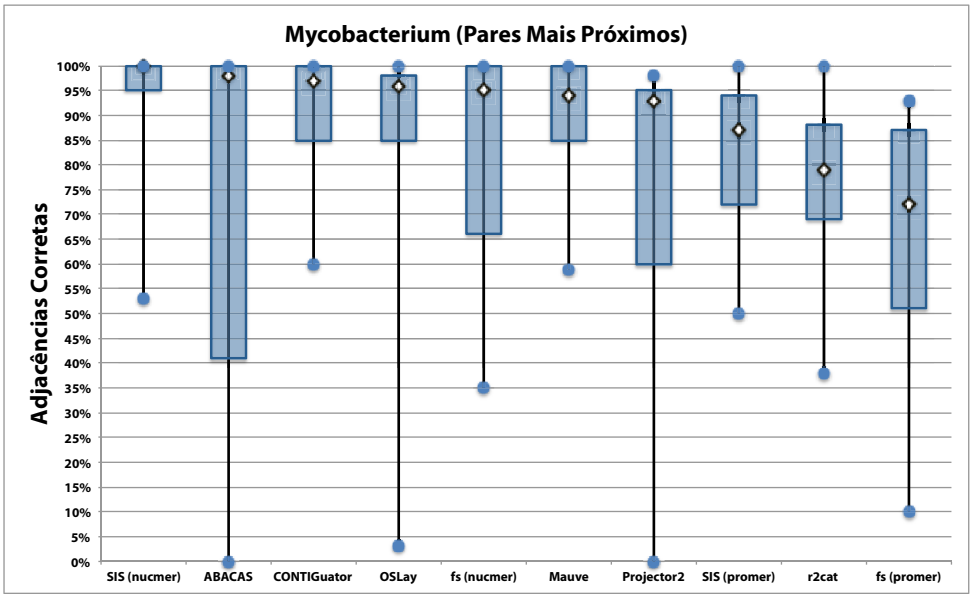


Figura 6.16: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Mycobacterium* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds.

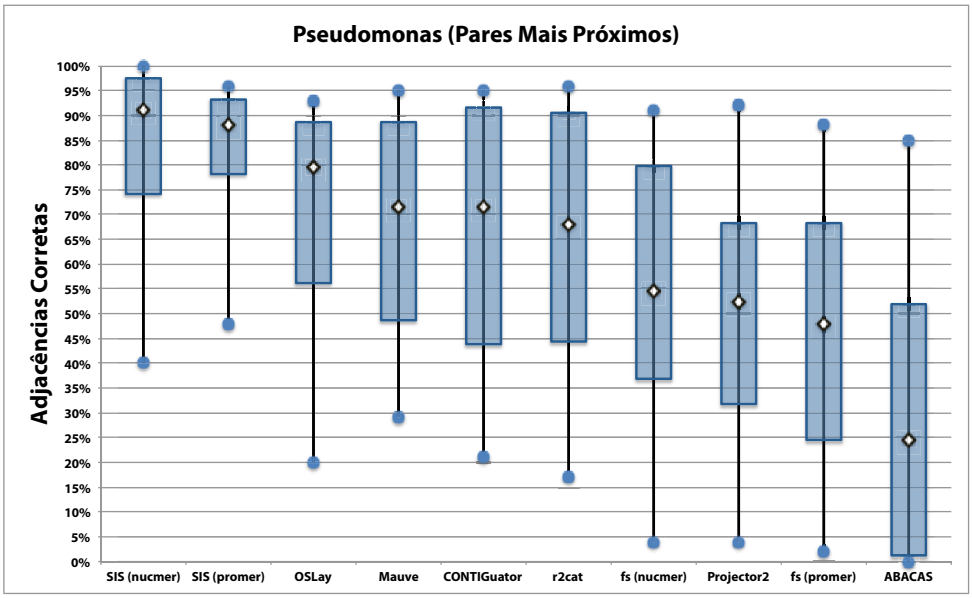


Figura 6.17: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Pseudomonadaceae* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds.

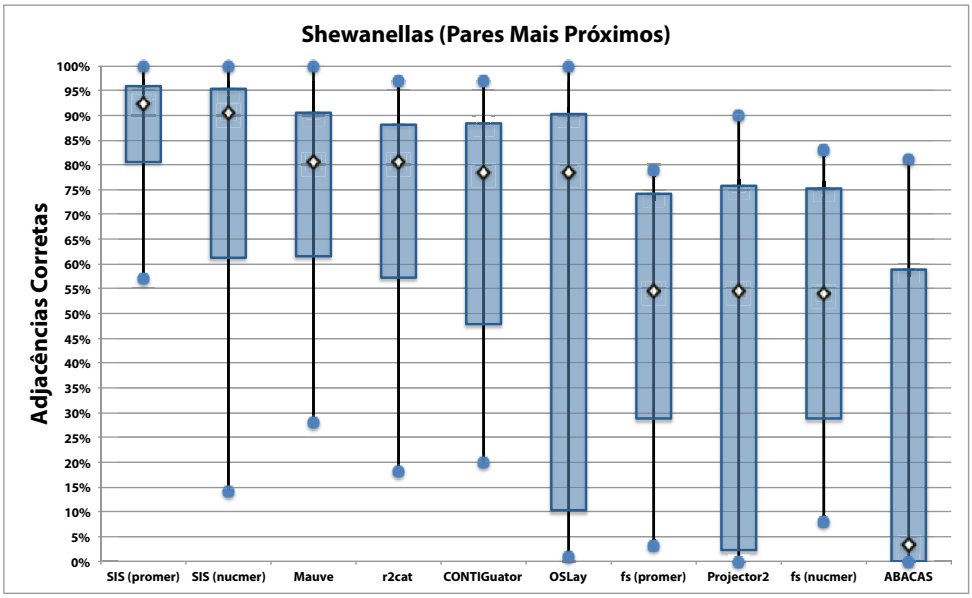


Figura 6.18: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Shewanella* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds.

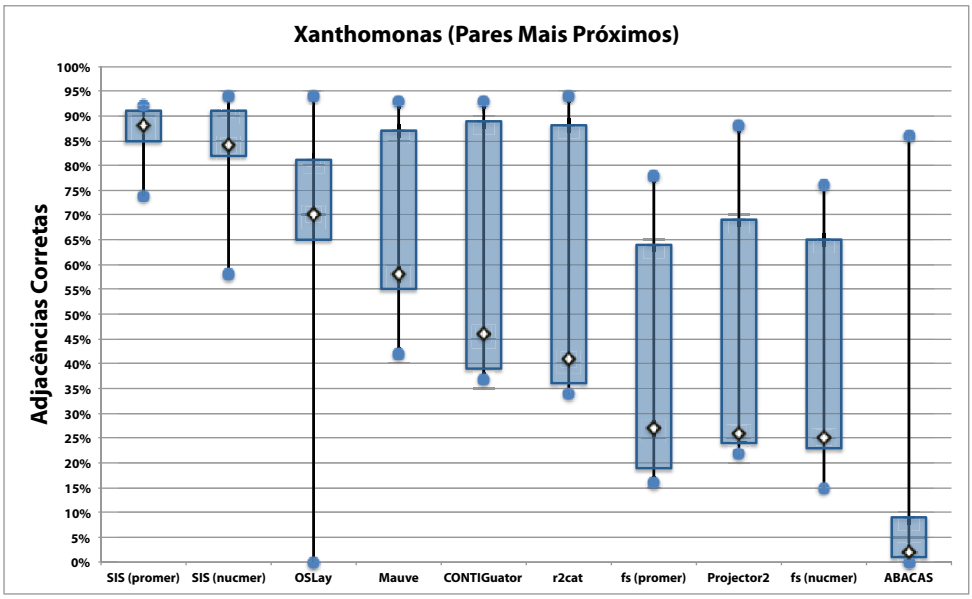


Figura 6.19: Distribuição de adjacências corretas nos *scaffolds* gerados pelos programas de reconstrução para o grupo *Xanthomonas* quando apenas os pares de genomas mais próximos são considerados. O termo **fs** foi usado para referenciar o programa fillScaffolds.

os genomas mais próximos suporta a noção intuitiva de que construir *scaffolds* a partir de genomas próximos é mais adequado do que usando genomas mais distantes. Por outro lado, quanto mais distante o genoma de referência usado para a construção dos *scaffolds*, melhores são os resultados do SIS quando comparado aos resultados dos outros programas.

Um outro ponto que precisa ser mencionado é que, apesar de SIS ser o único programa que usa assinaturas de reversão, os resultados obtidos foram superiores mesmo na presença de outros eventos de rearranjo.

Tempos de Execução

O tempo de execução de cada programa depende diretamente do tempo usado para alinhar dois genomas completos, primeiro passo de cada um dos algoritmos. Por exemplo, no caso do SIS, `nucmer` demora em média 1 minuto para cada par de genomas, enquanto que `promer` demora em média 5 minutos. O SIS, propriamente dito, demora menos de 1 segundo para computar o *s*caffold.

OSLay também utiliza o `nucmer` para computar blocos conservados e demora 3 segundos para gerar *scaffold*. O programa r2cat demora entre 15 e 20 segundos para determinar os blocos conservados e algo em torno de 2 segundos para gerar o *scaffold*. ABACAS demora de 1 a 3 minutos para completar todo o processo usando o `nucmer`. CONTIGuator

usa BLAST+ [18] e demora, em média, 2 minutos para completar todo processo. Mauve Aligner é o mais lento, demorando entre 30 minutos e 20 horas para cada par de genomas (em média, 85 minutos).

Todos os testes foram executados na mesma máquina, que possui um processador Intel Core2 Duo, 3.0 GHz e 4 GB RAM. O Projector 2, por outro lado, possui um servidor web próprio e não pode ser executado localmente. Nesse caso, observou-se que Projector 2 demora aproximadamente 1 minuto para computar o *scaffold* quando o servidor não está sobrecarregado.

6.4 Análise dos Algoritmos com Genomas Incompletos Reais

Na seção anterior, foram apresentados resultados de testes com *contigs* simulados de organismos de quatro grupos de organismos, um grupo cujo evento de rearranjo mais comum é a reversão. Nesta seção, os programas de construção de *scaffold* foram analisados com um conjunto mais heterogêneo de organismos, com representantes em 7 classes diferentes.

6.4.1 Dados de Entrada

O critério para escolha dos genomas foi a disponibilidade de dados no NCBI[1] em dezembro de 2011. Nesta data, foram selecionados todos os organismos que possuem genomas completos e genomas incompletos com mais de 4 *contigs*. Os genomas incompletos foram usados como entrada para os programas de construção de *scaffold* e os genomas completos foram usados para validar os resultados. Com essa abordagem, foram obtidos 19 organismos, sendo que 4 deles possuem 2 cromossomos avaliados individualmente, gerando um total de 23 instâncias para teste. A Tabela 6.1 lista todos os organismos usados.

Para cada um dos genomas da Tabela 6.1, foram obtidos uma lista dos 20 genomas mais próximos dentre os 1331 genomas completos de procariotos disponíveis no NCBI em maio de 2011. NUCMi foi usado para computar a distância entre cada um dos 1331 genomas de procariotos e os genomas da Tabela 6.1.

Como mencionado na seção anterior, na prática, apenas o genoma mais próximo seria considerado, mas o espaço amostral de organismos procariotos completamente sequenciados ainda é pequeno, o que torna perfeitamente possível que, no futuro, o genoma mais próximo disponível não seja tão próximo quanto ideal. Assim, é importante entender como a distância entre o genoma incompleto e o genoma de referência afeta a qualidade do *scaffold* construído.

[1] `ftp://ftp.ncbi.nih.gov/genomes/Bacteria_DRAFT`

Organismo	Cromossomo	Tamanho	Contigs	Cobertura
Aciduliprofundum boonei T469	NC_013926	1.486.778	29	89,36%
Bacillus subtilis 168	NC_000964	4.215.606	5	99,98%
Bifidobacterium longum DJO10A	NC_010816	2.375.792	43	60,02%
Brucella melitensis bv 1 16M	NC_003317	2.117.144	41	91,06%
Brucella melitensis bv 1 16M	NC_003318	1.177.787	12	99,94%
Brucella pinnipedialis B2 94	NC_015857	2.138.342	55	87,68%
Brucella pinnipedialis B2 94	NC_015858	1.260.926	34	84,58%
Burkholderia thailandensis E264	NC_007650	2.914.771	15	85,33%
Burkholderia thailandensis E264	NC_007651	3.809.201	26	75,90%
Chlamydia muridarum Nigg	NC_002620	1.072.950	4	99,09%
Clostridium cellulovorans 743B	NC_014393	5.262.222	293	94,16%
Corynebacterium aurimucosum ATCC 700975	NC_012590	2.790.189	88	85,71%
Corynebacterium efficiens YS 314	NC_004369	3.147.090	118	95,90%
Micrococcus luteus NCTC 2665	NC_012803	2.501.097	121	76,66%
Mycobacterium tuberculosis H37Ra	NC_009525	4.419.977	220	83,75%
Mycoplasma genitalium G37	NC_000908	580.076	24	78,59%
Saccharopolyspora erythraea NRRL 2338	NC_009142	8.212.805	237	96,43%
Selenomonas sputigena ATCC 35185	NC_015437	2.568.361	49	97,16%
Stigmatella aurantiaca DW4 3 1	NC_014623	10.260.756	466	97,62%
Streptococcus pneumoniae TIGR4	NC_003028	2.160.842	211	90,98%
Vibrio Ex25	NC_013456	3.259.580	176	91,77%
Vibrio Ex25	NC_013457	1.829.445	33	95,31%
Yersinia pestis Nepal516	NC_008149	4.534.590	17	84,21%

Tabela 6.1: Organismos usados nos testes. A coluna **Cobertura** contém a fração de cada cromossomo coberta pelos contigs. A coluna **Cromossomo** contém a chave de referência do cromossomo no NCBI. A coluna **Tamanho** mostra o tamanho do cromossomo em pares de base. A coluna **Contigs** mostra o número de contigs presentes no genoma incompleto.

6.4.2 Resultados

Cada um dos programas executou em todos os 23 cromossomos incompletos de entrada usando os 20 cromossomos mais próximos como referência, o que gera um total de 460 *scaffolds* por programa. Os resultados foram tabulados de acordo com o seguinte critério:

- **Top 1**: média dos resultados fornecidos por um dado programa quando apenas o genoma mais próximo é usado como referência.
- **Top 10**: média dos resultados fornecidos por um dado programa quando os 10 genomas mais próximos são usados como referência.
- **Top 20**: média dos resultados fornecidos por um dado programa quando os 20 genomas mais próximos são usados como referência.

A Tabela 6.2 mostra que, em média, a melhor performance foi obtida pelo SIS (promer) quando todos as 460 instâncias de testes são contabilizadas para cada programa.

Programas	Top 1	Top 10	Top 20
ABACAS	33,80%	23,95%	13,20%
CONTIGuator	42,02%	30,35%	20,01%
fillScaffolds (nucmer)	48,47%	34,10%	21,32%
fillScaffolds (promer)	44,19%	32,94%	21,63%
Mauve	58,65%	46,31%	32,17%
OSLay	46,83%	34,03%	21,28%
Projector 2	36,89%	25,35%	15,51%
r2cat	59,72%	44,19%	30,30%
SIS (nucmer)	56,39%	43,95%	28,55%
SIS (promer)	**60,96%**	**50,01%**	**37,27%**

Tabela 6.2: Performance de cada programa em cada uma das instâncias de testes. Os números se referem a uma média das porcentagem de adjacências corretas. O melhor resultado é realçado em negrito.

Para avaliar a performance de cada program à medida que a distância entre o genoma incompleto e o genoma de referência aumenta, foram gerados os gráficos mostrados nas figuras 6.20 e 6.21. Os gráficos mostram uma perda gradual de performance, mas o SIS (promer) se mantém com os melhores resultados.

Os resultados apresentados até o momento representam valores médios. A Tabela 6.3 apresenta resultados referentes à fração das instâncias em que cada programa fornece o melhor resultados. A soma das colunas da Tabela 6.3 é maior que 100% devido a casos de empate. Nesta análise, a superioridade do SIS (promer) em relação aos outros programas é ainda mais evidente.

Programas	Top 1	Top 10	Top 20
ABACAS	17,39%	11,74%	7,61%
CONTIGuator	13,04%	9,57%	6,30%
fillScaffolds (nucmer)	21,74%	11,30%	8,70%
fillScaffolds (promer)	0,00%	6,96%	8,04%
Mauve	26,09%	25,65%	23,26%
OSLay	21,74%	15,65%	11,30%
Projector 2	8,70%	6,09%	4,57%
r2cat	13,04%	16,09%	15,65%
SIS (nucmer)	39,13%	27,83%	20,87%
SIS (promer)	**56,52%**	**59,13%**	**64,57%**

Tabela 6.3: Porcentagem das instâncias em que cada programa forneceu os melhores resultados em termos de adjacências nos testes com *drafts* reais. As colunas não somam a 100% devido aos casos de empate. Melhor resultado realçado em negrito.

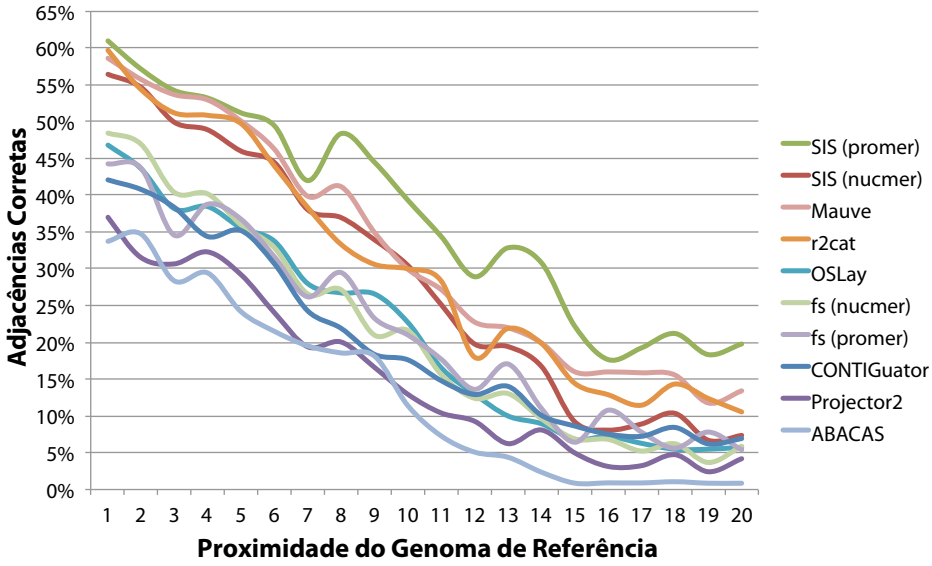


Figura 6.20: Variação de performance de cada programa quando a referência varia do genoma mais próximo para o genoma mais distante, em um total de 20 genomas. O termo **fs** foi usado para referenciar o programa fillScaffolds.

Vale ressaltar que os pares de genomas (genoma incompleto e genoma de referência) testados não foram filtrados conforme a presença de reversões. Assim, qualquer par pode conter qualquer evento de rearranjo. O sucesso de SIS sugere que reversões são, de fato, muito comuns em genomas de procariotos, partindo-se da premissa de que os genomas na Tabela 6.1 são uma amostra representativa.

