Enumeração de Soluções de Distância de Rearranjo e Alinhamento de Sequências utilizando Eventos de Rearranjo

Christian Baudet

Resumo

O estudo de eventos de rearranjo é de grande importância para a obtenção de informações sobre os mecanismos de evolução dos genomas dos organismos. Dois tópicos dentro da área de rearranjo de genomas serão abordados neste projeto de doutorado: *Enumeração de Soluções de Distância de Rerranjo* e *Alinhamento de Sequências utilizando Eventos de Rearranjo*. Sendo assim, apresentaremos neste texto os conceitos envolvidos com estes dois tópicos e as atividades planejadas para a execução do projeto de doutorado.

1 Introdução

Entender o mecanismo da evolução das espécies é um dos grandes objetivos da ciência. O estudo deste assunto envolve inúmeros ramos de pesquisa. A análise pode considerar desde aspectos sociais e comportamentais, para entender como a interação entre organismos influencia em suas evoluções, até aspectos moleculares, para compreensão de como as mudanças no DNA, nas proteínas e nas vias metabólicas trazem vantagens para que um indivíduo prevaleça em relação a outro.

Entre as diversas linhas de pesquisa que visam compreender o mecanismo de evolução molecular, podemos citar *Rearranjo de Genomas*. Este ramo de pesquisa tem o objetivo de buscar a compreensão da evolução dos indivíduos através de eventos de mutação que ocorrem em seus genomas.

Enquanto os algoritmos tradicionais de alinhamento de sequências de DNA (proteínas) realizam a comparação nucleotídeo a nucleotídeo (aminoácido a aminoácido) e detectam apenas eventos de inserção, remoção e substituição em uma única base (aminoácido), rearranjo de genomas considera mutações envolvendo grandes regiões do genoma. Estas regiões, normalmente genes, são analisadas em relação à ordem em que elas se encontram nos diferentes organismos. Dessa maneira, a análise busca obter uma estimativa da distância evolucionária entre as espécies estudadas.

O trabalho proposto para o desenvolvimento da tese de Doutorado se concentrará em tópicos na área de *Rearranjo de Genomas*. Trabalharemos com o problema de *Enumeração de Soluções para o Problema de Distância de Reversão* e abordaremos também o problema de *Alinhamento de Sequências Utilizando Eventos de Rearranjo*.

Sendo assim, a apresentação da proposta seguirá a seguinte estrutura: a Seção 2 apresentará alguns conceitos básicos da área de Rearranjo de Genomas e a Seção 3 apresentará os eventos de rearranjo e os resultados mais importantes obtidos para cada um deles. A Seção 4 introduzirá o problema de enumeração de todas as soluções ótimas possíveis que ordenam uma permutação orientada somente com reversões. Em seguida, na Seção 5 abordaremos o problema de alinhamento de sequências com reversões. A Seção 6 fará uma breve apresentação do laboratório onde a pesquisa será realizada durante o estágio no exterior. Finalmente, a Seção 7 detalhará as atividades de pesquisas que serão desenvolvidas ao longo do Doutorado e a Seção 8 apresentará o cronograma de atividades.

2 Rearranjo de Genomas

Os pioneiros da área de rearranjo de genomas foram Dobzhansky e Sturtevant que, em 1938, publicaram um estudo com uma árvore evolucionária que mostrava um cenário de 17 reversões para as espécies *Drosophila pseudoobscura* e *Drosophila miranda* [32]. Com o passar dos anos, outros estudos mostraram que eventos de rearranjos são comuns na evolução molecular de plantas, mamíferos, vírus e bactérias [3,19,33,40,41,43–45,50,61,62].

No final dos anos 80, Palmer *et al.* realizaram a comparação entre os genomas mitocondriais das espécies *Brassica oleracea* e *Brassica campestris* e descobriram que elas são muito similares (muitos genes são entre 99 e 99,9% idênticos) [58]. Em outro exemplo, O'Brien traz em seu livro "Genetics Maps: Locus Maps of Complex Genomes" um estudo que mostra que as maiores diferenças entre os genomas das bactérias *Escherichia coli* e *Salmonella typhimurium* são as ordens de seus genes em seus cromossomos [25]. Com a evolução das técnicas de sequenciamento, que permitiu que genomas completos fossem obtidos, a área de rearranjo de genomas passou a ter uma quantidade imensa de material para estudo.

Os eventos de rearranjo mais comumente estudados são *reversões*, *transposições*, *translocações*, *fusões* e *fissões*.

Uma *reversão* ocorre quando um segmento do genoma é destacado e ligado no mesmo lugar com sua ordem invertida em relação a sua posição original. Este tipo de evento tem papel de extrema importância na diversidade das plantas e bactérias [54].