Os resultados acima levam em consideração apenas a quantidade de adjacências corretas. Entretanto, é interessante analisar se os scaffolds gerados apresentam uma boa cobertura nos genomas incompletos.

A seguinte abordagem foi usada para obter uma estimativa: quando um contig possui duas adjacências corretas, considera-se que o mesmo foi corretamente posicionado e, desse modo, o tamanho total do contig é usado no cálculo da cobertura. Por outro lado, se o contig possui apenas uma adjacência correta, então apenas 50% do tamanho do contig é usado no cálculo da cobertura. Por fim, se nenhuma das adjacências ao redor do contig está correta, então o contig é descartado.

Seja x a soma dos tamanhos dos contigs utilizando o critério acima, a cobertura é dada por $\frac{x}{length}$, onde $length$ é a soma dos tamanhos de todos os contigs. A Tabela 6.4 mostra a média das coberturas (em porcentagem). É possível perceber que o SIS (promer) é o que obteve os melhores resultados, sendo seguido de perto pelo r2cat e pelo Mauve Aligner.

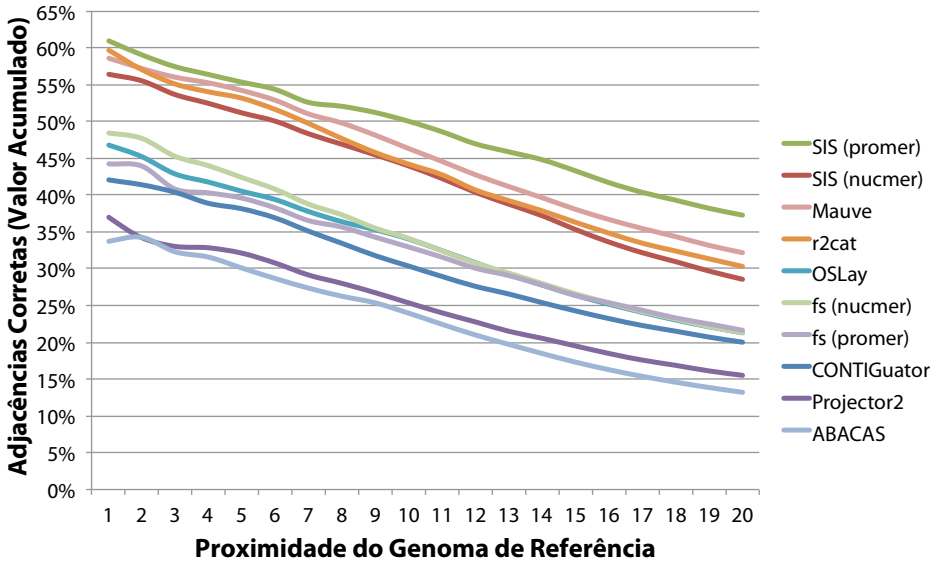


Figura 6.21: Variação de performance de cada programa quando a referência varia do genoma mais próximo para o genoma mais distante, em um total de 20 genomas. Esta é uma versão com os valores médios acumulados do gráfico exibido na Figura 6.20. O termo **fs** foi usado para referenciar o programa fillScaffolds.

SIS (promer) é também o programa que exibe o melhor resultado na grande maioria das instâncias usando a cobertura como critério (Tabela 6.5). Por exemplo, quando apenas o genoma de mais próximo é considerado, 47,83% dos melhores resultados individuais são fornecidos pelo SIS (promer), enquanto que o segundo colocado, SIS (nucmer), fornece 39,13%. Além disso, a vantagem em favor do SIS (promer) tende a aumentar à medida que o genoma de referência se torna mais distante do genoma *draft*.

Tempo de Execução

O tempo de execução dos testes desta seção foi bem próximo do observado na seção anterior. Novamente, o tempo de execução de cada programa dependeu diretamente do tempo usar para alinhar genomas completos. Os programas SIS, OSLay e r2cat, para cada instância, demoram o mesmo que demoraram nos testes anteriores: em média, 1 segundo para SIS, 3 segundos para o OSLay e 3 segundos para o fillscaffold, quando fornecidos os blocos conservados como entrada. O r2cat demora entre 15 e 20 segundos para determinar os blocos conservados e 2 segundos para gerar o *scaffold*. O CONTIGuator demora, em média, 2 minutos para terminar todo o processo.

O tempo de execução do ABACAS e do Mauve Aligner, entretanto, variou conside-

Programas	Top 1	Top 10	Top 20
ABACAS	27,82%	19,91%	11,27%
CONTIGuator	41,96%	29,34%	19,27%
fillScaffolds (nucmer)	42,94%	29,87%	18,99%
fillScaffolds (promer)	40,15%	29,23%	19,51%
Mauve	55,90%	41,85%	29,46%
OSLay	42,50%	29,03%	18,24%
Projector 2	34,29%	22,54%	13,93%
r2cat	56,81%	40,36%	27,51%
SIS (nucmer)	52,37%	39,45%	25,63%
SIS (promer)	**58,57%**	**46,01%**	**34,62%**

Tabela 6.4: Performance de cada programa em cada uma das instâncias de testes. Os números se referem a uma média das coberturas. O melhor resultado é realçado em negrito.

ravelmente. O ABACAS passou a demorar 30 segundos, enquanto que anteriormente demorava de 1 a 3 minutos. O Mauve Aligner, programa mais lento, demorou entre 8 e 322 minutos, com uma média de 46 minutos, uma mudança significativa levando-se em conta que nos testes anteriores esse programa demorara 86 minutos em média. O Projector2 seguiu o mesmo comportamento, com uma média de 1 minutos quando o servidor web não está sobrecarregado.

6.4.3 Influência das Duplicações

A ocorrência de duplicações torna mais difícil o trabalho de geração de scaffold. O efeito das duplicações no programa SIS (nucmer e promer) foi analisado de forma extensiva da seguinte maneira:

- Para cada instância de entrada (genoma de referência e genoma no estágio *draft*), verificou-se a existência de segmentos duplicados de tamanho maior ou igual a k, com k variando no conjunto $\{100, 500, 1000, 1500, 2000\}$.

- As duplicações foram determinados com a execução do programa `nucmer` nos pares de genoma. Entretanto, neste passo, não foi realizada a remoção das duplicações com o programa `delta-filter` (conforme foi feito nas análises anteriores).

- Para cada valor de k, as instâncias de entrada com a presença de duplicação foram selecionadas e os contigs onde ocorrem os segmentos duplicados de tamanho maior ou igual a k foram agrupados no conjunto D.

Programas	Top 1	Top 10	Top 20
ABACAS	13,04%	11,74%	7,83%
CONTIGuator	13,04%	10,00%	6,52%
fillScaffolds (nucmer)	17,39%	11,30%	7,39%
fillScaffolds (promer)	0,00%	6,96%	7,39%
Mauve	34,78%	25,65%	20,22%
OSLay	17,39%	12,17%	8,91%
Projector 2	8,70%	6,09%	4,57%
r2cat	13,04%	17,39%	14,35%
SIS (nucmer)	39,13%	24,35%	16,95%
SIS (promer)	**47,83%**	**51,74%**	**56,52%**

Tabela 6.5: Performance de cada programa nos testes. Os números representam a quantidade de vezes que um dado programa fornece o melhor resultado, usando como critério a cobertura. Os melhores resultados são realçados em negrito.

- As adjacências corretas foram computadas para contigs em D (Tabela 6.6) e para contigs fora de D (Tabela 6.7).

	SIS (nucmer) %				SIS (promer) %			
K	Top1	Top10	Top20	Média	Top1	Top10	Top20	Média
100	58,13	52,54	40,46	50,38	57,98	54,46	42,77	51,74
500	52,83	43,93	36,64	44,47	51,58	44,80	39,04	45,14
1000	49,93	47,75	40,71	46,13	46,25	47,90	42,09	45,41
1500	47,77	47,91	34,61	43,43	41,00	48,18	38,98	42,72
2000	42,66	34,57	24,68	33,97	29,17	39,19	32,31	33,55
Média	50,26	45,34	35,42		45,20	46,91	39,04	

Tabela 6.6: Porcentagem de adjacências corretas para contigs com duplicações. Resultados obtidos com SIS (nucmer e promer).

A comparação entre a Tabela 6.6 e a Tabela 6.7 permite concluir que houve uma queda de performance do SIS (nucmer e promer) devido a duplicações. Por exemplo, para duplicações maiores ou iguais a 1000 bases, o SIS usando o `nucmer` apresentou uma queda de performance na ordem de 12%. Usando o mesmo critério, há uma perda de aproximadamente 18% quando se usa o `promer`.

A ferramenta SIS está disponível em http://marte.ic.unicamp.br:8747.

	SIS (nucmer) %				SIS (promer) %			
K	Top1	Top10	Top20	Média	Top1	Top10	Top20	Média
100	65,11	58,69	47,19	57,00	64,57	59,98	51,85	58,80
500	63,17	51,72	42,67	52,52	61,70	53,95	47,18	54,28
1000	57,77	54,56	46,13	52,82	57,24	59,22	50,35	55,60
1500	51,50	50,84	39,17	47,17	53,07	57,50	46,19	52,25
2000	49,09	42,13	31,72	40,98	52,23	48,28	37,92	46,14
Média	57,33	51,59	41,38		57,76	55,79	46,70	

Tabela 6.7: Porcentagem de adjacências corretas para contigs sem duplicações. Resultados obtidos com SIS (nucmer e promer).

6.5 Publicações

Este capítulo foi apresentado no artigo "*Using inversion signatures to generate draft genome sequence scaffolds*" (Zanoni Dias, Ulisses Dias e João Carlos Setubal) em agosto de 2011 na conferência BCB'2011 (*ACM International Conference on Bioinformatics and Computational Biology*) realizada em Chicago, EUA [40]. Este capítulo representa uma considerável expansão do artigo apresentado. A Seção 6.4, que contém a análise com genomas incompletos reais, não consta no artigo original.

Parte IV

Distância entre Genomas

Capítulo 7

Distâncias entre Genomas

Motivação: A metodologia clássica para obter a distância evolutiva entre duas espécies se baseia na comparação de pequenas regiões muito conservadas durante o processo evolutivo. Em geral, as regiões escolhidas são um grupo de genes de manutenção comum a todas as espécies em análise. Entretanto, o advento de técnicas mais eficazes para sequenciamento de genomas resultou na disponibilização de uma grande quantidade de genomas completos, o que requer novas metodologias de classificação de genomas baseadas na comparação do genoma completo.

Metodologia: O alinhamento par-a-par de genomas permite identificar uma série de regiões conservadas, observáveis principalmente em genomas de espécies próximas. No presente capítulo, um método para calcular distâncias entre dois genomas é obtido usando a projeção dessas regiões conservadas nos genomas alinhados. As medidas de distância podem ser obtidas de forma rápida, geralmente em menos de 1 segundo, o que torna o método eficaz para geração de árvores filogenéticas a partir de um grande número de genomas. Duas medidas de distância são propostas: NUCMi, adequada para espécies próximas e PROMi, adequada para espécies distantes.

Resultados: Uma análise com 332 genomas bacteriais, abrangendo um escopo de 17 classes, concluiu que há a congruência entre as árvores filogenéticas geradas por NUCMi e PROMi e as árvores filogenéticas usadas como referência. Uma análise comparativa com outra medida, MUMi, mostra que os resultados de NUCMi e PROMi são, em média, superiores, embora equivalentes em alguns grupos de espécies.

7.1 Introdução

Com a crescente disponibilidade de genomas de organismos procariotos, surge a necessidade de métodos para comparar de maneira rápida e eficiente grandes conjuntos de genomas. Um modo de comparar dois genomas é computar uma distância entre eles. Idealmente, essa distância deve refletir as divergências evolucionárias entre ambos.

Uma distância precisa e calculável rapidamente possui diversas aplicações. Uma dessas aplicações é tema da Seção 6: dado um novo genoma incompleto, o uso do método de distância pode determinar qual é o genoma completo mais próximo, com o propósito de usá-lo como referência na montagem.

A metodologia clássica para obter uma distância entre genomas baseia-se na comparação de pequenas regiões muito conservadas durante o processo evolutivo. Em geral, as regiões escolhidas são um grupo de genes de manutenção comuns a todas as espécies em questão. Regiões como "small subunit rRNA" [100] são muito usadas como marcadores filogenéticos.

Entretanto, a metodologia clássica não faz uso de todas as informações disponíveis na forma de genomas completos. Os principais métodos que utilizam o genoma completo como fonte de informação são o "Average Nucleotide Identity" (ANI) [54, 67], "Gene Content" [91] e "MUM index" [31].

O método "ANI" [54, 67] utiliza o BLAST [3] para alinhar dois genomas completos e identificar todas as regiões altamente conservadas. Para cada um dessas regiões conservadas, calcula-se a quantidade de nucleotídeos semelhantes em proporção com o tamanho dos segmentos conservados. A distância genômica é dada pela média dos valores obtidos.

O método "Gene Content" [91] define como medida de similaridade entre duas espécies o número de genes compartilhados dividido pelo total de genes. Nos resultados apresentados pelos autores, foi utilizado o total de genes compartilhados dividido pelo número de genes do menor genoma.

Os métodos ANI e "Gene Content" não produzem resultados rapidamente, o que impede a utilização com grandes grupos de genomas. Um método eficiente foi proposto por Deloger e co-autores [31] e recebeu o nome de "MUM index", ou MUMi. O MUMi utiliza segmentos maximais conservados como ponto de partida, tais segmentos são obtidos pela ferramenta MUMmer [68]. Os autores mostram que MUMi é congruente a outros métodos como ANI [54, 67], superando nos critérios simplicidade e facilidade de obtenção. Outros métodos semelhantes ao MUMi foram propostos por Chan e co-autores [23] e Canchaya e co-autores [19].

Neste capítulo, duas medidas são propostas para comparar a sequência completa de dois genomas. Uma delas visa a comparação de genomas próximos, objetivo análogo ao do MUMi. A outra medida compara genomas mais distantes. Ambas as medidas podem

ser computadas de maneira eficiente.

As próximas seções são organizadas da seguinte forma. A Seção 7.2 apresenta as medidas para comparar genomas próximos e genomas distantes. A Seção 7.3 descreve a análise comparativa entre as medidas apresentadas neste capítulo e a medida MUMi.

7.2 Medidas de Distância

MUMi depende dos segmentos maximais conservados (MUMs - *Maximal Unique Matches*) obtidos pelo MUMmer ao comparar dois genomas. Uma restrição é imposta ao tamanho desses segmentos para que não sejam menores que 19 pares de bases, de modo a diminuir a quantidade de segmentos obtidos ao acaso. O cálculo do MUMi é baseado na fórmula:

$$\text{MUMi}(A, B) = 1 - \frac{L_{\text{mum}}}{L_{\text{av}}}$$

onde L_{mum} é a soma de todas as regiões presentes em algum MUM e L_{av} é a média dos tamanhos dos genomas comparados. Os valores do MUMi estão no intervalo [0,1], sendo que valores próximos de 0 indicam genomas mais próximos e valores próximos de 1 indicam genomas mais distantes.

As duas medidas propostas neste capítulo são chamadas NUCMi e PROMi. Ambas são baseadas na saída de programas inclusos no pacote MUMmer [68]. O programa `nucmer` (usado para o cálculo da nova medida NUCMi) realiza dois passos adicionais após a obtenção dos segmentos maximais conservados usando o MUMmer: a clusterização dos segmentos e a extensão de clusters próximos. Estes dois passos permitem a obtenção de regiões conservadas maiores com a presença de algumas remoções, inserções ou substituições de bases.

Para o cálculo do PROMi, usa-se o programa `promer`, que realiza a comparação usando a tradução da sequência de DNA em uma sequência de aminoácidos. Como a sequência de aminoácidos tende a se divergir mais lentamente que a sequência de nucleotídeos, o `promer` possui uma maior sensibilidade, encontrando mais regiões conservadas que o `nucmer`.

O resultado de `nucmer` e `promer` é uma lista de blocos conservados, onde alguns desses blocos correspondem à regiões duplicadas ou a pareamentos espúrios. Assim, a lista foi processada usando o utilitário `delta-filter` com parâmetro -1, cuja função é manter apenas os blocos que formam um mapeamento um para um entre os genomas.

Uma representação gráfica para visualizar os blocos conservados obtidos por `nucmer` e `promer` consiste em posicioná-los nas ordenadas e abscissas de um gráfico cartesiano com as origens de replicação na coordenada (0,0). Para cada bloco conservado, existe um segmento de reta delimitado pelos pontos (x_1, y_1) e (x_2, y_2). Dessa forma, é possível projetar os segmentos de reta nas ordenadas e abscissas do gráfico de modo a criar uma

cobertura nesses eixos, como mostra a Figura 7.1.

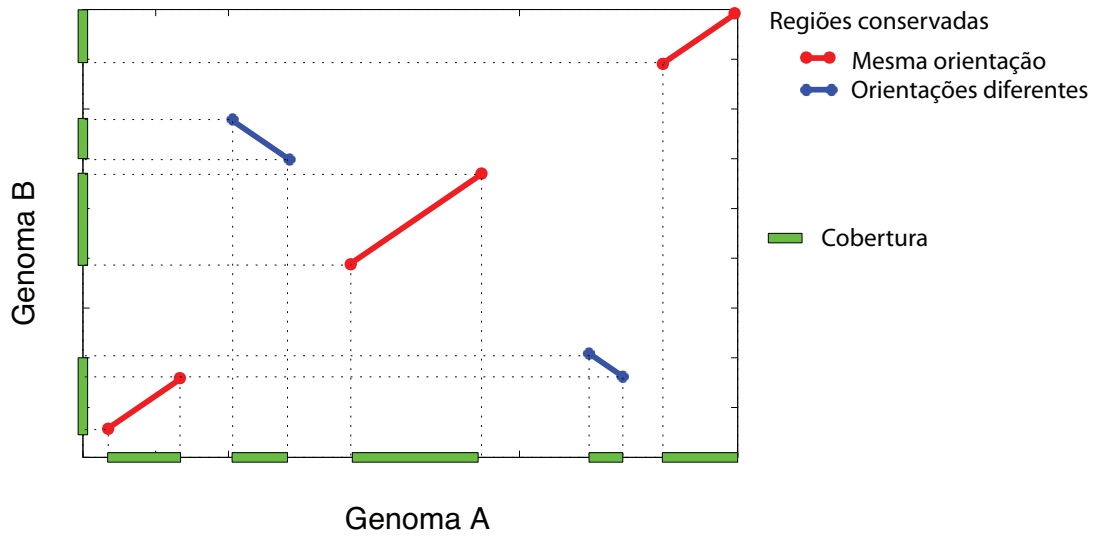


Figura 7.1: A projeção dos segmentos conservados nos eixos x e y é ilustrada com a cor verde. Essa projeção é usada para calcular as medidas NUCMi e PROMi

Seja P_x a soma dos tamanhos de todas as projeções no eixo x e P_y o análogo para o eixo y. Sejam também L_A e L_B os tamanhos dos genomas A e B, respectivamente. A seguinte fórmula é usada para obter tanto os valores do NUCMi quanto do PROMi.

$$\text{Dist}(A, B) = 1 - \frac{Px + Py}{L_A + L_B}$$

Esta distância tem o mesmo intervalo do MUMi: [0,1], onde valores próximos de 0 indicam genomas próximos, enquanto que valores próximos de 1 indicam genomas distantes.

7.3 Análise Comparativa das Medidas

Esta seção apresenta uma análise das medidas de distância usando diferentes conjuntos de genomas. Esta análise segue como critério a qualidade das distâncias geradas pelos métodos.

A validação dos resultados depende da quantidade de informação disponível como referência. Árvores de referência foram obtidas na literatura. Entretanto, na maioria dos casos, apenas a topologia da árvore é fornecida, não sendo possível recriar a matriz de distância. Nestes casos, a comparação é feita entre essas árvores com as árvores obtidas usando as medidas NUCMi, MUMi e PROMi. Em outros casos, é possível obter uma matriz

de referência à partir da árvore de referência, o que permite uma análise com a própria matriz de distância gerada pelas medidas.

Para cada grupo de genomas, uma combinação par-a-par de distâncias foi obtida usando NUCMi, PROMi e MUMi, sendo este último usado para comparação. A matriz de distância é usada para construir árvores filogenéticas com o algoritmo *neighbor-joining* [86] implementado no pacote PHYLIP [48]. A partir dessas árvores, é possível calcular a congruência em relação às árvores de referência usando um algoritmo para obtenção de uma sub-árvore consenso chamada MAST (do inglês, *maximum agreement subtree*).

A MAST consiste em uma sub-árvore contida em duas ou mais árvores de entrada e com o máximo número de folhas. A Figura 7.2 ilustra esse conceito ao apresentar duas árvores de entrada (Árvore 1 e Árvore 2), cada uma com 5 folhas rotuladas de A, B, C, D e E. A árvore consenso (MAST) é criada agrupando as folhas de modo a estar contida nas árvores de entrada. A folha rotulada com C, no entanto, não pode ser adicionada à MAST, pois ou concordará com a Árvore 1 ou concordará com a Árvore 2, não podendo concordar com ambas ao mesmo tempo. A implementação para cálculo de MAST usada neste capítulo pertence a Vienne e co-autores [94].

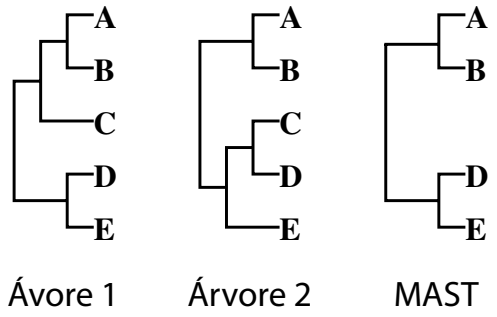


Figura 7.2: MAST calculada a partir de duas árvores de entrada (Árvore 1 e Árvore 2). A MAST é a sub-árvore com maior número de folhas contida em ambas as árvores de entrada.

Os genomas de testes foram divididos em dois conjuntos: organismos próximos e organismos distantes. Os organismos próximos correspondem às bactérias da família *Pseudomonadaceae* e dos gêneros *Mycobacterium*, *Xanthomonas* e *Shewanella*. Todos os grupos possuem árvores confiáveis publicadas na literatura. A árvore da família *Pseudomonadaceae* foi proposta por Setubal *et al.* [90], a árvore do grupo *Mycobacterium* foi proposta por Alam *et al.* [2], a árvore do grupo *Xanthomonas* foi proposta por Moreira *et al.* [77] e a árvore do grupo *Shewanella* foi proposta por Williams *et al.* [99]. As árvores de referência foram apresentadas no Capítulo 4, no lado direito das figuras 4.4, 4.5, 4.6 e 4.7

O grupo de organismos distantes foi obtido a partir da análise de Wu e co-autores [101],

que produziram uma árvore filogenética abrangendo 350 organismos de 21 classes diferentes. Entretanto, no presente trabalho, apenas 332 organismos de 17 classes foram utilizados. A filtragem dos organismos usou os seguintes critérios: disponibilidade dos organismos no NCBI e presença de no mínimo três espécies em cada classe de organismo.

A grande maioria dos organismos utilizados neste trabalho se caracteriza por possuir um único cromossomo principal e uma série de plasmídeos. Entretanto, alguns genomas são compostos por mais de um cromossomo. Nestes casos, os cromossomos foram concatenados para formar uma única sequência. Os plasmídeos foram descartados da análise, pois são geralmente considerados elementos acessórios aos genomas.

O restante desta seção é organizado da seguinte forma. A Seção 7.3.1 apresenta os resultados da análise com organismos próximos. A Seção 7.3.2 apresenta os resultados da análise com organismos distantes.

7.3.1 Análise com Organismos Próximos

A Tabela 7.1 mostra os resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas próximos. A análise usa o número de folhas presentes na árvore MAST calculada usando como entrada a árvore de referência do organismo e as árvores geradas nos testes. Quanto menor o número de espécies na MAST, pior a qualidade da árvore gerada. Se a MAST possuir o mesmo número de espécies que a árvore de referência, então a congruência é perfeita. O número de espécies na árvore de referência é dada pela coluna "Organismos" da Tabela 7.1.

O PROMi apresentou resultados inferiores, o que era de se esperar, já que essa medida é mais adequada para genomas mais distantes. Os métodos NUCMi e MUMi retornam a mesma árvore nos grupos *Shewanella* e *Xanthomonas*. Entretanto, o NUCMi é superior para a família *Pseudomonadaceae* e para o gênero *Mycobacterium*, como pode ser visto nas comparações mostradas nas Figuras 7.3 e 7.4. As arestas que participam da MAST são apresentadas em linhas sólidas, enquanto que as arestas que não pertencem à MAST são apresentadas com linhas tracejadas.

7.3.2 Análise com Organismos Distantes

A Tabela 7.2 mostra os resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas distantes. O PROMi produziu os melhores resultados na maioria dos grupos individualmente. Além disso, a coluna soma mostra que, em geral, o PROMi é superior aos demais.

Uma última análise foi realizada fornecendo o conjunto completo de 332 organismos. O PROMi foi superior aos demais com 105 espécies na MAST. O NUCMi foi o segundo colocado com 94 espécies. O MUMi obteve apenas 53 espécies na MAST.

Grupo Bacterial	Organismos	NUCMi	PROMi	MUMi
Pseudomonadaceae	18	**15**	14	14
Mycobacterium	16	**15**	14	14
Xanthomonas	9	**8**	6	**8**
Shewanella	9	**8**	7	**8**
Soma	50	46	41	44

Tabela 7.1: Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas próximos. Os resultados correspondem ao número de organismos presentes na MAST calculada usando como entrada a árvore de referência do organismo e as árvores geradas usando as três medidas. A coluna "Organismos" indica o número de espécies utilizadas. Os melhores resultados são realçados em negrito.

Grupo Bacterial	Organismos	NUCMi	PROMi	MUMi
actinobacteria	52	23	**29**	16
alphaproteobacteria	43	19	**21**	11
bacteroidetes	15	5	**8**	**8**
betaproteobacteria	18	**12**	10	9
chlamydiae	7	5	**6**	**6**
chlorobia	5	**3**	**3**	**3**
chloroflexi	5	2	2	**3**
cyanobacteria	18	8	10	**11**
deinococcus e thermus	4	**3**	**3**	**3**
deltaproteobacteria	19	10	**14**	9
epsilonproteobacteria	12	**7**	**7**	5
firmicutes	57	26	25	**27**
fusobacteriales	4	**3**	**3**	**3**
gammaproteobacteria	45	**15**	13	14
spirochaetales	7	**4**	**4**	3
tenericutes	16	8	**13**	6
thermotogae	5	**3**	**3**	**3**
Soma	332	156	174	140

Tabela 7.2: Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas distantes. Os resultados correspondem ao número de organismos presentes na MAST calculada usando como entrada a árvore de referência do organismo e as árvores geradas usando as três medidas. A coluna "Organismos" indica o número de espécies utilizadas. Os melhores resultados são realçados em negrito.

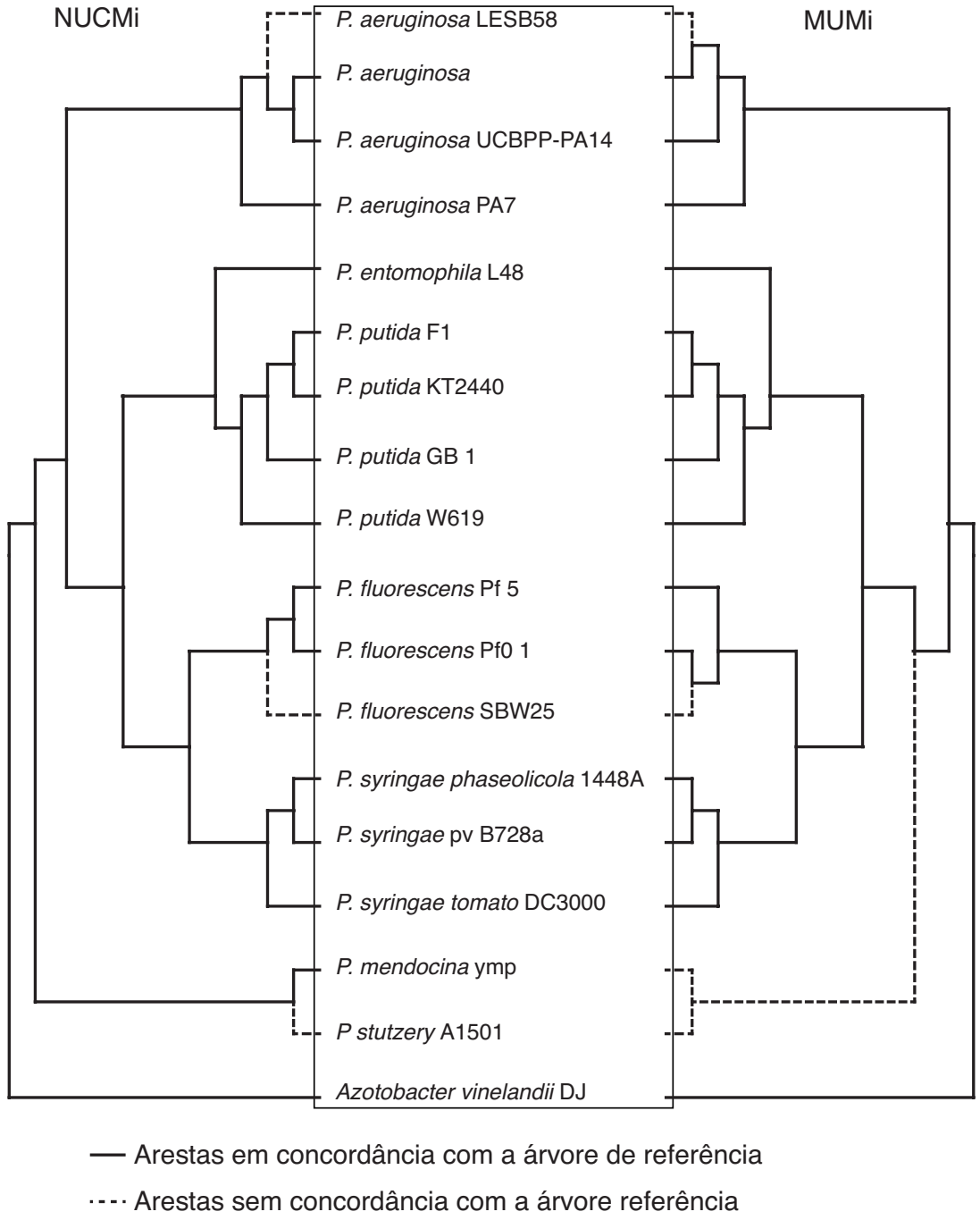


Figura 7.3: Comparação da árvore gerada pelo NUCMi e pelo MUMi com genomas de espécies próximas (família *Pseudomonadaceae*). As linhas sólidas são utilizadas para demarcar a concordância entre as árvores geradas pelas duas abordagens e a as árvores de referência (MAST), enquanto que as linhas tracejadas evidenciam as regiões com discordâncias.

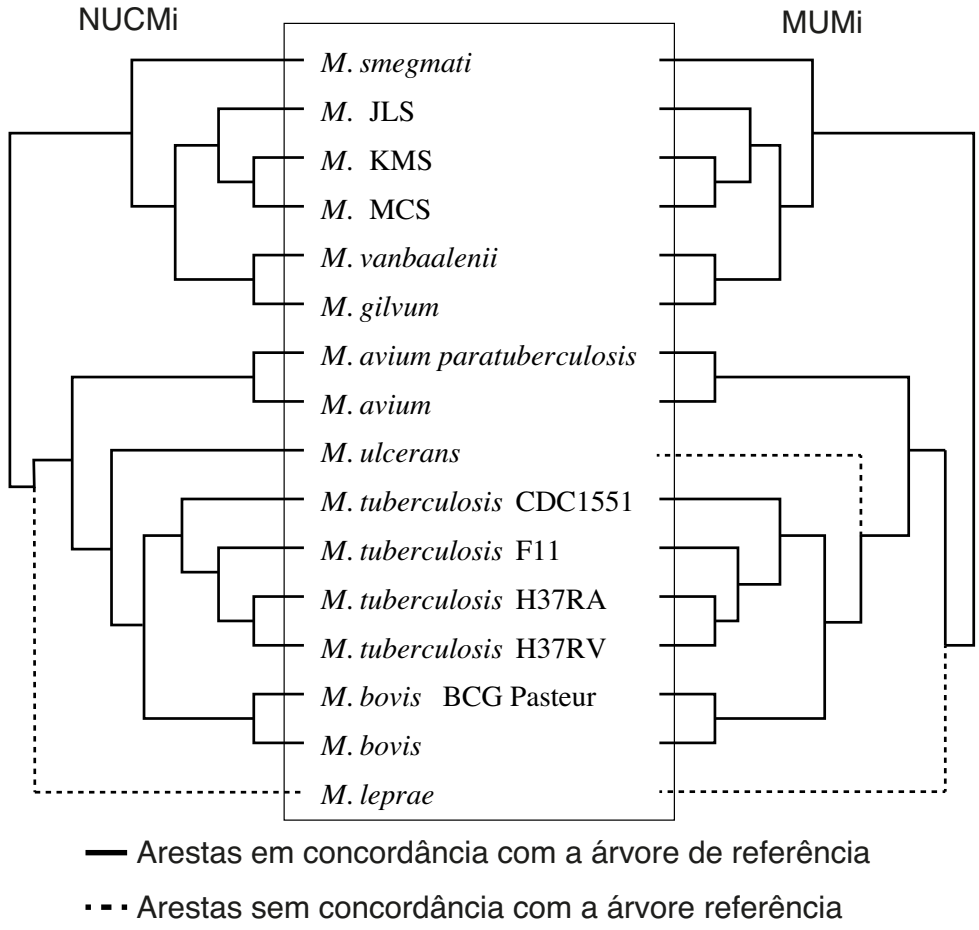


Figura 7.4: Comparação da árvore gerada pelo NUCMi e pelo MUMi com genomas de espécies próximas (gênero *Mycobacterium*). As linhas sólidas são utilizadas para demarcar a concordância entre as árvores geradas pelas duas abordagens e a as árvores de referência (MAST), enquanto que as linhas tracejadas evidenciam as regiões com discordâncias.

A análise apresentada leva em consideração apenas o posicionamento correto das espécies nas árvores. Entretanto, é importante distinguir quando algumas espécies, embora posicionadas incorretamente, estão mais próximas de outras espécies de sua própria classe. Para tanto, seja T uma árvore gerada por alguma das medidas NUCMi, PROMi ou MUMi; seja G a menor sub-árvore de T que contém todos os organismos de um grupo g da Tabela 7.2. A razão $\frac{|g|}{|G|}$, onde $|g|$ é o número de espécies no conjunto g e $|G|$ é o número de espécies na sub-árvore G, mede o quão agrupadas na árvore T estão as espécies de g. Valores próximos de 1 indicam um agrupamento mais coeso que valores próximos de 0.

A Tabela 7.3 mostra os novos valores. É importante destacar que várias classes são particularmente difíceis de agrupar. Por exemplo: actinobacteria, bacteroidetes, chlamydiae, chloroflexi, deltaproteobacteria e spirochaetales obtiveram valores menores que 0.2 nas árvores geradas pelas três medidas. Outras classes como a chlorobia e deinococcus/thermus foram agrupadas corretamente por todas as medidas. Outros padrões também emergiram como: cyanobacteria e fusobacteriales agrupados corretamente apenas por NUCMi e PROMi, e betaproteobacteria agrupada corretamente apenas por PROMi.

Em média, o PROMi realiza um trabalho mais consistente em agrupar as espécies, sendo seguido pelo NUCMi e depois pelo MUMi.

Análise com Distâncias Derivadas da Árvore de Referência

A árvore filogenética apresentada por Wu e co-autores [101] é composta apenas da topologia. Entretanto, foi fornecida pelos autores, mediante requisição, uma árvore contendo a distância entre cada nó e o seu nó pai. Essa informação extra permite realizar uma análise mais precisa do que a mostrada na Tabela 7.2.

Esta análise consiste em, primeiramente, recriar uma matriz de distância de referência R usando as medidas obtidas na árvore. Seja $Parent$ o menor ancestral comum entre dois nós A e B, a distância entre A e B é a soma da distância entre A e $Parent$ com a distância entre B e $Parent$. Esse método de construir a matriz de distância garante que a mesma será aditiva [89, Section 6.5], sendo que o algoritmo *neighbor-joining* será capaz de reconstruir corretamente a árvore original [86]. Assim, a matriz de distância obtida pelo método acima representa perfeitamente a árvore em si.

A segunda parte consiste em comparar a matriz de referência R com as matrizes obtidas por cada uma das medidas, denotadas pela letra A. Existem métodos numéricos para comparação de matrizes, mas eles não são apropriados porque focam apenas nas diferenças numéricas entre as matrizes. Deseja-se, nesta segunda parte, uma análise mais qualitativa. Dessa forma, seja $R(X, Y)$ a distância entre os genomas X e Y na matriz R e, analogamente, $A(X, Y)$ a distância entre os genomas X e Y na matriz A; adotou-se a seguinte metodologia de comparação: sejam X, Y e Z três genomas, um erro é tabulado se $A(X, Y) < A(X, Z)$ e $R(X, Y) > R(X, Z)$, ou se $A(X, Y) > A(X, Z)$ e

Grupo Bacterial	NUCMi	PROMi	MUMi
actinobacteria	0.173	0.158	0.157
alphaproteobacteria	0,651	0,439	0,129
bacteroidetes	0,046	0,058	0,050
betaproteobacteria	0,134	1,000	0,054
chlamydiae	0,023	0,027	0,021
chlorobia	1,000	1,000	1,000
chloroflexi	0,017	0,015	0,015
cyanobacteria	1,000	1,000	0,061
deinococcus e thermus	1,000	1,000	1,000
deltaproteobacteria	0,063	0,074	0,057
epsilonproteobacteria	0,857	1,000	0,104
firmicutes	0,175	0,934	0,172
fusobacteriales	1,000	1,000	0,308
gammaproteobacteria	0,336	0,196	0,135
spirochaetales	0,024	0,027	0,055
tenericutes	1,000	1,000	0,222
thermotogae	0,625	1,000	0,041
Média	0,478	0,584	0,211

Tabela 7.3: Resultados produzidos pelas medidas MUMi, NUCMi e PROMi para o grupo de genomas distantes. Os resultados correspondem à razão $\frac{|g|}{|G|}$, onde $|g|$ é o número de espécies no conjunto g e $|G|$ é o número de espécies na sub-árvore G, que é a menor sub-árvore que contém todos os organismos de g. Essa razão mede o quão agrupados na árvore T estão as espécies de g. Valores próximos de 1 indicam um agrupamento mais coeso que valores próximos de 0. A coluna "Média" contém a média simples dos valores de MUMi, NUCMi e PROMi.