A *transposição* é um evento que envolve a mudança de posição de dois blocos de genes adjacentes no genoma. Existe uma generalização do evento de transposição chamada *block-interchange* [22]. Neste evento, não existe a restrição de que os segmentos trocados sejam adjacentes.

Ao contrário da reversão e da transposição, a *translocação* é um evento que envolve dois cromossomos. Suponha que dois cromossomos X e Y são divididos em duas partes cada, formando segmentos (X_1, X_2) e (Y_1, Y_2) (nenhum deles vazio). A translocação produz dois novos cromossomos combinando o prefixo ou sufixo de um cromossomo com o prefixo ou sufixo do outro. Assim, uma translocação pode produzir os cromossomos (X_1, Y_2) e (Y_1, X_2) ou (Y_1, X_1^R) e (Y_2^R, X_2), onde X^R denota o complemento reverso de X. As translocações, junto com as reversões, são os eventos mais comuns na evolução dos mamíferos [54].

Fusão e *fissão* são outros eventos observados em mamíferos. A fusão age de forma a unir dois cromossomos para formar um único novo cromossomo. A fissão, por outro lado, realiza o papel inverso ao quebrar um cromossomo em dois novos cromossomos.

No estudo de rearranjo de genomas, cada gene é representado por um identificador. Seja $\mathcal{E}$ um conjunto de identificadores. Um *cromossomo* é definido como uma permutação de identificadores do conjunto $\mathcal{E}$ e um *genoma* é uma coleção de cromossomos. Em geral, assumimos que cada gene aparece uma única vez em um cromossomo.

Normalmente, utilizamos como identificadores números inteiros positivos. Se, adicionalmente, existir informação sobre a orientação dos genes, sinais positivos e negativos são associados aos identificadores para formar uma permutação que é denominada *permutação orientada*.

Quando comparamos dois cromossomos, assumimos que um deles é a *permutação identidade* $i = (1, 2, \ldots, n)$ e a outra é a permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. O objetivo aqui é obter o número mínimo de eventos necessários (ou a própria sequência de eventos necessários) para que a permutação π seja transformada na permutação identidade.

3 Eventos de Rearranjo

Nesta seção detalharemos os eventos de rearranjo e listaremos, para cada um deles, uma série de resultados existentes na literatura. A Seção 3.1 tratará do evento de reversão e a Seção 3.2 abordará o evento de transposição. Translocações serão discutidas na Seção 3.3 e os estudos de eventos de rearranjo combinados serão apresentados na Seção 3.4.

3.1 Reversões

Eventos de reversão foram observados pela primeira vez por Dobzhansky e Sturtevant [31] em 1936. Neste estudo, os autores apontaram a ocorrência de reversões no terceiro cromossomo de variedades selvagens da espécie *Drosophila pseudoobscura*.

Algebricamente, uma reversão é definida como uma operação $\rho(i, j)$ sobre um intervalo $[i, j]$ de uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ que realiza a seguinte transformação:

$$\begin{aligned}\rho(i,j)(\pi_1, \ldots, \pi_{i-1}, \underline{\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j}, \pi_{j+1}, \ldots, \pi_n) \\ \rightarrow (\pi_1, \ldots, \pi_{i-1}, \underline{\pi_j, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_i}, \pi_{j+1}, \ldots, \pi_n)\end{aligned}$$

Caso π seja uma permutação orientada, a reversão $\rho(i, j)$ também muda os sinais dos elementos $\pi_i, \ldots, \pi_j$. A Figura 1a mostra um exemplo do evento de reversão.

Uma das classes de problemas na área de rearranjo de genomas utiliza a reversão como evento base das transformações. Assim, a *distância de reversão*, denotada por $d(\pi, \sigma)$, é definida como sendo o número mínimo de reversões necessárias para transformar a permutação π na permutação σ. O problema de descobrir o número mínimo de reversões necessárias para transformar π na permutação identidade, denotado por $d(\pi)$, é conhecido como *Ordenação por Reversões*.

Kececioglu e Sankoff apresentaram o primeiro algoritmo de aproximação para o problema de ordenação de permutações não orientadas [51]. Este algoritmo utiliza uma estratégia gulosa que atinge uma aproximação de fator 2 e executa em tempo $O(n^2)$ e espaço $O(n)$ para permutações de n elementos.

Posteriormente, Bafna e Pevzner construíram um algoritmo de aproximação com fator 1.75, e tempo de execução $O(n^2)$, usando uma estrutura denominada *Grafo de Break-*

points [5]. Para o caso de permutações orientadas, o fator de aproximação cai para 1.5 e o tempo de execução para $O(n^{1.5})$.