$R(X,Y) < R(X,Z)$, caso contrário, um acerto será tabulado. A pontuação da matriz A é a porcentagem de acertos.

A Tabela 7.4 mostra os resultados desta nova análise, que confirmam que PROMi é a melhor medida. Vale ressaltar que, esta nova análise com genomas distantes coloca NUCMi e MUMi em situação de igualdade.

Para ser justo com MUMi e NUCMi, ressalta-se que essas medidas não foram desenvolvidas para comparar genomas tão distantes como os da Tabela 7.4. Entretanto, essa observação proporciona uma justificativa para o desenvolvimento de uma medida como o PROMi.

Grupo Bacterial	Organismos	NUCMi (%)	PROMi (%)	MUMi (%)
actinobacteria	52	76,18	81,15	69,23
alphaproteobacteria	43	72,63	80,60	75,84
bacteroidetes	15	58,11	83,99	75,22
betaproteobacteria	18	65,45	76,14	64,84
chlamydiae	7	70,95	93,33	86,67
chlorobia	5	64,44	86,67	68,89
chloroflexi	5	55,56	86,67	82,22
cyanobacteria	18	62,42	78,60	72,73
deinococcus e thermus	4	80,00	93,33	73,33
deltaproteobacteria	19	71,65	76,98	60,70
epsilonproteobacteria	12	63,78	80,42	57,58
firmicutes	57	71,72	84,17	61,15
fusobacteriales	4	66,67	66,67	33,33
gammaproteobacteria	45	50,68	79,66	65,05
spirochaetales	7	67,14	87,62	61,43
tenericutes	16	80,59	75,42	62,69
thermotogae	5	51,11	95,56	64,44
Média	19,52	66,42	82,76	66,78

Tabela 7.4: Porcentagem de acertos das medidas MUMi, NUCMi e PROMi por grupo de organismos. Valores próximos de 100 indicam uma maior semelhança com a matriz de referência. A coluna "Média" contém a média simples dos valores de MUMi, NUCMi e PROMi.

7.4 Publicações

Este capítulo foi apresentado em dois resumos estendidos na conferência BSB'2011 (*Brazilian Symposium on Bioinformatics*) realizada em Brasília [39]. O primeiro resumo estendido "*Two new whole-genome distance measures*" (Ulisses Dias, Zanoni Dias e João

C. Setubal) apresenta as medidas NUCMi e PROMi da forma definida neste capítulo. A análise utilizou a árvore consenso (MAST) como mostrado na Tabela 7.1 para genomas próximos e pela Tabela 7.2 para genomas distantes.

O segundo resumo estendido "*Evaluation of genome distance measures using tree-derived distances*" (Ulisses Dias, Zanoni Dias e João C. Setubal) descreve a metodologia para comparar as matrizes de distâncias geradas usando as medidas NUCMi, PROMi e MUMi com uma matriz de referência derivada de uma árvore de referência que informa a distância entre cada nó e o seu nó pai [37]. Este resumo estendido apresenta a Tabela 7.4 como resultado final da análise.

Capítulo 8

Considerações Finais

Esta tese relata seis trabalhos distintos na área de comparação de genomas. Para facilitar a descrição dos resultados, dividiu-se a tese em quatro partes: conceitos básicos, distância de transposição, distância de reversão simétrica e distância não vinculada a nenhum tipo específico de rearranjo.

A primeira parte, conceitos básicos, reúne uma série de trabalhos relevantes. A quantidade de informação fornecida depende da importância de cada trabalho para esta tese. Em especial, os trabalhos de Bafna e Pevzner [9] e Elias e Hartman [46] foram expostos em detalhes.

A segunda parte, distância de transposição, é constituída de duas abordagens. A primeira, mostrada no Capítulo 2, utiliza os trabalhos de Bafna e Pevzner [9] e Elias e Hartman [46] como ponto de partida para um grupo de heurísticas. Essas heurísticas compõem um algoritmo capaz de ordenar as permutações de entrada com menos transposições. Esse algoritmo foi também comparado a outros encontrados na literatura, ocasião em que se exibiu a superioridade de duas versões que usam *look-ahead* ante os demais. Entretanto, as versões que usam *look-ahead* exigem mais tempo para concluir o processamento.

A segunda abordagem, mostrada no Capítulo 3, modela a distância de transposição com técnicas de programação em lógica com restrição. Dois paradigmas foram usados: Satisfação de Restrição e Otimização de Restrição. O primeiro paradigma ofereceu melhores resultados usando o tempo de processamento como critério. Vale ressaltar que, ao contrário do algoritmo especificado na primeira abordagem, esses modelos fornecem a distância exata e não um valor aproximado.

A terceira parte desta tese descreve os avanços obtidos no contexto de reversões, em particular, reversões quase-simétricas. Três capítulos compõem essa parte. O primeiro deles, Capítulo 4, introduz uma ferramenta de simulação, SIB, para estudar propriedades do problema. A ferramenta foi validada qualitativamente e concluiu-se que realiza um bom trabalho de simulação.

O segundo capítulo relacionado a reversões, Capítulo 5, fornece um arcabouço computacional introdutório para o problema da distância de reversões quase-simétricas. Uma definição formal foi proposta e uma série de lemas e conjecturas foram apresentados. Alguns algoritmos foram implementados, tanto algoritmos gulosos para permutações genéricas quanto algoritmos exatos para algumas famílias de permutações pré-definidas. Acredita-se que esse foi um passo importante para solucionar o problema, que continua em aberto.

O Capítulo 6 foi inspirado em reversões simétricas, mas uma generalização contemplou todos os tipos de reversões na ferramenta SIS, cujo propósito é a geração de *scaffolds* usando assinaturas de reversão. Vários experimentos mostram que SIS é a melhor ferramenta para esse propósito com genomas bacteriais.

A quarta parte difere das demais por não tratar diretamente de eventos de rearranjo de genomas. O Capítulo 7 introduz duas medidas de distância entre genomas: NUCMi, adequada para espécies próximas e PROMi, adequada para espécies distantes. Essas medidas indicam a proximidade dos genomas calculando as regiões conservadas. Tais regiões estão localizadas em diferentes porções dos genomas, dado a presença de vários tipos de rearranjos. NUCMi e PROMi levam esse fator em consideração e são capazes de fornecer resultados na presença de rearranjos.

Todos os trabalhos tratados nesta tese foram apresentados em congressos importantes na área de bioinformática e publicados em seus anais. No total, foram obtidas 8 publicações, divididas em artigos completos e resumos estendidos em congressos nacionais e internacionais. A Tabela 8.1 resume os trabalhos produzidos durante esta tese.

	Congressos		
	Nacional	**Internacional**	**Total**
Trabalho Completo	[32]	[35, 38, 40, 41]	5
Resumo Estendido	[37, 39]	[36]	3
Total	3	5	8

Tabela 8.1: Resumo da produção científica.

Vários tópicos tratados nesta tese continuam em aberto. Espera-se ter produzido uma contribuição para auxiliar o entendimento dos vários mecanismos envolvidos na evolução dos genomas, particularmente por rearranjo de genomas.

Apêndice A

Revisão Bibliográfica para a Distância de Transposição

Este apêndice apresenta uma série de resumos de artigos disponíveis na literatura científica para o problema da distância de transposição. Os artigos foram escolhidos de modo a contextualizar o Capítulo 2 e o Capítulo 3. Ambos os capítulos lidam com o problema da distância de transposição, problema este que já conta com uma extensa quantidade de trabalhos publicados.

Pretende-se, com este apêndice, tornar a tese auto-contida, permitindo que a mesma seja lida e entendida sem a necessidade de textos adicionais.

A.1 Sorting by Transpositions [9]

Este trabalho é um dos primeiros a tratar o problema da distância por transposições, apresentando limitantes para o problema e um algoritmo capaz de fornecer uma resposta aproximada na razão de 1.5.

Representando um genoma por uma sequência de genes $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, uma transposição $\rho(i,j,k)$, onde $1 \leq i < j < k \leq n+1$, é uma operação em que dois blocos de genes lado a lado trocam de lugar, de modo que o intervalo $[i, j-1]$ de π seja inserido entre π_{k-1} e π_k. No trabalho de Bafna e Pevzner, assume-se que a permutação π foi estendida para incluir os pares de elementos $\pi_0 = 0$ e $\pi_{n+1} = n+1$.

Dados dois genomas π e σ, o problema da distância de transposição consiste em obter o número mínimo t de transposições $\rho_1, \rho_2, \ldots, \rho_t$ tais que $\pi\rho_1\rho_2 \ldots \rho_t = \sigma$. A distância de transposição entre π e σ é igual à distância entre $\sigma^{-1}\pi$ e a permutação identidade. Nesse caso, ordenar por transposição objetiva encontrar a distância $d(\pi)$ entre um genoma π e a identidade.

O trabalho aborda duas maneiras para lidar com o problema, uma delas com o conceito

de *breakpoints* e a outra com uma representação chamada de grafo orientado cíclico com arestas de cores alternadas, ou simplesmente grafo de ciclos. Devido à grande importância dessas abordagens e a frequência com que aparecem em trabalhos posteriores, ambas serão detalhadas nas próximas seções.

A.1.1 Breakpoints

Dada uma permutação estendida $\pi = (\pi_0\ \pi_1\ \ldots\ \pi_n\ \pi_{n+1})$, define-se um *breakpoint* como o par (π_i, π_{i+1}), $\forall i \in [0, n]$ se $\pi_{i+1} \neq \pi_i + 1$. Representa-se por $b(\pi)$ o número de breakpoints na permutação π. *Breakpoints* podem ser entendidos como os elementos que ainda precisam ser ordenados. Assim, ordenar uma permutação se resume em diminuir o número de *breakpoints* até zero, o que corresponderia a uma permutação identidade.

O conceito de *breakpoint* permite definir um limitante inferior para o problema da distância por transposição. Observa-se que, em uma dada transposição, $\rho(i, j, k)$ as modificações ocorrem apenas em três pontos da sequência, entre os elementos $[\pi_{i-1}, \pi_i]$, $[\pi_{j-1}, \pi_j]$ e $[\pi_{k-1}, \pi_k]$. Dessa forma, uma transposição pode no máximo diminuir em três a quantidade de *breakpoints*, o que gera o seguinte limitante:

$$d(\pi) \geq \frac{b(\pi)}{3}$$

A.1.2 Grafo Orientado Cíclico com Arestas de Cores Alternadas

Um grafo $G(\pi)$ orientado cíclico com arestas de cores alternadas é composto por um conjunto de $n + 2$ vértices numerados de 0 a $n + 1$ e um conjunto de $2(n + 1)$ arestas orientadas e coloridas de duas formas:

Arestas cinzas: orientadas do vértice rotulado com $i-1$ ao vértice rotulado com i, para todo $i \in [1, n + 1]$

Arestas pretas: orientadas do i-ésimo vértice ao $(i - 1)$-ésimo vértice, para todo $i \in [1, n + 1]$.

Para cada vértice existe uma aresta cinza de entrada pareada com uma aresta preta de saída, o que permite uma única decomposição do grafo em ciclos C de cores alternadas. A Figura A.1 mostra o grafo de ciclos da permutação $\pi = (4\ 2\ 5\ 3\ 1)$. O exemplo da figura não pertence ao trabalho original e foi adicionado para facilitar a compreensão do grafo de ciclos.

O tamanho de um ciclo é dado pelo número de arestas pretas. Um ciclo com k arestas pretas é dito um k-ciclo. Um ciclo pode ser classificado em par ou ímpar de acordo com

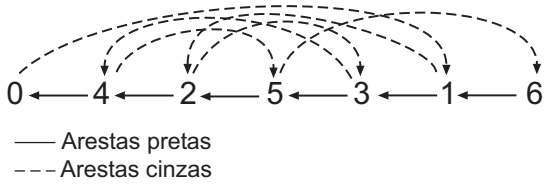


Figura A.1: Grafo de ciclos para a permutação $\pi = (4\ 2\ 5\ 3\ 1)$

seu tamanho k. Se k for par, então o ciclo é par e, por outro lado, se k for ímpar, então o ciclo é ímpar.

Como existem $2(n+1)$ arestas e a identidade possui $n+1$ ciclos de tamanho 1, conclui-se que ordenar uma permutação implica encontrar um conjunto mínimo de transposições que aumentam o número de ciclos até $n+1$. Dessa forma, denominando $c(\pi)$ o número de ciclos e $\Delta_c(\rho) = c(\pi\rho) - c(\pi)$ a variação sofrida no número de ciclos em decorrência da transposição ρ, o trabalho apresenta o Teorema A.1.

Teorema A.1 *Se após uma transposição ρ há variação no número de ciclos, então essa variação é sempre de 2, para mais ou para menos, ou seja, $\Delta c(\rho) \in \{-2,0,2\}$.*

O Teorema A.1 permite definir o seguinte limitante inferior:

$$d(\pi) \geq \frac{n+1-c(\pi)}{2}$$

A identidade possui apenas ciclos ímpares, então um limitante que leve em consideração as alterações no número de ciclos ímpares tende a ser mais preciso que o limitante obtido pelo Teorema A.1. Assim, seja $c_{odd}(\pi)$ o número de ciclos ímpares e $\Delta_{c_{odd}}(\rho) = c_{odd}(\pi\rho) - c_{odd}(\pi)$ a variação sofrida no número de ciclos ímpares em decorrência da transposição ρ, o trabalho apresenta o Teorema A.2.

Teorema A.2 *Se após uma transposição ρ há variação no número de ciclos ímpares, então essa variação é sempre de 2, para mais ou para menos, ou seja, $\Delta_{c_{odd}}(\rho) \in \{2,0,-2\}$.*

O Teorema A.2 gera o seguinte limitante inferior, mais próximo da distância real que o limitante definido anteriormente:

$$d(\pi) \geq \frac{n+1-c_{odd}(\pi)}{2}$$

A.1.3 Propriedades do Grafo de Ciclos

Para representar um ciclo C em um grafo de ciclos $G(\pi)$, numera-se as arestas pretas de 1 até $n+1$ em $G(\pi)$, onde rótulo i é atribuído à aresta preta de π_i a π_{i-1}. Após isso, lista-se os rótulos das arestas pretas na ordem em que aparecem em C.

Por exemplo, a Figura A.1 possui dois ciclos C_1 e C_2 com 4 e 2 arestas pretas respectivamente, como é mostrado na Figura A.2. Iniciando a leitura do ciclo C_1 pela aresta cinza que sai do elemento $\pi_1 = 4$, temos que os rótulos lidos no grafo $G(\pi)$ são 3,4,6,2. Dessa forma, $C_1 = (3,4,6,2)$.

Figura A.2: Ciclos C_1 e C_2 para a permutação $\pi = (4\ 2\ 5\ 3\ 1)$.

Como existem k formas de se escrever um k-ciclo, dependendo do elemento em que se inicia a leitura do ciclo em $G(\pi)$, assume-se que o primeiro rótulo do ciclo deve representar a aresta preta que está mais à direita em $G(\pi)$. Dessa forma, $C_1 = (6,2,3,4)$ e, de modo análogo, $C_2 = (5,1)$.

O Teorema A.2 afirma que uma transposição ρ pode aumentar no máximo em 2 o número de ciclos ímpares. Para agilizar a ordenação de uma permutação, é desejável maximizar o número dessas transposições. Alguns conceitos sobre ciclos são relevantes para decidir se tais transposições são possíveis.

Um ciclo $C = (i_1, i_2, \ldots, i_k)$ é dito não orientado se $i_1 < i_2 < \ldots < i_k$; caso contrário, C é dito orientado. A classificação dos ciclos em orientados e não orientados é importante, como mostram os Teoremas A.3 e A.4.

Teorema A.3 *Se existe um ciclo orientado em $G(\pi)$, então é possível encontrar uma transposição que aumente em 2 o número de ciclos.*

Teorema A.4 *Se existe um ciclo orientado em $G(\pi)$, então ou é possível encontrar uma transposição que aumenta em 2 o número de ciclos ímpares ou é possível encontrar uma sequência de três transposições que alteram em (0,2,2) o número de ciclos ímpares.*

Quando não existem ciclos orientados não é possível utilizar o Teorema A.4, mas o trabalho descreve um rigoroso estudo de caso a fim de derivar uma algoritmo capaz de fornecer uma resposta aproximada na razão de 1.5.

A.2 Sorting permutations by block-interchanges [26]

O trabalho aborda uma operação chamada troca de blocos. Uma troca de blocos é um evento que não ocorre realmente na natureza, mas cuja solução pode auxiliar na resolução do problema da distância da transposição, pois uma troca de blocos é uma generalização de uma transposição que possui solução em tempo polinomial.

Dada uma permutação $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, uma troca de blocos $\rho(i, j, k, l)$, onde $1 \leq i < j \leq k < l \leq n+1$, é uma operação em que dois blocos $[\pi_i \ldots \pi_{j-1}]$ e $[\pi_k \ldots \pi_{l-1}]$ são trocados de posição. A transposição é uma troca de blocos consecutivos onde $j = k$.

O artigo utiliza o conceito de *breakpoints* definido por Bafna e Pevzner [9]. Assim, em uma dada permutação π, acrescenta-se inicialmente dois elementos $\pi_0 = 0$ e $\pi_{n+1} = n+1$ e, após isso, define-se i como um *breakpoint* se $1 \leq i \leq n+1$ e $\pi_i - \pi_{i-1} \neq 1$. Uma operação de troca de blocos pode alterar em 4 o número de *breakpoints*.

O resultado principal do trabalho é apresentar um método que resolve o problema em tempo polinomial ao utilizar apenas o conceito de *breakpoint*, apesar de usar a estrutura de grafo de ciclos de cores alternadas para provar que o método resolve o problema de forma exata.

O algoritmo parte do princípio de que, para qualquer permutação π que não seja identidade, existe uma troca de blocos específica que remove no mínimo dois *breakpoints*. Para encontrar essa operação de troca de blocos, utiliza-se o seguinte procedimento: se π não é identidade, então existem no mínimo dois elementos x e y, tais que $x < y$ mas $\pi = [\ldots y \ldots x \ldots]$. Dessa forma, sendo x o menor valor possível em que isso ocorre, então $x-1$ deve estar à esquerda de y em π e, de modo análogo, $y+1$ deve estar à direita de x.

Assim, a operação de troca de blocos transforma a permutação $\pi = (1\ \ldots\ x-1\ \ldots\ y\ \ldots\ x\ \ldots\ y+1\ \ldots)$ em $\pi' = (1\ \ldots\ x-1\ x\ \ldots\ y\ y+1\ \ldots)$. Dessa forma, a operação possui os seguintes parâmetros: $\rho(\pi^{-1}(x-1)+1, \pi^{-1}(y)+1, \pi^{-1}(x), \pi^{-1}(y+1))$. A aplicação de trocas de blocos sucessivas com esses parâmetros ocorre até a obtenção da identidade. O artigo original provou que o algoritmo que aplica essas trocas de blocos até obter a permutação identidade ordena π com o menor número possível de operações.

A.3 Sorting a Bridge Hand [47]

O trabalho apresenta um limitante superior que demonstra ser possível ordenar qualquer permutação de tamanho $n \geq 9$ usando no máximo $\lfloor 2(n-1)/3 \rfloor$ transposições. Além disso, os autores fornecem uma expressão exata para a distância entre uma permutação e a sua reversa, notadamente $\lceil (n+1)/2 \rceil$. Por fim, o trabalho esclarece que a distância entre uma permutação e sua reversa nem sempre caracteriza o diâmetro, como conjecturava-se anteriormente [27, 74], com o primeiro contra-exemplo surgindo para $n = 13$.

A.3.1 Modelo toroidal de permutações

Estende-se uma permutação arbitrária π para uma permutação circular π° ao inserir um elemento 0, que será o predecessor de π_1 e sucessor de π_n. Por exemplo, uma permutação original (3 1 2) pode ser estendida a (0 3 1 2) = (3 1 2 0) = (1 2 0 3) = (2 0 3 1). Para obter a permutação original basta remover o elemento 0 e configurar o sucessor deste como primeiro elemento.

Partindo-se de uma permutação circular π°, define-se a operação:

$$m + \pi^\circ = (m \;\; m + \pi_1 \;\; m + \pi_2 \ldots m + \pi_n) \; (mod \; n + 1)$$

onde m é um número inteiro. Por exemplo, dada uma permutação $\pi = (3\ 1\ 2)$,

$\pi^\circ \quad = (0\ 3\ 1\ 2)$
$1 + \pi^\circ \;\; = (1\ 0\ 2\ 3)$
$2 + \pi^\circ \;\; = (2\ 1\ 3\ 0)$
$3 + \pi^\circ \;\; = (3\ 2\ 0\ 1)$

A permutação toroidal $\pi^\circ_\circ$ é a classe de equivalência gerada a partir de π°. Assim, $(3\ 1\ 2)^\circ_\circ = (2\ 3\ 1)^\circ_\circ = (2\ 1\ 3)^\circ_\circ = (1\ 3\ 2)^\circ_\circ$. É interessante notar que, dadas duas permutações circulares π° e $m + \pi^\circ$, se uma sequência de transposições transforma π° em ι, então a mesma sequência transformará $m + \pi^\circ$ em $m + \iota = \iota$.

O grupo simétrico S_n é o conjunto de todas as permutações do conjunto $\{1,2,\ldots,n\}$. Define-se o grafo *Cayley* com um conjunto de vértices S_n e uma aresta orientada $\rho(i,j,k)$ de todo $\pi \in S_n$ para $\pi\rho(i,j,k)$. Uma sequência de transposições que ordena π é um caminho de π a ι. A leitura dos rótulos do caminho gera $\pi\rho_1\rho_2\ldots\rho_l = \iota$.

A inversa de uma transposição $\rho(i,j,k)$ é uma transposição $\rho(i,j,k)^{-1} = \rho(i,r,k)$, onde $r = i + k - j$. Isso implica que $\pi = \iota\rho_l^{-1}\ldots\rho_2^{-1}\rho_1^{-1}$. Dessa forma, é fácil concluir que, se há um caminho orientado de π a σ, então existe um caminho de σ a π passando pelos mesmos vértices. Além disso, se π e σ pertencem à mesma permutação toroidal, então π^{-1} e σ^{-1} também pertencem à mesma permutação toroidal.

Ao unir qualquer par de arestas opostas no grafo *Cayley* em uma única aresta não orientada, obtém-se um grafo não orientado. Por conseguinte, ao unir em um único vértice todos os vértices que pertencem à mesma permutação toroidal é obtido o grafo toroidal.

A.3.2 Algoritmo Ótimo para a Inversa da Identidade

Seja $w_0 = (n\ n-1\ \ldots\ 2\ 1)$ a inversa da identidade, o trabalho apresenta um algoritmo ótimo capaz de efetuar a ordenação por transposição.

Teorema A.5 *Para $n \geq 3$, é possível ordenar w_0 em $\lceil (n+1)/2 \rceil$ transposições e essa ordenação é ótima.*

Inicialmente, para permutações de tamanho ímpar $n = 2k + 1$, é possível ordenar usando $k + 1$ transposições. Primeiramente, move-se o bloco $[k + 1\ k]$ para o início da permutação, logo em seguida $[k + 2\ k - 1]$ é inserido no meio do bloco recém movido. Esse invariante continua até que o último bloco a ser movido seja $[n\ 0]$, depois do qual a permutação será $(k + 1 \ldots n\ 0 \ldots k)$, um equivalente toroidal da permutação identidade. Para $n = 2k$ é possível utilizar um algoritmo similar.

É necessário, entretanto, verificar que o algoritmo é ótimo. Definindo *descent* em uma permutação π como a ocorrência de $[\pi_k\ \pi_{k+1}]$ tal que $\pi_k > \pi_{k+1}$. O trabalho apresenta o Lema A.1, cuja demonstração não será apresentada neste resumo.

Lema A.1 *O número de* descents *em uma permutação pode decrescer de no máximo dois por transposição.*

A permutação w_0 possui $n - 1$ *descents*, enquanto ι não possui nenhum. É possível perceber que a primeira e a última transposição só poderão diminuir em 1 o número de *descents*, enquanto que as transposições intermediárias do algoritmo alcançam o limitante do Lema A.1. Dessa forma, o número mínimo de transposições intermediárias é $(n-3)/2$.

A.3.3 Limitante superior para o problema do diâmetro

O diâmetro é a maior distância possível entre duas permutações de um dado tamanho n. Pode-se também definir o diâmetro como a maior distância entre ι e uma permutação π arbitrária:

$$d(n) = \max_{\pi \in S_n} d(\pi)$$

Teorema A.6 *Um limitante superior para o número de transposições necessárias para ordenar uma permutação π de tamanho n é $d(n) \leq \lfloor 2(n-1)/3 \rfloor$ para $n \geq 9$.*

A demonstração do limitante superior do Teorema A.6 está fora do escopo deste resumo por necessitar de uma série de passos adicionais.

A.3.4 Conjecturas sobre o diâmetro

Os valores de $d(n)$, para $n \leq 10$ foram calculados por um algoritmo de busca em largura aplicado na construção do grafo *Cayley*. Como o grafo *Cayley* possui $O(n^3 n!)$ arestas, uma outra heurísticas foi utilizada para determinar os valores do diâmetro para $11 \leq n \leq 15$.

Define-se um *k-move* como uma transposição ρ tal que $\pi\rho$ tem k pares $[x\ x + 1]$ a mais que π. Um contra-exemplo mínimo para a conjectura $d(n) = \lceil (n + 1)/2 \rceil$ não pode permitir um 2-*move* ou um 1-*move* seguido por um 3-*move*.

Para $n \leq 13$ foi realizada uma busca em todos as permutações toroidais que satisfazem essa restrição e, para cada uma delas, verificou-se quais podem ser ordenadas em $\lceil (n+1)/2 \rceil$ transposições. Para $n = 11$ e $n = 12$ não foram encontrados contra-exemplos, mas para $n = 13$ foi encontrada a seguinte permutação: (4 3 2 1 5 13 12 11 10 9 8 7 6).

Além dessa, outras quatro permutações que também necessitam de 8 transposições foram encontradas. Observe que w_0 de tamanho 13 pode ser ordenado com 7 transposições.

Para $n = 14$, o Teorema A.5 mostra que $d(w_0) = 8$, o que corresponde ao limitante superior do diâmetro pelo Teorema A.6. Dessa forma, nenhum cálculo precisa ser feito.

Para $n = 15$, a permutação (4 3 2 1 5 15 14 13 12 11 10 9 8 7 6) exige a ocorrência de 9 transposições, enquanto w_0 necessita apenas de 8.

O trabalho afirma que, baseado no que sugerem os experimentos, os padrões que são especialmente difíceis de ordenar para $n = 13$ e $n = 15$ deixam de ser difíceis para n maiores. Dessa forma, os autores conjecturaram que, para todo $n \geq 3$, exceto para $n = 13$ e $n = 15$, $d(n) = \lceil (n+1)/2 \rceil$.

A.4 A 1.375-Approximation Algorithm for Sorting by Transpositions [46]

O trabalho apresenta um algoritmo que fornece resposta aproximada na razão de 1.375 para o problema da ordenação por transposições. O trabalho introduz outros resultados interessantes como, por exemplo, a apresentação de um novo limitante inferior para o problema do diâmetro de transposições e a determinação exata do diâmetro para permutações simples e 2-permutações.

Muitos dos conceitos preliminares apresentados no trabalho decorrem de trabalhos anteriores e recomenda-se a leitura da Seção A.1, principalmente no que diz respeito a introdução do grafo de ciclos, que no trabalho de Elias e Hartman é chamado de grafo de *breakpoints*.

Uma técnica comum em rearranjo de genomas consiste em transformar permutações com ciclos longos (maiores que 3) em permutações simples, que são aquelas permutações que possuem ciclos de tamanho menor ou igual a 3. Quando uma permutação possui apenas ciclos de tamanho 2, ela é chamada de uma 2-permutação. Analogamente, se uma permutação possui apenas ciclos de tamanho 3, ela é chamada de uma 3-permutação.

A transformação de uma permutação em uma permutação simples é feita com a inserção de novos elementos [58]. Seja $\hat{\pi}$ a permutação obtida inserindo elementos em π, então $d(\pi) \leq d(\hat{\pi})$. A transformação é dita segura (do inglês *safe*) se ela mantém o limitante inferior do Teorema A.2. Vale ressaltar que uma transforma segura apenas mantém o limitante inferior e não a distância real.

O algoritmo apresentado no trabalho primeiramente transforma uma permutação π em uma permutação simples equivalente $\hat{\pi}$ e, em seguida, ordena $\hat{\pi}$. Dessa forma, a maior parte do trabalho é dedicada a encontrar resultados para permutações simples e ciclos curtos.

A.4.1 Diâmetro de transposição

Como os resultados são obtidos para permutações circulares, os limitantes devem ser modificados para permutações lineares. Para tanto, basta lembrar que ordenar uma permutação linear de tamanho n é equivalente a ordenar uma permutação circular de tamanho $n + 1$ [75].

Teorema A.7 *O limitante superior do diâmetro de transposição $TD(n)$ para o grupo simétrico de tamanho n é dado por:*

$$TD(n) \geq \left\lfloor \frac{n}{2} + 1 \right\rfloor$$

Teorema A.8 *O diâmetro de transposição $TD2(n)$ para 2-permutações de tamanho n é dado por:*

$$TD2(n) = \frac{n}{2}$$

Teorema A.9 *O diâmetro de transposição $TDS(n)$ para permutações simples de tamanho n é dado por:*

$$TDS(n) = \left\lfloor \frac{n}{2} \right\rfloor$$

O resultado mais importante está relacionado com o limitante superior encontrado para o diâmetro de 3-permutações $TD3(n)$, que são as permutações π que produzem um grafo de ciclos $G(\pi)$ composto por ciclos de tamanho 3. Esse limitante superior é a base para o algoritmo de aproximação 1.375. A prova exige uma rigorosa análise de casos. Entretanto, como são muitos casos, foi desenvolvido um programa de computador que sistematicamente analisa todos eles.

O objetivo da análise de casos é mostrar que para toda 3-permutação com no mínimo 8 ciclos existe uma (x,y)-sequence tal que $x \leq 11$ e $\frac{x}{y} \leq \frac{11}{8}$. A notação "$(x,y)$-sequence" de transposições indica uma sequência de x transposições, tais que no mínimo y delas são 2-moves. Por exemplo, um 0-move seguido de dois 2-moves consecutivos é denominado uma (3,2)-sequência.

O algoritmo de ordenação usa a idéia de ordenar a permutação aplicando repetidamente (11,8)-sequences e como $\frac{11}{8} = 1.375$ obtém-se a razão de aproximação.

Algoritmo 15: Elias e Hartman

Data: π
Transformar a permutação π em uma permutação simples $\hat{\pi}$
Verificar se há uma (2,2)-sequence e aplicá-la
Enquanto $G(\hat{\pi})$ possuir 2-ciclos, aplicar 2-moves
Enquanto $G(\hat{\pi})$ contém no mínimo 8 ciclos, aplicar uma (11,8)-sequence
Enquanto $G(\hat{\pi})$ contém um 3-ciclo, aplicar uma (3,2)-sequence
Gerar a ordenação de π usando a ordenação de $\hat{\pi}$

O algoritmo possui complexidade tempo de $O(n^2)$.

A.5 New Bounds and Tractable Instances for the Transposition Distance [69]

O trabalho apresenta novas classes de permutações tratáveis para o problema da distância de transposição. Também são apresentados novos limitantes para o problema.

O ponto de partida é o estabelecimento de uma conexão entre uma variante do grafo da permutação (Seção A.5.2) e do grafo de ciclos (Seção A.1). Entretanto, antes de identificar essa conexão, é necessário conhecer um pouco da nomenclatura usada. Os conceitos serão mostrados supondo conhecimento prévio sobre o exposto na Seção A.1.

A.5.1 Conceitos Iniciais

Sendo $G(\pi)$ um grafo orientado cíclico de arestas de cores alternadas criado a partir da permutação π, como visto na Seção A.1, e C um ciclo que pertence a sua decomposição única, diz-se que C é não-orientado se contém exatamente uma aresta cinza orientada da esquerda para direita; caso contrário, C é dito orientado.

Sendo $c(G(\pi))$ o número de ciclos de $G(\pi)$, define-se um *k-move* como uma transposição ρ tal que $c(G(\rho\pi)) = c(G(\pi)) + k$. Bafna e Pevzner [9] mostraram que uma transposição que age em exatamente dois ciclos é uma 0-*move*.

As três próximas definições são relacionadas ao modo como dois ciclos C_1 e C_2 interagem. Sendo I_C o intervalo definido pelo menor e maior índices dos vértices que pertencem a C, temos:

- C_1 contém C_2 se $I_{C_1} \supset I_{C_2}$ e não há arestas pretas de C_1 pertencendo a I_{C_2}.
- C_1 e C_2 se cruzam se C_1 não contém C_2 e há no mínimo uma aresta preta de C_1 pertencente a C_2.

- C_1 e C_2 se entrelaçam se, ao ler as arestas pretas de C_1 e C_2 da esquerda para a direita, uma aresta de cada ciclo é lida alternadamente.

Uma permutação π é reduzida se $b(\pi) = n-1$, $\pi_1 \neq 1$ e $\pi_n \neq n$, onde $b(\pi)$ representa o número de *breakpoints* de π (Seção A.1). Christie [27] mostrou que qualquer permutação pode ser unicamente transformada em uma permutação reduzida sem afetar a distância. Encontrar a permutação reduzida $gl(\pi)$ consiste em decompor π em r *strips*, que são intervalos maximais que não contém *breakpoints*. Depois disso, remove-se a primeira *strip* se ele começar com 1, a r-ésima *strip* se ela terminar com n e substitui-se todos as outras *strips* pelo seu menor elemento. Por fim, a sequência resultante é renumerada de modo que o k-ésimo menor número seja substituído por k.

Um teorema conhecido do trabalho de Christie [27] será útil mais adiante.

Teorema A.10 *Duas permutações π e σ são equivalentes por redução se $gl(\pi) = gl(\sigma)$. Nesse caso, $d(\pi) = d(\sigma)$.*

Um conceito importante refere-se à classe de equivalência que engloba as chamadas permutações toroidais [47]. As permutações toroidais são definidas na Seção A.3.

Define-se $\overline{x}^m$ como $\overline{x}^m = (x+m)(mod\ n+1)$. Essa operação será usada para representar a operação $m + \pi^\circ$ definida na Seção A.3, sendo π° a permutação circular obtida a partir de π:

$$m + \pi^\circ = \overline{0}^m\ \overline{\pi_1}^m\ \overline{\pi_2}^m\ \ldots\ \overline{\pi_n}^m.$$

A permutação toroidal $\pi^\circ_\circ$ é uma classe formada pelo conjunto de permutações reconstruídas a partir de todas as permutações $m + \pi^\circ$, com $0 \leq m \leq n$. Duas permutações π e σ pertencem à mesma classe toroidal se $\sigma \in \pi^\circ_\circ$ (ou $\pi \in \sigma^\circ_\circ$). Uma propriedade importante é que se π e σ pertencem à mesma classe toroidal, então $d(\pi) = d(\sigma)$. Além disso, todo ciclo em $G(\pi)$ corresponde a um ciclo em $G(\sigma)$.

A.5.2 O grafo Γ

O grafo Γ de uma permutação π é o grafo orientado com o conjunto de vértices ordenados $(\pi_1, \ldots, \pi_n)$ e arestas $(\pi_i, \pi_j)|\pi_i = j$. O grafo Γ pode ser decomposto em ciclos, também denominados C em Γ, onde $C = (i_1, i_2, \ldots, i_k)$ é um k-ciclo. Um k-ciclo é classificado como ímpar se k é ímpar e, caso contrário, o k-ciclo é classificado como par. A Figura A.3 mostra o grafo Γ construído para a permutação $\pi = (4\ 1\ 6\ 2\ 5\ 7\ 3)$.

Um k-ciclo em Γ é crescente se $k \geq 3$ e seus elementos podem ser escritos como uma sequência crescente. Por outro lado, se $k \geq 3$ e os elementos podem ser escritos como uma sequência decrescente, então o k-ciclo é decrescente. Se o k-ciclo é crescente ou

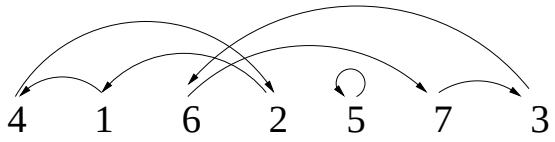


Figura A.3: Grafo Γ para a permutação $\pi = (4\ 1\ 6\ 2\ 5\ 7\ 3)$

decrescente, então o mesmo é dito monotônico e, caso contrário, não monotônico. Por exemplo, na Figura A.3 o ciclo (4 2 1) é decrescente, o ciclo (3 6 7) é crescente e o ciclo (5) é não monotônico.

É importante mencionar que as definições relacionadas ao modo como dois ciclos interagem, vistas anteriormente, podem ser facilmente adaptadas para um ciclo C em Γ.

A.5.3 Permutações γ

Uma permutação γ é uma permutação reduzida cujos elementos pares estão na posição correta. Logo, a permutação γ deve possuir um número ímpar de elementos. Assim, $\pi_i = i$ para todo elemento par π_i. Um exemplo de uma permutação γ é $\pi = (3\ 2\ 1\ 4\ 7\ 6\ 9\ 8\ 5)$, cujo grafo Γ é mostrados na Figura A.4.