Em 1998, Christie aprimorou o fator de aproximação para 1.5 para o caso de permutações não orientadas, permanecendo por muito tempo como o melhor resultado conhecido [23].

Berman, Hannenhalli e Karpinski apresentaram em 2002 um algoritmo de aproximação para permutações não orientadas [12]. Este algoritmo possui fator de aproximação 1.375.

Caprara provou que o problema de ordenação por reversões para permutações não orientadas é NP-Completo [17]. Contudo, Hannenhalli e Pevzner provaram que a versão do problema que considera permutações orientadas pode ser resolvida em tempo polinomial, apresentando um algoritmo de complexidade $O(n^4)$ [42]. Em 1996, Berman e Hannenhalli apresentam um algoritmo com complexida $O(n^2\alpha(n))$, onde $\alpha(n)$ é o inverso da função de Ackerman, uma função praticamente constante [11]. Em 2000, Kaplan, Shamir e Tarjan construíram um algoritmo mais rápido e mais simples que resolve o problema em tempo $O(n^2)$ [49]. Neste mesmo ano, Meidanis, Walter e Dias provaram que o problema de ordenação por reversões em permutações orientadas circulares pode ser resolvido em tempo $O(n^2)$ [56].

Bergeron apresentou uma visão elementar da teoria de Hannenhalli e Pevzner [42] e apresentou um algoritmo simples com complexidade $O(n^2)$ e que não necessita da construção do grafo de *breakpoints* [8].

Para o problema de apenas calcular a distância de reversão de duas permutações orientadas, Bader, Moret e Yan apresentaram em 2001 um algoritmo que resolve o problema em tempo linear [2].

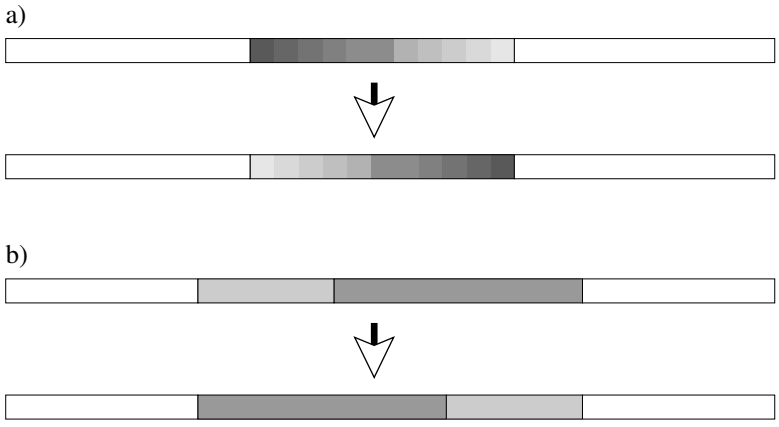


Figura 1: a) No evento de reversão, um segmento de DNA é destacado do genoma e reinserido com a sua ordem invertida. b) No evento de transposição, dois segmentos consecutivos de DNA são destacados do genoma e reinseridos com as suas posições trocadas.

3.2 Transposições

Como mencionado anteriormente, transposições são eventos que trocam a posição de dois blocos adjacentes no mesmo cromossomo. É importante notar que durante o evento, nenhum dos blocos tem sua orientação alterada. A Figura 1b mostra um exemplo do evento de transposição.

Algebricamente uma transposição é definida como uma operação $\rho(i, j, k)$ sobre uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ que realiza a seguinte transformação:

$$\begin{aligned}\rho(i,j,k)(\pi_1,\ldots,\pi_{i-1},\underline{\pi_i,\ldots,\pi_{j-1}},\underline{\pi_j,\ldots,\pi_{k-1}},\pi_k,\ldots,\pi_n)\\ \rightarrow(\pi_1,\ldots,\pi_{i-1},\underline{\pi_j,\ldots,\pi_{k-1}},\underline{\pi_i,\ldots,\pi_{j-1}},\pi_k,\ldots,\pi_n),\end{aligned}$$

onde $1 \leq i < j \leq n+1$, $1 \leq k \leq n+1$ e $k \notin [i, j]$.

Transposições foram primeiramente estudadas por Bafna e Pevzner. Eles determinaram os limites inferiores para a distância de transposição entre cromossomos e propuseram o primeiro algoritmo de aproximação com fator 1.5 e complexidade $O(n^2)$ [4, 6].

Walter, Dias e Meidanis desenvolveram um algoritmo de implementação simples e que executa em $O(n^2)$, porém, seu fator de aproximação é de 2.25 [69].

Christie também produziu um algoritmo com fator de aproximação 1.5 e, apesar do algoritmo possuir complexidade de tempo $O(n^4)$, ele é mais simples de entender [24].