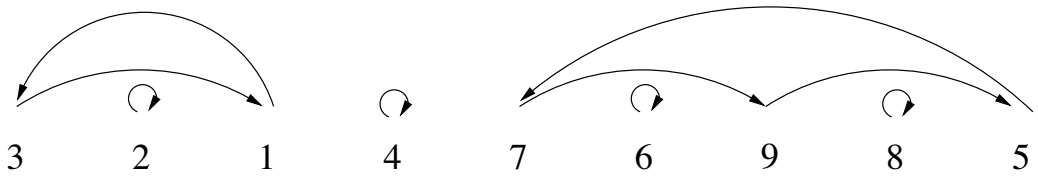


Figura A.4: Grafo $\Gamma(\pi)$ para a permutação $\pi = (3\ 2\ 1\ 4\ 7\ \ 6\ 9\ 8\ 5)$. Esta permutação é um exemplo de permutações γ, que são permutações reduzidas com os elementos pares na posição correta.

Seja π uma permutação γ arbitrária, há uma relação entre o número de ciclos pares e ímpares nos grafos $G(\pi)$ e $\Gamma(\pi)$:

$$\begin{cases} c_{even}(G(\pi)) &= 2(c_{even}(\Gamma(\pi))); \\ c_{odd}(G(\pi)) &= 2(c_{odd}(\Gamma(\pi)) - \frac{n-1}{2}). \end{cases}$$

Vale ressaltar que a classificação de C em pares e ímpares no grafo de ciclos ocorre de acordo com o número de arestas pretas, enquanto que a classificação de C em Γ em pares e ímpares está relacionada ao tamanho do ciclo k.

Uma relação útil é que, se dois ciclos C_1 e C_2 em $G(\pi)$ correspondem a um único k-ciclo C em $\Gamma(\pi)$, então C_1 e C_2 se intercalam. Além disso, ainda é possível encontrar informações mais específicas.

1. Se $k = 2$, então C_1 e C_2 são não orientados
2. Se C é monotônico, então C_1 ou C_2 são orientados.
3. Se C é não monotônico e $k \geq 4$, então C_1 e C_2 são orientados.

Com o conjunto de relações acima é possível derivar um limitante inferior para permutações γ:

$$d(\pi) \geq n - c_{odd}(\Gamma(\pi)).$$

Permutações α e β

Uma permutação α é uma permutação reduzida cujos elementos pares estão na posição correta e cujos $\frac{n+1}{2}$ elementos ímpares formam um ciclo monotônico em Γ, chamado de ciclo principal. Um exemplo de uma permutação α para $n = 7$ é (3 2 5 4 7 6 1).

É importante notar que, fixado n, só são possíveis duas permutações α. Dessa forma, para $n = 7$, uma outra permutação que atende as restrições é (7 2 1 4 3 6 5).

Uma permutação β é uma permutação reduzida cujos elementos pares estão na posição correta e cujos elementos ímpares formam um ciclo não monotônico em Γ, que, de modo análogo às permutações α, será chamado de ciclo principal.

A importância das permutações α e β é que, dado que π pertence a uma dessas classes de permutações, temos:

$$d(\pi) = n - c_{odd}(\Gamma(\pi)) = |C| - (|C| \ mod \ 2)$$

onde $|C| = \frac{n+1}{2}$ é o número de elementos no ciclo principal.

As classe de permutações α e β são restritas, mas o autor mostra que é possível generalizar o limitante acima para todas as permutações γ. Logo, pelo Teorema A.10, temos que todas as permutações que podem ser reduzidas para uma permutação γ apresentam o limitante. Além disso, o trabalho prova o seguinte teorema.

Teorema A.11 *Toda permutação σ com um número n ímpar de elementos e cujos elementos ímpares ocupam posições ímpares e formam uma subsequência crescente módulo $n + 1$ pode ser transformada em tempo linear em uma permutação π redutível à classe γ, tal que $d(\sigma) = d(\pi) = n - c_{odd}(\Gamma(\pi))$.*

Por fim, o trabalho mostra que essa equação de distância para permutações γ nada mais é do que um limite superior para a distância de transposição.

Teorema A.12 *Para todo π em S_n.*

$$d(\pi) \leq n - c_{odd}(\Gamma(\pi)).$$

Dois aperfeiçoamentos do Teorema A.12 podem ser gerados: O Teorema A.13 usa a equivalência toroidal e o Teorema A.14 usa equivalência por redução.

Teorema A.13 *Para todo π em S_n.*

$$d(\pi) \leq n - \max_{\sigma \in \pi_\circ^\circ} c_{odd}(\Gamma(\sigma)).$$

Teorema A.14 *Para todo $\pi \neq \iota$ em S_n.*

$$d(\pi) \leq m - \max_{\sigma \in (gl(\pi))_\circ^\circ} c_{odd}(\Gamma(\sigma)).$$

Onde m é o número de elementos de $gl(\pi)$.

O autor afirma que o melhoramento trazido pelo teorema A.13 em relação ao teorema A.12 é substancial segundo os experimentos, mas é difícil expressar ou avaliar numericamente esse melhoramento devido a dificuldade em prever a evolução de Γ sob a equivalência toroidal.

A.5.4 Extensões para permutações α

As permutações γ possuem a restrição de que elementos pares devem estar no lugar correto. Uma dúvida que surge é sobre o que acontece ao sistema quando essas posições fixas são removidas. Uma análise cuidadosa permite melhorar o limitante do Teorema A.12 no caso das permutações α.

Para início de análise, define-se uma permutação *k-perforation* π em S_n de uma permutação σ em S_{n+k} da classe α, ao remover $k \geq 1$ elementos pares de $\Gamma(\sigma)$ e renumerando os elementos restantes.

Por exemplo, uma 3-*perforation* da permutação σ = (3 2 5 4 7 6 9 8 11 10 1) é (3 $\underline{2}$ 5 4 7 $\underline{6}$ 9 $\underline{8}$ 11 10 1) = (2 4 3 5 6 8 7 1). Os próximos passos do trabalho consistem em analisar a evolução da estrutura de G quando é aplicada uma *k-perforation* em permutações α.

Para toda *k-perforation* π de uma permutação σ da classe α:

$$c(G(\pi)) = c_{odd}(G(\pi)) = k$$

e $G(\pi)$ contém apenas ciclos que não cruzam, exceto por um ciclo maior que contém todos os outros. Isso leva a uma fórmula para computar a distância dessas permutações:

$$d(\pi) = n - c_{odd}(\Gamma(\pi)) - k + (|C| \; mod \; 2),$$

onde $|C| = \frac{n+k+1}{2}$ é o número de elementos no ciclo principal.

O trabalho apresenta outros resultados menos genéricos e apresenta questões em aberto como, por exemplo, analisar *k-perforations* em permutações β e classificar as permutações cujo grafo Γ contém apenas ciclos que se cruzam e não podem ser reduzidas para permutações γ.

A.6 Polynomial-sized ILP Models for Rearrangement Distance Problems [43]

O trabalho apresenta um modelo para a distância de rearranjo de genomas usando Programação Linear Inteira (PLI). O modelo é o primeiro conhecido a possuir tamanho polinomial. De uma maneira mais específica, o trabalho é restrito aos problemas das distâncias de reversão e transposição, além da distância quando ambos os eventos são permitidos.

Os conceitos relacionados ao domínio de rearranjo de genomas são fortemente baseados nos trabalhos de Bafna e Pevzner sobre transposições [7] e sobre reversões [6], além de Hannenhalli e Pevzner [56].

Um genoma arbitrário formado por n genes será representado como uma permutação $\pi = (\pi_1 \; \pi_2 \; \ldots \; \pi_n)$, onde cada elemento de π representa um gene. O genoma identidade ι é definido como $\iota_n = (1 \; 2 \; \ldots \; n)$.

Utilizando essa notação, define-se os eventos de transposição e reversão da seguinte maneira: uma transposição $\rho(x, y, z)$, onde $1 \leq x < y < z \leq n + 1$, é um rearranjo que transforma π em $\rho\pi = (\pi_1 \; \ldots \; \pi_{x-1} \; \pi_y \; \ldots \; \pi_{z-1} \; \pi_x \; \ldots \; \pi_{y-1} \; \; \pi_z \; \ldots \; \pi_n)$.

Uma reversão $\rho(x, y)$, onde $1 \leq x < y \leq n$, é um evento de rearranjo que transforma π em $\rho\pi = (\pi_1 \; \ldots \; \pi_{x-1}\pi_y \; \pi_{y-1} \; \ldots \; \pi_{x+1} \; \pi_x \; \pi_{y+1} \; \ldots \; \pi_n)$. É importante notar a diferença no número de parâmetros entre os eventos de transposição e reversão, respectivamente 3 e 2. Essa diferença será importante para o modelo PLI.

Um modelo PLI é baseado em formulações matemáticas. O trabalho divide essas equações em dois grupos. O primeiro contém as equações matemáticas que introduzem as variáveis e restrições que são comuns a todos os modelos de distância e cujo papel é garantir que apenas as permutações válidas serão contempladas pelo modelo. A Seção A.6.1 define essas equações.

O segundo grupo envolve algumas restrições específicas para transposição (Seção A.6.2) e reversão (Seção A.6.3). As restrições específicas para os eventos de transposição e reversão serão utilizadas na modelagem da distância quando ambos os os eventos são

permitidos.

Nas equações seguintes, representa-se por motivos de clareza um elemento π_i por $\pi[i]$.

A.6.1 Restrições comuns a todos os modelos

As variáveis binárias B_{ijk} indicam se a i-ésima posição de π possui valor j após a k-ésima operação, onde $1 \leq i, j \leq n$ e $0 \leq k < n$:

$$B_{ijk} = \begin{cases} 1, & \text{se } \pi[i] = j \text{ após a } k\text{-ésima operação} \\ 0, & \text{caso contrário.} \end{cases}$$

Dadas essas variáveis, as seguintes restrições garantem que a permutação inicial e final são corretas, respectivamente:

$$B_{i,\pi[i],0} = 1, \text{ para todo } 1 \leq i \leq n.$$

$$B_{i,\sigma[i],n-1} = 1, \text{ para todo } 1 \leq i \leq n.$$

É preciso, no entanto, garantir que as permutaçãos intermediárias serão válidas. Para isso, são estabelecidas duas restrições, a primeira é que qualquer posição de uma permutação possui exatamente um valor associado a ela.

$$\sum_{j=1}^{n} B_{ijk} = 1, \text{ para todo } 1 \leq i \leq n,\ 0 \leq k < n.$$

A segunda restrição deve garantir que todos os valores no intervalo $[1..n]$ sejam atribuídos a alguma posição da permutação.

$$\sum_{i=1}^{n} B_{ijk} = 1, \text{ para todo } 1 \leq j \leq n,\ 0 \leq k \leq n.$$

A.6.2 Restrições específicas para transposições

Para as equações que representam uma transposição, utilizou-se as variável binárias t_{abck}, definidas para todo $1 \leq a < b < c \leq n+1$ e todo $1 \leq k < n$. A variável t_{abck} indica se a k-ésima transposição é um evento de troca entre os blocos $\pi[a \ldots b-1]$ e $\pi[b \ldots c-1]$ de π.

$$t_{abck} = \begin{cases} 1, & \text{se } \rho_k = \rho(a,b,c) \\ 0, & \text{caso contrário.} \end{cases}$$

A partir da variável t_{abck} é possível definir a variável binária t_k, que é usada para decidir se a k-ésima transposição modificou a permutação.

$$t_k = \begin{cases} 1, & \text{se } \rho_k = \rho(x,y,z) \text{ e } \rho_k\rho_{k-1}\dots\rho_1\pi \neq \rho_{k-1}\dots\rho_1\pi \\ 0, & \text{caso contrário.} \end{cases}$$

Dado que a transposição não modificou uma permutação, as transposições subsequentes não podem modificá-la também.

$$t_k \leq t_{k-1}, \text{para todo } 1 \leq k < n.$$

Na Seção A.6.4 serão criadas restrições para a distância quando os eventos de transposição e reversão são permitidos. É interessante ressaltar que a restrição acima não será incluída naquele conjunto de restrições.

Uma última restrição é necessária para impor que, em cada passo, no máximo uma transposição seja executada.

$$\sum_{a=1}^{n-1}\sum_{b=a+1}^{n}\sum_{c=b+1}^{n+1} t_{abck} \leq t_k, \text{ para todo } 1 \leq k < n.$$

Com base nas restrições acima, é possível gerar uma nova restrição que reflete as modificações na permutação causadas por eventos de transposição.

$$\sum_{a=i+1}^{n-1}\sum_{b=a+1}^{n}\sum_{c=b+1}^{n+1} t_{abck} + \sum_{a=1}^{n-1}\sum_{b=a+1}^{n}\sum_{c=b+1}^{i} t_{abck} + (1 - t_k) + B_{ijk-1} - B_{ijk} \leq 1,$$

para todo $1 \leq i, j \leq n$, e todo $1 \leq k < n$.

Os autores dividem a análise do significado dessa inequação em três casos e apresentam para cada caso a prova de que a fórmula está correta.

1. Os valores nas posições $i < a$ ou $i \geq c$ permanecem inalterados.

2. As posições $a \leq i < a + c - b$ são preenchidas com elementos que estavam em $[b, c-1]$.

3. As posições $a+c-b \leq i < c$ são preenchidos com elementos que estavam em $[a, b-1]$.

Por fim, é possível inserir restrições obtidas a partir de limitantes apresentados por Bafna e Pevzner [7]. Como esses limitantes são específicos para a distância de transposição, as duas restrições abaixo não participarão da modelagem da Seção A.6.4.

$$t_k * n + k - 1 \geq LB(\pi, \sigma), \text{ para todo } 1 \leq \text{k} \leq \text{n}.$$

$$t_k * k \leq UB(\pi, \sigma), \text{ para todo } 1 \leq \text{k} \leq \text{n}.$$

onde $LB(\pi, \sigma)$ e $UB(\pi, \sigma)$ são, respectivamente, os limitantes inferiores e superiores obtidos por Bafna e Pevzner.

O modelo PLI para eventos de transposição descrito nesta seção possui um conjunto de variáveis e restrições de ordem polinomial no tamanho da permutação fornecida como entrada. O número de variáveis é $O(n^4)$ e o número de restrições é $O(n^6)$.

A.6.3 Restrições específicas para reversões

Para as equações que representam reversões, utilizou-se a variável binária r_{abk} definidas para todo $1 \leq a < b \leq n$ e todo $1 \leq k < n$. A variável r_{abk} indica se a k-ésima reversão afeta o bloco $\pi[a \dots b]$ de π.

$$r_{abk} = \begin{cases} 1, & \text{se } \rho_k = \rho(a, b) \\ 0, & \text{caso contrário.} \end{cases}$$

A partir da variável r_{abk} é possível definir a variável binária r_k, que é usada para decidir se a k-ésima reversão modificou a permutação.

$$r_k = \begin{cases} 1, & \text{se } \rho_k = \rho(x, y) \text{ e } \rho_k \rho_{k-1} \dots \rho_1 \pi \neq \rho_{k-1} \dots \rho_1 \pi \\ 0, & \text{caso contrário.} \end{cases}$$

Dado que a reversão não modificou uma permutação, as transposições subsequentes não podem modificá-la também.

$$r_k \leq r_{k-1}, \text{ para todo } 1 \leq k < n.$$

A restrição acima não será utilizada na modelagem da Seção A.6.4.

Uma última restrição é necessária para impor que em cada passo no máximo uma reversão seja executada.

$$\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} r_{abk} \leq r_k, \text{ para todo } 1 \leq k < n.$$

Com base nas restrições acima, é possível gerar uma nova restrição que reflete as modificações na permutação causadas por eventos de reversão.

Os autores inserem duas novas restrições que lidam com o problema da distância de reversão propriamente dito:

- Os valores nas posições $i < a$ ou $i > b$ permanecem inalterados:

$$\sum_{a=i+1}^{n-1}\sum_{b=a+1}^{n} r_{abk} + \sum_{a=1}^{n-1}\sum_{b=a+1}^{i-1} r_{abk} + B_{i,j,k-1} + (1 - r_k) - B_{ijk} \leq 1,$$

para todo $1 \leq i, j \leq n$, e todo $1 \leq k < n$.

- As posições $a \leq i \leq b$ sofrem reversão de seus elementos:

$$r_{abk} + B_{b+a-i,j,k-1} - B_{ijk} \leq 1,$$

para todo $1 \leq a < b \leq n$, $a \leq i \leq b$, $1 \leq j \leq n$ e $1 \leq k < n$.

O modelo PLI para eventos de reversão descrito nesta seção possui um conjunto de variáveis e restrições de ordem polinomial no tamanho da permutação fornecida como entrada. O número de variáveis é $O(n^3)$ e o número de restrições é $O(n^5)$.

A.6.4 Distância de transposição e reversão

Para formular o problema da distância de transposição e reversão, usa-se as variáveis e restrições definidas anteriormente, exceto, como já fora dito, aquelas em que, no momento de sua definição, foi ressaltado que eram específicas para o modelo em questão.

Inclui-se a variável z_k para denotar se a k-ésima operação, que poderia ser uma reversão ou uma transposição, modifica a permutação:

$$z_k = \begin{cases} 1, & \text{se } \rho_k = \rho(x,y) \text{ ou } \rho_k = \rho(x,y,z) \text{ e } \rho_k\rho_{k-1}\ldots\rho_1\pi \neq \rho_{k-1}\ldots\rho_1\pi \\ 0, & \text{caso contrário.} \end{cases}$$

Duas inequações são ainda necessárias. A primeira indica que se um dado evento de rearranjo não modificou a permutação, então os eventos seguintes não a modificam também.

$$z_k \leq z_{k-1}, \text{ para todo } 1 \leq k < n.$$

A restrição seguinte garante que apenas um evento de rearranjo (ou reversão ou transposição), serão executados em cada passo.

$$r_k + t_k = z_k, \text{ para todo } 1 \leq k < b.$$

O modelo PLI para eventos de reversão e transposição possui um conjunto de variáveis e restrições de ordem polinomial no tamanho da permutação fornecida como entrada. O número de variáveis é $O(n^4)$ e o número de restrições é $O(n^6)$.

No restante do trabalho, os autores detalham os experimentos computacionais realizados com o modelo. O resultado mais significativo é que o modelo, apesar de polinomial no tamanho, é pouco viável devido o tempo que demora para terminar a execução.

A.7 Faster Algorithms for Sorting by Transpositions and Sorting by Block Interchanges [49]

O trabalho apresenta uma nova estrutura de dados, chamada de árvore de permutação, com o intuito de melhorar o tempo de execução do algoritmo de ordenação por transposições de Hartman e Shamir [57] de $O(n^{\frac{3}{2}}\sqrt{\log n})$ para $O(n \log n)$ e do algoritmo de ordenação por troca de blocos de Christie [26] de $O(n^2)$ para $O(n \log n)$. No presente resumo, não serão mostrados os algoritmos apresentados no trabalho, mas apenas a estrutura de dados criada.

A.7.1 Árvore de Permutação

Uma árvore de permutação é uma árvore binária balanceada T com raiz r e onde cada nó da árvore possui um rótulo. Seja $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ a permutação para qual a árvore T foi gerada, então T possui n nós folhas que são rotulados como $\pi_1, \pi_2, \ldots, \pi_n$. Os nós internos da árvore são rotulados com o maior dos rótulos dos filhos. A Figura A.5 corresponde à árvore de permutação gerada para $\pi = (9\ 6\ 1\ 4\ 7\ 5\ 2\ 3\ 8)$.

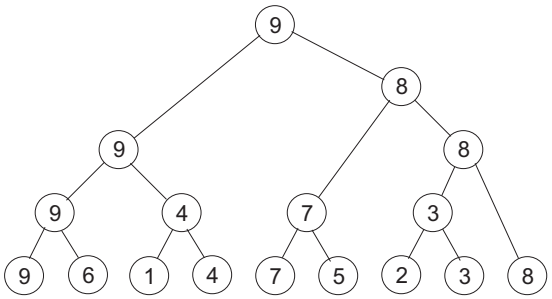


Figura A.5: Árvore de permutação para $\pi = (9\ 6\ 1\ 4\ 7\ 5\ 2\ 3\ 8)$

Cada nó interno da árvore representa um intervalo na permutação π, e o nó raiz r representa a permutação inteira. É interessante notar que não é necessário inserir ou remover nós da árvore de permutação após a mesma ser criada, pois os efeitos decorrentes de uma transposição podem ser projetados na árvore com a utilização de apenas três operações: *build*, *join* e *split*.

A operação *build* constrói uma árvore a partir de uma permutação e possui uma implementação simples: a árvore é criada das folhas até a raiz, camada por camada. Para a criação dos nós internos, usa-se a camada mais abaixo que foi criada no passo anterior.

O algoritmo *build* é mostrado abaixo. Para facilitar a legibilidade, o trabalho omite os cálculos de atribuição do rótulo e da altura dos nós. A complexidade do algoritmo *build* é $O(n)$.

Algoritmo 16: Build

Data: π
Criar nós folhas u_i para $\pi_i, 1 \leq i \leq n$
$U = [u_1, u_2, \ldots, u_n]$
$k = n$
while $k > 1$ **do**
  Criar $v_i : L(v_i) = u_{2i-1}, R(v_i) = u_{2i}$ para $1 \leq i \leq \lfloor k/2 \rfloor$
  if k *é par* **then**
    $U = [v_1, v_2, \ldots, v_{k/2}]$
  end
  else
    Criar $v : L(v) = v_{\lfloor k/2 \rfloor}, R(v) = u_k$
    $U = [v_i | 1 \leq i < \lfloor k/2 \rfloor] \cup [v]$
  end
  $k = \lfloor k/2 \rfloor$
end
return U

A operação *join* recebe duas árvores T_1 e T_2, que representam as permutações π_1 e π_2, respectivamente. A operação gera a árvore T que representa a permutação π obtida pela concatenação dos elementos de π_2 em π_1.

Vale ressaltar que T deve ser uma árvore balanceada, ou seja, se a altura $H(T_1)$ de T_1 for maior que a altura $H(T_2)$ de T_2 acrescida de 1, então a árvore resultante não poderá ser construída apenas inserindo um nó raiz r e adicionando T_1 e T_2 como filhos. O algoritmo abaixo mostra os passos para criar a árvore T quando $H(T_1) \geq H(T_2)$. A complexidade do algoritmo *join* é $O(H(T_1) - H(T_2))$.

Algoritmo 17: Join

Data: T_1, T_2
if $H(T_1) - H(T_2) \leq 1$ **then**
  Criar um nó $T : L(T) = T_1, R(t) = T_2$
  return T
end
if $H(T_1) - H(T_2) = 2$ **then**
  if $H(L(T_1)) \geq H(R(T_1))$ **then**
    Criar dois nós T' e T
    $T' : L(T') = R(T_1), R(T') = T_2$
    $T : L(T) = L(T_1), R(T) = T'$
    return T
  end
  else
    Criar três nós T'', T' e T
    $T'' : L(T'') = R(R(T_1)), R(T'') = T_2$
    $T' : L(T') = L(T_1), R(T') = L(R(T_1))$
    $T : L(T) = T', R(T) = T''$
    return T
  end
end
if $H(T_1) - H(T_2) > 2$ **then**
  $T = Join(L(T_1), Join(R(T_1), T_2))$
  return T
end

A operação *split* recebe uma árvore T, que representa uma permutação π de tamanho n e uma posição m, onde $1 \leq m \leq n$. A operação retorna duas árvores T_r e T_l. A árvore T_r representa a permutação $\pi_r = (\pi_1 \ \ldots \ \pi_{m-1})$ e a árvore T_l representa a permutação $\pi_l = (\pi_m \ \ldots \ \pi_n)$.

A idéia principal para implementar a operação *split* é encontrar em T o caminho $P = v_0, v_1, \ldots, v_k$ que vai de π_m até a raiz r, dessa forma $v_0 = \pi_m$ e $v_k = \pi_r$. Calculado o caminho P, percorre-se os elementos v_i verificando se o elemento anterior v_{i-1} está no lado esquerdo ou direito de v_i. Se o elemento v_{i-1} está do lado esquerdo, então o elemento do lado direito de v_i pertencerá à árvore T_r. De modo análogo, se o elemento v_{i-1} está no lado direito de v_i, então o elemento do lado esquerdo de v_i pertencerá à árvore T_l. O algoritmo abaixo mostra os passos para dividir a árvore. O algoritmo *split* executa com complexidade $O(\log n)$.

Além das operações acima, para utilizar a árvore de permutação será preciso calcular

Algoritmo 18: Split

Data: T, m
Encontrar o caminho $P = v_0, v_1, \ldots, v_k$ de π_m a r
$T_r = v_0$
$T_l = null$
for $i = 1 \;\; To \;\; k$ **do**
  if $L(v_i) = v_{i-1}$ **then**
    $T_r = Join(T_r, R(v_i))$
  end
  if $R(v_i) = v_{i-1}$ **then**
    $T_l = Join(L(v_i), T_l)$
  end
end
return T_r, T_l

a posição de um nó de T na permutação π. Para isso, utiliza-se a noção de que cada nó interno de T corresponde a um intervalo na permutação π. Dessa forma, cada nó t deverá guardar o número de elementos $e(t)$ representados por ele.

Com a informação $e(t)$ armazenada em cada nó, a posição em π do nó rotulado com π_i pode ser obtida percorrendo o caminho de π_i até a raiz da árvore e contando o número de elementos à esquerda de π_i. Esse cálculo pode ser efetuado em $O(\log n)$ para qualquer elemento da árvore T.

A.8 An Alternative Algebraic Formalism for Genome Rearrangements [73]

O trabalho apresenta um novo formalismo capaz de representar problemas em rearranjo de genomas. O formalismo diminui a dependência de representações visuais na exposição de argumentos, provas e teoremas.

No método clássico, representa-se um genoma como a permutação $\pi = (\pi_1 \; \pi_2 \; \ldots \; \pi_n)$, onde π é uma função tal que $\pi(1) = \pi_1, \pi(2) = \pi_2, \ldots, \pi(n) = \pi_n$. O formalismo algébrico propõe que a mesma permutação π seja vista como uma função que mapeia um dado elemento da permutação no elemento seguinte, em outras palavras $\pi(\pi_1) = \pi_2, \pi(\pi_2) = \pi_3$ e assim por diante. Nessa nova interpretação, o último elemento π_n é mapeado no primeiro: $\pi(\pi_n) = \pi_1$.

A motivação desse novo formalismo é aplicar diretamente alguns resultados deduzidos pela teoria clássica e definir algebricamente, ao invés de graficamente, alguns conceitos importantes como, por exemplo, ciclos bons e ruins, arestas pretas e cinzas, breakpoints

e grafos de breakpoints.

A.8.1 Ciclos

Dado um conjunto base E, uma permutação em E é uma função injetora $f : E \to E$. As permutação em E são compostas de ciclos, por exemplo, dado $E = \{a, b, c, d\}$, a permutação dada pelo ciclo $\alpha = (a\, b\, c)$ significa que a é mapeado em b, que é mapeado em c, que é mapeado em a. Ao elemento d, que não aparece explicitamente, entende-se que é um ciclo unitário (d), em que d é mapeado nele mesmo.

Duas operações com ciclos são de grande importância ao se utilizar o formalismo algébrico: o produto e a conjugação. O produto de dois ciclos α e β, denotado por $\alpha\beta$, é computado como segue: fixa-se um elemento x no ciclo do lado direito e obtém-se sua imagem y, a imagem y é usada como entrada no segundo ciclo da direita para a esquerda e obtém-se a resposta z, que é para onde x, no produto $\alpha\beta$, deve mapear. Caso exista mais de dois ciclos, o processo é repetido até se alcançar o ciclo mais à esquerda.

A conjugação de β por α, denotada por $\alpha.\beta$ é da forma $\alpha\beta\alpha^{-1}$. Tal operação resulta em uma permutação da mesma estrutura de β, mas com os elementos "renomeados" por α da seguinte maneira:

$$\alpha\beta\alpha^{-1} = (\alpha(\beta_1)\, \alpha(\beta_2)\, \ldots\, \alpha(\beta_k))$$

A.8.2 Formalisto algébrico aplicado a rearranjo de genomas

Para rearranjo de genomas, o conjunto base utilizado é $E_n = \{-1, +1, -2, +2, ..., -n, +n\}$, onde n é o número de genes. Para cada elemento $+i$, o complemento na fita oposta de DNA é chamado $-i$. Denomina-se γ a permutação que mapeia cada elemento de E no seu complementar.

$$\gamma = (-1\ +1)(-2\ +2)...(-n\ +n)$$

Para um ciclo α, o reverso dado por α^{-1} é obtido invertendo a ordem dos elementos. Assim, se $\alpha = (a\, b\, c)$ então $\alpha^{-1} = (c\, b\, a)$. O complementar reverso de α é, então, dado por $\gamma.(\alpha^{-1})$ ou $(\gamma.\alpha)^{-1}$.

Ciclos admissíveis são aqueles que representam algum trecho de uma fita genômica e, para isso, não podem conter elementos $-i$ e $+i$ ao mesmo tempo. Por exemplo, γ definido anteriormente não pertence ao conjunto dos ciclos admissíveis..

Um genoma π é o produto de duas fitas complementares reversas π_1 e $\gamma.(\pi_1^{-1})$. Os problemas de rearranjos aplicados a π são em geral definidos da seguinte maneira: dados dois genomas π, σ e uma classe de operações ρ, encontre o número mínimo de operações

da classe ρ que transformam π em σ. Esse número mínimo é chamado de distância para aquela classe de operações.

Algumas classes de operações são mostradas no trabalho como forma de ilustrar a utilização da notação algébrica. Nas formulações abaixo, os literais u, v, w e x representam elementos em uma mesma fita de π.

Reversão com sinal: $\rho = (u\ \gamma\pi v)(v\ \gamma\pi u)$

Reversão sem sinal: $\rho = (u\ v)(\gamma\pi v\ \gamma\pi u)$

Transposição sem sinal: $\rho = (u\ v\ w)(\gamma\pi w\ \gamma\pi v\ \gamma\pi u)$

Transposição com sinal: $\rho = (u\ v\ \gamma\pi w)(w\ \gamma\pi v\ \gamma\pi u)$

Troca de Blocos: $\rho = (u\ w)(\gamma\pi w\ \gamma\pi u)(vx)(\gamma\pi x\ \gamma\pi v)$

A.9 Transposition Distance based on the Algebraic Formalism [76]

O trabalho utiliza a abordagem algébrica de Meidanis e Dias (Seção A.8) e apresenta um algoritmo de aproximação para o problema da distância de transposição com razão 1.5, o que exemplifica a viabilidade da abordagem algébrica.

O formalismo algébrico [73] propõe um modo diferente de representar uma permutação e insere novos conceitos como conjugação e produto. Dessa forma, recomenda-se fortemente a leitura da Seção A.8 antes de iniciar esta seção.

Para uma permutação π, cujos elementos pertencem ao conjunto E, o elemento x é dito fixado quando $\pi(x) = x$. O suporte de π, $Supp(\pi)$, é o subconjunto de todos os elementos não fixados em π. Por exemplo, na permutação π = (0 3 1 5 4 2), com elementos pertencentes ao conjunto $E = \{0, 1, 2, 3, 4, 5\}$, os elementos 0 e 4 são fixados e $Supp(\pi) = \{1,2,3,5\}$.

Toda permutação π está associada a um conjunto E que contém os elementos de π. O conjunto E é importante para saber quais elementos existem na permutação π, pois alguns podem ser omitidos (elementos fixados são normalmente omitidos). Para facilitar a leitura, quando o conjunto E não for mencionado, deve-se entender que ele é composto pelos elementos $\{1 \ldots n\}$, onde n é o maior elemento que aparece em π.

A órbita de um elemento $x \in E$ em π é o conjunto $orb(x, \pi) = \{y | y = \pi^k(x)$ para algum inteiro $k\}$. No exemplo acima, a órbita do elemento 2 é $\{1,2,3,5\}$. O número de órbitas de π, com os elementos pertencendo ao conjunto E, será denominado $\circ(\pi, E)$

e o número de órbitas pares e ímpares de π será denominado $\circ_{even}(\pi, E)$ e $\circ_{odd}(\pi, E)$, respectivamente.

Uma órbita é dita trivial quando contém apenas um elemento e não trivial caso contrário. Um ciclo é uma permutação com no máximo uma órbita não trivial. Um k-ciclo, $k > 1$, é um ciclo cujo órbita não trivial possui tamanho k. No exemplo acima, a permutação π é um 4-ciclo que pode ser representado por (1 3 5 2). Um k-ciclo é dito ímpar se k é ímpar; caso contrário, é dito par.

Para uma permutação π, a decomposição em ciclos é um produto de órbitas disjuntas que representam π. Vale ressaltar que elementos fixados são omitidos nesta representação.

Uma transposição utilizando o formalismo algébrico é um 3-ciclo $\gamma = (u\ v\ w)$, com $u, v, w \in E$. A transposição é dita aplicável se u, v e w estão no mesmo ciclo na decomposição de π. Formalmente, diz-se que w segue v, $v \rightarrow_{u,\pi} w$, para o par (u, π) quando $v = \pi^{k_1}(u)$, $w = \pi^{k_2}(u)$ e $0 < k_1 < k_2 < |orb(u, \pi)|$.

Assim, o problema da distância de transposição consiste em encontrar um sequência de transposições $\gamma_1, \ldots, \gamma_t$ tal que $\gamma_t\gamma_{t-1} \ldots \gamma_1\pi = \sigma$, γ_i é aplicável a $\gamma_{i-1} \ldots \gamma_1\pi$, $1 \leq i \leq t$ e t é mínimo.

O produto $\sigma\pi^{-1}$ no formalismo algébrico produz órbitas correspondentes aos ciclos do grafo de ciclos proposto por Bafna e Pevzner [9]. Dessa forma, o trabalho propõe seguir, em linhas gerais, os passos de Bafna e Pevzner em seu algoritmo utilizando este produto.

Dados π e σ (quando houver referências a π e σ, assume-se que ambos estão associados ao mesmo conjunto E), uma transposição γ aplicável a π é um x-move quando $\circ(\sigma\pi^{-1}\gamma^{-1}) = \circ(\sigma\pi^{-1}) + x$. Em outras palavras, um x-move, com $x \in \{-2, 0, 2\}$, aumenta em x o número de ciclos no grafo de ciclos de Bafna e Pevzner [9].

Um x-move é válido se a variação de x ocorre no número de ciclos ímpares do grafo de ciclos de Bafna e Pevzner, ou seja, $\circ_{odd}(\sigma\pi^{-1}\gamma^{-1}) = \circ_{odd}(\sigma\pi^{-1}) + x$.

Dados π e σ sob E, dois ciclos α ($|\alpha| = k_1$, $k_1 \geq 2$) e β ($|\beta| = k_2$, $k_2 \geq 2$) em $\sigma\pi^{-1}$ são ligados na decomposição de ciclos de $\sigma\pi^{-1}$ se $x \rightarrow_{u,\pi} \pi\sigma^{-1}u \rightarrow_{u,\pi} \sigma\pi^{-1}x$, $u \in orb(\alpha, \sigma\pi^{-1})$ e $x \in orb(\beta, \sigma\pi^{-1})$.

Abaixo são apresentados alguns lemas que auxiliam na criação do algoritmo de aproximação na razão 1.5. Os lemas, bem como as provas (não apresentadas aqui, mas presentes no trabalho), foram enunciados utilizando o formalismo algébrico. Alguns desses resultados já eram conhecidos da literatura clássica de rearranjo de genomas, como pode ser visto na Seção A.1.

Lema A.2 *Dados π e σ e um ciclo não orientado $C = (x \ldots y \ldots z \ldots)$ em $\sigma\pi^{-1}$. Para todo $x, y, z \in C$, o 3-ciclo $\gamma = (x\ y\ z)$, com $y \rightarrow_{z,\pi} x$, transforma C em um ciclo não orientado em $\sigma\pi^{-1}\gamma^{-1}$.*

Lema A.3 *Dados π e σ, os ciclos $C = (x \ldots y \ldots)$ e $D = (x' \ldots y' \ldots)$ em $\sigma\pi^{-1}$ e uma*

transposição γ aplicável a π formada por três entre os quatro elementos x, y, x' e y', então:

1. *Se os ciclos C e D não são ligados, então γ cria um ciclo não orientado em $\sigma\pi^{-1}\gamma^{-1}$.*
2. *Se os ciclos C e D são ligados, então γ cria um ciclo orientado em $\sigma\pi^{-1}\gamma^{-1}$.*

Lema A.4 *Dados π e σ, se há um ciclo orientado em $\sigma\pi^{-1}$, então há uma transposição aplicável a π tal que γ é um 2-move válido ou existem as transposições γ_1, γ_2 e γ_3 aplicadas a π, $\gamma_1\pi$ e $\gamma_2\gamma_1\pi$ respectivamente que são um 0-move válido e dois 2-move válidos respectivamente.*

Lema A.5 *Sejam π e σ com $y \rightarrow_{x',\pi} y' \rightarrow_{x',\pi} x$, $D = (x \ldots y \ldots)$ e $D' = (x' \ldots y' \ldots)$ dois ciclos não orientados em $\sigma\pi^{-1}$ que não se cruzam, mas estão ligados. Além disso, supondo que $d(x, y)$ é ímpar. Então γ formado por três dentre os quatro elementos x, y, x' e y' tais que γ é aplicável a π, cria um ciclo fortemente orientado.*

Lema A.6 *Dados π e σ, se em $\sigma\pi^{-1}$ há um ciclos fortemente orientado fortemente cruzado com um ciclo não orientado, então existem dois 2-moves válidos consecutivos.*

Lema A.7 *Dados π e σ sem ciclos orientados em $\sigma\pi^{-1}$. Se há no mínimo um k-ciclo não orientado com $k > 3$ em $\sigma\pi^{-1}$, então existem as transposições γ_1, γ_2 e γ_3 aplicadas a π, $\gamma_1\pi$ e $\gamma_2\gamma_1\pi$ respectivamente que são um 0-move válido e dois 2-move válidos respectivamente.*

O pseudocódigo do algoritmo que utiliza os lemas acima é mostrado no trabalho, bem como a comparação com outros algoritmos existentes na literatura.

Bibliografia

[1] G. Achaz, E. Coissac, P. Netter, and E. P. C. Rocha. Associations between inverted repeats and the structural evolution of bacterial genomes. *Genetics*, 164(4):1279–1289, 2003.

[2] M. T. Alam, M. E. Merlo, E. Takano, and R. Breitling. Genome-based phylogenetic analysis of Streptomyces and its relatives. *Molecular Phylogenetics and Evolution*, 54:763–772, 2010.

[3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[4] S. Assefa, T. M. Keane, T. D. Otto, C. Newbold, and M. Berriman. ABACAS: algorithm-based automatic contiguation of assembled sequences. *Bioinformatics*, 25:1968–1969, 2009.

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

[6] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS'1993)*, pages 148–157, Palo Alto, USA, 1993. IEEE Computer Society Press.

[7] V. Bafna and P. A. Pevzner. Sorting by transpositions. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 614–623, San Francisco, USA, 1995.

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

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

[10] S. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J. P. Mesirov, and E. S. Lander. ARACHNE: a whole-genome shotgun assembler. *Genome Research*, 12:177–189, 2002.

[11] M. Benoit-Gagne and S. Hamel. A new and faster method of sorting by transpositions. In *Proceedings of the 18ht Annual Symposium on Combinatorial Pattern Matching (CPM'2007)*, pages 131–141, London, Canada, 2007.

[12] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. *Discrete Applied Mathematics*, 146:134–145, 2005.

[13] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th European Symposium on Algorithms (ESA'2002)*, pages 200–210, Rome, Italy, 2002.

[14] M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler, and W. Pirovano. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, 27:578–579, 2011.

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

[16] L. Bulteau, G. Fertin, and U. Rusu. Sorting by transpositions is difficult. In *Automata, Languages and Programming*, volume 6755 of *Lecture Notes in Computer Science*, pages 654–665. Springer Berlin / Heidelberg, 2011.

[17] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18:810–820, 2008.

[18] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. BLAST+: architecture and applications. *BMC Bioinformatics*, 10:421, 2009.

[19] C. Canchaya, M. J. Claesson, G. F. Fitzgerald, D. van Sinderen, and P. W. O'Toole. Diversity of the genus Lactobacillus revealed by comparative genomics of five species. *Microbiology*, 152:3185–3196, 2006.

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

[21] A. Caprara, G. Lancia, and S.-K. Ng. A column-generation based branch-and-bound algorithm for sorting by reversals. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 47:213–226, 1999.

[22] A. Caprara, G. Lancia, and S.-K. Ng. Fast practical solution of sorting by reversals. In *Proceedings of the 11th ACM-SIAM Annual Symposium on Discrete Algorithms (SODA'2000)*, pages 12–21, San Francisco, USA, 2000. ACM Press.

[23] P. Y. Chan, T. W. Lam, and S. M. Yiu. In *Proceedings of 4th Asia-Pacific Bioinformatics Conference*, Taipei,Taiwan.

[24] T. Chen and S. S. Skiena. Sorting with fixed-length reversals. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 71:269–295, 1996.

[25] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. Shields, I. Sudborough, and W. Voit. An (18/11)n upper bound for sorting by prefix reversals. *Theoretical Computer Science*, 410(36):3372–3390, 2009.

[26] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996.

[27] D. A. Christie. *Genome rearrangement problems*. PhD thesis, Glasgow University, 1998.

[28] A. E. Darling, B. Mau, F. R. Blattner, and N. T. Perna. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome Research*, 14:1394–1403, 2004.

[29] A. E. Darling, I. Miklós, and M. A. Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS Genetics*, 4(7):e1000128, 2008.

[30] A. Dayarian, T. P. Michael, and A. M. Sengupta. SOPRA: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics*, 11:345, 2010.

[31] M. Deloger, M. El Karoui, and M. A. Petit. A genomic distance based on MUM indicates discontinuity between most bacterial species and genera. *Journal of Bacteriology*, 191:91–99, 2009.

[32] U. Dias and Z. Dias. Constraint programming models for transposition distance problem. In *Proceedings of the 4th Brazilian Symposium on Bioinformatics (BSB'2009)*, volume 5676 of *Lecture Notes in Computer Science*, pages 13–23, Porto Alegre, Brasil, 2009. 10.1007/978-3-642-03223-3_2.

[33] U. Dias and Z. Dias. Extending Bafna-Pevzner algorithm. In *Proceedings of the 5th International Conference of the Brazilian Association for Bioinformatics and Computational Biology (X-Meeting'2009)*, pages 5–5, Angra dos Reis, Brasil, 2009.

[34] U. Dias and Z. Dias. (4,3)-sequences for two oriented cycles that do not lead to a valid 2-move. *Maintained by the authors*, Institute of Computing, University of Campinas, Brazil, 2010. http://www.ic.unicamp.br/∼udias/DD2010a/proof1/index.html.

[35] U. Dias and Z. Dias. Extending Bafna-Pevzner algorithm. In *Proceedings of the 1st International Symposium on Biocomputing (ISB'2010)*, pages 1–8, Calicut, India, 2010.

[36] U. Dias and Z. Dias. An improved 1.375-approximation algorithm for the transposition distance problem. In *Proceeding of the 1st ACM International Conference on Bioinformatics and Computational Biology (ACM-BCB'2010)*, pages 334–337, Niagara Falls, EUA, 2010. ACM.

[37] U. Dias, Z. Dias, and J. C. Setubal. Evaluation of genome distance measures using tree-derived distances. In *Proceedings of the 6th Brazilian Symposium on Bioinformatics (BSB'2011)*, pages 73 – 76, Brasília, Brasil, 2011.

[38] U. Dias, Z. Dias, and J. C. Setubal. A simulation tool for the study of symmetric inversions in bacterial genomes. In *Comparative Genomics*, volume 6398 of *Lecture Notes in Computer Science*, pages 240–251. Springer Berlin / Heidelberg, 2011.

[39] U. Dias, Z. Dias, and J. C. Setubal. Two new whole-genome distance measures. In *Proceedings of the 6th Brazilian Symposium on Bioinformatics (BSB'2011)*, pages 61 – 64, Brasília, Brasil, 2011.

[40] Z. Dias, U. Dias, and J. C. Setubal. Using inversion signatures to generate draft genome sequence scaffolds. In *Proceeding of the 2nd ACM International Conference on Bioinformatics and Computational Biology (BCB'2011)*, pages 39–48, Chicago, EUA, 2011.

[41] Z. Dias, U. Dias, J. C. Setubal, and L. S. Heath. Sorting genomes using almost-symmetric inversions. In *Proceedings of the 27th Symposium On Applied Computing (SAC'2012)*, pages 1–7, Riva del Garda, Italy, 2012.

[42] Z. Dias and J. Meidanis. Genome rearrangements distance by fusion, fission, and transposition is easy. In *Proceedings of the String Processing and Information Retrieval (SPIRE'2001)*, pages 250–253, Laguna de San Rafael, Chile, 2001. IEEE Computer Society.

[43] Z. Dias and C. Souza. Polynomial-sized ILP models for rearrangement distance problems. In *Poster Proceedings of the 3rd Brazilian Symposium on Bioinformatics (BSB'2007)*, pages 74–85, Angra dos Reis, Brasil, 2007.

[44] The ECLiPSe constraint programming system, 2009. http://www.eclipse-clp.org.

[45] J. A. Eisen, J. F. Heidelberg, O. White, and S. L. Salzberg. Evidence for symmetric chromosomal inversions around the replication origin in bacteria. *Genome Biology*, 1(6):research0011.1–0011.9, 2000.

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

[47] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, and J. Wastlund. Sorting a bridge hand. *Discrete Mathematics*, 241:289–300, 2001.

[48] J. Felsenstein. Phylip (phylogeny inference package) version 3.5c. *Distributed by the author*, Department of Genetics, University of Washington, Seattle, 2007.

[49] J. Feng and D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms*, 3(3):25, 2007.

[50] R. D. Fleischmann, M. D. Adams, O. White, R. A. Clayton, E. F. Kirkness, A. R. Kerlavage, C. J. Bult, J. F. Tomb, B. A. Dougherty, and J. M. Merrick. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science*, 269:496–512, 1995.

[51] V. J. Fortuna. Distâncias de transposição entre genomas. Master's thesis, Institute of Computing, University of Campinas, 2005.

[52] M. Galardini, E. G. Biondi, M. Bazzicalupo, and A. Mengoni. CONTIGuator: a bacterial genomes finishing tool for structural insights on draft genomes. *Source Code for Biology and Medicine*, 6(11), 2011.

[53] W. H. Gates and C. H. Papadimitriou. Bounds for sorting by prefix reversals. *Discrete Mathematics*, 27:47–57, 1979.

[54] J. Goris, K. T. Konstantinidis, J. A. Klappenbach, T. Coenye, P. Vandamme, and J. M. Tiedje. DNA-DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology*, 57:81–91, 2007.

[55] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. In *Combinatorial Pattern Matching, 6th Annual Symposium (CPM'1995)*, volume 937 of *Lecture Notes in Computer Science*, pages 162–176, Espoo, Finland, 1995.

[56] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.

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

[58] T. Hartman and R. Sharan. A simpler 1.5-approximation algorithm for sorting by transpositions. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'2003)*, pages 156–169, Morelia, Mexico, 2003.

[59] R. A. Hausen, L. Faria, C. M. H. Figueiredo, and L. A. B. Kowada. On the toric graph as a tool to handle the problem of sorting by transpositions. In *Proceedings of the 3rd Brazilian symposium on Bioinformatics (BSB'2008)*, pages 79–91, Santo André, Brazil, 2008.

[60] M. H. Heydari and I. H. Sudborough. On the diameter of the pancake network. *Journal of Algorithms*, 25:67–94, 1997.

[61] S. A. Hijum, A. L. Zomer, O. P. Kuipers, and J. Kok. Projector 2: contig mapping for efficient gap-closure of prokaryotic genome sequence assemblies. *Nucleic Acids Research*, 33:560–566, 2005.

[62] C. W. Hill and B. W. Harnish. Inversions between ribosomal RNA genes of Escherichia coli. *Proceedings of the National Academy of Sciences*, 78(11):7069–7072, 1981.

[63] M. I. Honda. Implementation of the algorithm of Hartman for the problem of sorting by transpositions. Master's thesis, Department of Computer Science, University of Brasilia, 2004.

[64] X. Huang, J. Wang, S. Aluru, S. P. Yang, and L. Hillier. PCAP: a whole-genome assembly program. *Genome Research*, 13:2164–2170, 2003.

[65] P. Husemann and J. Stoye. r2cat: synteny plots and comparative assembly. *Bioinformatics*, 26:570–571, 2010.

[66] D. H. Huson, K. Reinert, and E. W. Myers. The greedy path-merging algorithm for contig scaffolding. *Journal of the ACM*, 49:603–615, 2002.

[67] K. T. Konstantinidis and J. M. Tiedje. Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences*, 102:2567–2572, 2005.

[68] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[69] A. Labarre. New bounds and tractable instances for the transposition distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):380–394, 2006.

[70] B. Larget, D. L. Simon, J. B. Kadane, and D. Sweet. A bayesian analysis of metazoan mitochondrial genome arrangements. *Molecular Biology and Evolution*, 22(3):486–95, 2005.

[71] I. Maccallum, D. Przybylski, S. Gnerre, J. Burton, I. Shlyakhter, A. Gnirke, J. Malek, K. McKernan, S. Ranade, T. P. Shea, L. Williams, S. Young, C. Nusbaum, and D. B. Jaffe. ALLPATHS 2: small genomes assembled accurately and with high continuity from short paired reads. *Genome Biology*, 10:R103, 2009.

[72] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[73] J. Meidanis and Z. Dias. An alternative algebraic formalism for genome rearrangements. In *Comparative Genomics: Empirical and Analyitical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families*, pages 213–223. Kluwer Academic Publishers, 2000.

[74] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Transposition distance between a permutation and its reverse. In *Proceedings of the 4th South American Workshop on String Processing (WSP'1997)*, pages 70–79, Valparaiso, Chile, 1997. Carleton University Press.

[75] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. Technical Report IC-00-23, Institute of Computing - University of Campinas, 2000.

[76] C. V. G. Mira, Z. Dias, H. P. Santos, G. A. Pinto, and M. E. Walter. Transposition distance based on the algebraic formalism. In *Proceedings of the 3rd Brazilian symposium on Bioinformatics (BSB'2008)*, pages 115–126, Santo André, Brazil, 2008.

[77] L. M. Moreira, N. F. Almeida, N. Potnis, L. A. Digiampietri, S. S. Adi, J. C. Bortolossi, A. C. da Silva, A. M. da Silva, F. E. de Moraes, J. C. de Oliveira, R. F. de Souza, A. P. Facincani, A. L. Ferraz, M. I. Ferro, L. R. Furlan, D. F. Gimenez, J. B. Jones, E. W. Kitajima, M. L. Laia, R. P. Leite, M. Y. Nishiyama, J. Rodrigues Neto, L. A. Nociti, D. J. Norman, E. H. Ostroski, H. A. Pereira, B. J. Staskawicz, R. I. Tezza, J. A. Ferro, B. A. Vinatzer, and J. C. Setubal. Novel insights into the genomic basis of citrus canker based on the genome sequences of two strains of Xanthomonas fuscans subsp. aurantifolii. *BMC Genomics*, 11:238, 2010.

[78] B. M. Moret, L. S. Wang, T. Warnow, and S. K. Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17 Suppl 1:S165–S173, 2001.

[79] A. Munoz, C. Zheng, Q. Zhu, V. A. Albert, S. Rounsley, and D. Sankoff. Scaffold filling, contig fusion and comparative gene order inference. *BMC Bioinformatics*, 11:304, 2010.

[80] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, K. A. Remington, E. L. Anson, R. A. Bolanos, H. H. Chou, C. M. Jordan, A. L. Halpern, S. Lonardi, E. M. Beasley, R. C. Brandon, L. Chen, P. J. Dunn, Z. Lai, Y. Liang, D. R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, and J. C. Venter. A whole-genome assembly of Drosophila. *Science*, 287:2196–2204, 2000.

[81] N. Nagarajan, T. D. Read, and M. Pop. Scaffolding and validation of bacterial genome assemblies using optical restriction maps. *Bioinformatics*, 24:1229–1235, 2008.

[82] E. Ohlebusch, M. I. Abouelhoda, K. Hockel, and J. Stallkamp. The median problem for the reversal distance in circular bacterial genomes. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM'2005)*, pages 116–127, Jeju Island, Korea, 2005.

[83] M. Pop, D. S. Kosack, and S. L. Salzberg. Hierarchical scaffolding with Bambus. *Genome Research*, 14:149–159, 2004.

[84] D. C. Richter, S. C. Schuster, and D. H. Huson. OSLay: optimal syntenic layout of unfinished assemblies. *Bioinformatics*, 23:1573–1579, 2007.

[85] A. I. Rissman, B. Mau, B. S. Biehl, A. E. Darling, J. D. Glasner, and N. T. Perna. Reordering contigs of draft genomes using the Mauve aligner. *Bioinformatics*, 25:2071–2073, 2009.

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

[87] D Sankoff and M Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–70, 1998.

[88] C. Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of Mathematics*, 13:179–191, 1961.

[89] J. C. Setubal and J. Meidanis. *Introduction to Computional Molecular Biology*. PWS Publishing Company, 1997.

[90] J. C. Setubal, P. Santos, B. S. Goldman, H. Ertesvåg, G. Espin, L. M. Rubio, S. Valla, N. F. Almeida, D. Balasubramanian, L. Cromes, L. Curatti, Z. Du, E. Godsy, B. Goodner, K. Hellner-Burris, J. A. Hernandez, K. Houmiel, J. Imperial, C. Kennedy, T. J. Larson, P. Latreille, L. S. Ligon, J. Lu, M. Maerk, N. M. Miller, S. Norton, I. P. O'Carroll, I. Paulsen, E. C. Raulfs, R. Roemer, J. Rosser, D. Segura, S. Slater, S. L. Stricklin, D. J. Studholme, J. Sun, C. J. Viana, E. Wallin, B. Wang, C. Wheeler, H. Zhu, D. R. Dean, R. Dixon, and D. Wood. Genome sequence of azotobacter vinelandii, an obligate aerobe specialized to support diverse anaerobic metabolic processes. *Journal of Bacteriology*, 191(14):4534–45, 2009.

[91] B. Snel, P. Bork, and M. A. Huynen. Genome phylogeny based on gene content. *Nature Genetics*, 21:108–110, 1999.

[92] E. Tannier and M.-F. Sagot. Sorting by reversals in subquadratic time. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM'2004)*, pages 1–13, Istambul, Turkey, 2004.

[93] A. Valouev, Y. Zhang, D. C. Schwartz, and M. S. Waterman. Refinement of optical map assemblies. *Bioinformatics*, 22:1217–1224, 2006.

[94] D. M. Vienne, T. Giraud, and O. C. Martin. A congruence index for testing topological similarity between trees. *Bioinformatics*, 23:3119–3124, 2007.

[95] M. E. M. T. Walter, L. R. A. F. Curado, and A. G. Oliveira. Working on the problem of sorting by transpositions on genome rearrangements. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM'2003)*, volume 2676 of *Lecture Notes in Computer Science*, pages 372–383. Springer, Berlin, 2003.

[96] M. E. M. T. Walter, Z. Dias, and J. Meidanis. A new approach for approximating the transposition distance. In *Proceedings of the String Processing and Information Retrieval (SPIRE'2000)*, pages 199–208, Coruña, Spain, 2000.

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

[98] R. L. Warren, D. Varabei, D. Platt, X. Huang, D. Messina, S. P. Yang, J. W. Kronstad, M. Krzywinski, W. C. Warren, J. W. Wallis, L. W. Hillier, A. T. Chinwalla, J. E. Schein, A. S. Siddiqui, M. A. Marra, R. K. Wilson, and S. J. Jones. Physical map-assisted whole-genome shotgun sequence assemblies. *Genome Research*, 16:768–775, 2006.

[99] K. P. Williams, J. J. Gillespie, B. W. Sobral, E. K. Nordberg, E. E. Snyder, J. M. Shallom, and A. W. Dickerman. Phylogeny of gammaproteobacteria. *Journal of Bacteriology*, 192:2305–2314, 2010.

[100] C. R. Woese. Bacterial evolution. *Microbiological Reviews*, 51:221–271, 1987.

[101] D. Wu, P. Hugenholtz, K. Mavromatis, R. Pukall, E. Dalin, N. N. Ivanova, V. Kunin, L. Goodwin, M. Wu, B. J. Tindall, S. D. Hooper, A. Pati, A. Lykidis, S. Spring, I. J. Anderson, P. D'haeseleer, A. Zemla, M. Singer, A. Lapidus, M. Nolan, A. Copeland, C. Han, F. Chen, J. F. Cheng, S. Lucas, C. Kerfeld, E. Lang, S. Gronow, P. Chain, D. Bruce, E. M. Rubin, N. C. Kyrpides, H. P. Klenk, and J. A. Eisen. A phylogeny-driven genomic encyclopaedia of bacteria and archaea. *Nature*, 462:1056–1060, 2009.

[102] F. Zhao, H. Hou, Q. Bao, and J. Wu. Pga4genomics for comparative genome assembly based on genetic algorithm optimization. *Genomics*, 94:284–286, 2009.