O problema de ordenação por transposições permanece em aberto. Nenhum algoritmo polinomial exato é conhecido e nenhuma prova de que ele seja NP-completo foi produzida até a presente data.

Se restrições são impostas ao problema, algoritmos exatos eficientes ou algoritmos de aproximação com melhores fatores podem ser produzidos. Por exemplo, Jerrum apresentou um algoritmo polinomial para ordenação por transposições quando apenas pares de genes adjacentes podem ser trocados [48].

Em outro exemplo, Heath e Vergara consideraram a versão que limita o tamanho dos blocos que estão sendo trocados [46]. O problema pode ser resolvido em tempo $O(n^2)$ se um dos blocos possui um único elemento e o outro bloco não tem limite de tamanho. Para o caso em que o tamanho total dos dois blocos adjacentes é limitado por uma função proporcional a n, os autores reduziram o problema geral para o problema restrito e, assim, mostraram que este é, ao menos, tão difícil quanto o caso geral. Limitando a 3 o tamanho total dos blocos que estão sendo trocados, Heath e Vergara produziram um algoritmo de aproximação com fator de performance 1.33 [47].

Em 1996, Christie introduziu uma generalização do problema de ordenação por transposição [22]. O evento de *block-interchange* generaliza a transposição no aspecto em que ele não exige que os blocos, que estão sendo trocados, sejam adjacentes. Christie mostrou neste trabalho que a ordenação por *block-interchange* pode ser resolvida em tempo polinomial com um algoritmo de complexidade $O(n^2)$.

Resultados mais recentes conseguiram aprimorar o fator de aproximação do problema de ordenação por transposições. Em 2005, Elias e Hartman conduziram uma análise em relação ao limite superior do diâmetro de transposição (valor máximo que a distância de transposição pode alcançar) e como resultado produziram uma algoritmo de aproximação com fator 1.375 [34].

3.3 Translocações

O evento de translocação envolve o rearranjo de blocos em dois cromossomos distintos. A Figura 2 demonstra um exemplo do evento de translocação.

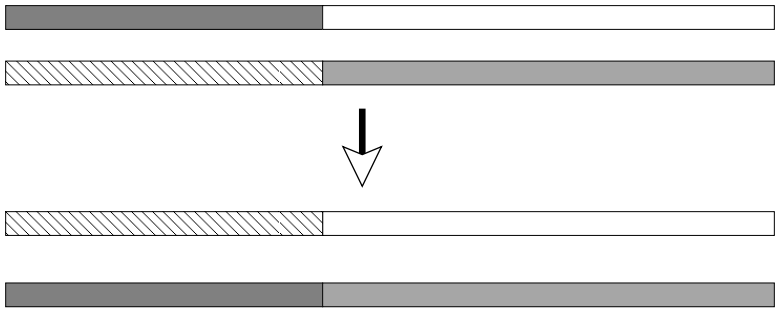


Figura 2: O evento de translocação envolve dois cromossomos diferentes. Cada um dos cromossomos é dividido em dois segmentos não vazios e os prefixos/sufixos de cada cromossomo combinam-se entre si, formando dois novos cromossomos.

Dados duas permutações $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ e $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, uma translocação prefixo-prefixo $\rho_{pp}(\pi, \sigma, i, j)$ produz duas permutações $(\pi_1, \pi_2, \ldots, \pi_{i-1}, \sigma_j, \sigma_{j+1}, \ldots, \sigma_n)$ e $(\sigma_1, \sigma_2, \ldots, \sigma_{j-1}, \pi_i, \sigma_{i+1}, \ldots, \sigma_m)$, onde $1 < i \leq m$ e $1 < j \leq n$.

Uma translocação prefixo-sufixo $\rho_{ps}(\pi, \sigma, i, j)$ aplicada sobre π e σ produz duas permutações $(\sigma_n, \sigma_{n-1}, \ldots, \sigma_j, \pi_i, \ldots, \pi_m)$ e $(\sigma_1, \ldots, \sigma_{j-1}, \pi_{i-1}, \pi_{i-2}, \ldots, \pi_1)$, onde $1 < i \leq m$ e $1 < j \leq n$. Se π e σ são permutações orientadas, então uma translocação prefixo-sufixo $\rho_{ps}(\pi, \sigma, i, j)$ produz duas permutações orientadas $(-\sigma_n, -\sigma_{n-1}, \ldots, -\sigma_j, \pi_i, \ldots, \pi_m)$ e $(\sigma_1, \ldots, \sigma_{j-1}, -\pi_{i-1}, -\pi_{i-2}, \ldots, -\pi_1)$.

Kececioglu e Ravi foram os primeiros a estudar o problema do cálculo da distância de translocação do ponto de vista computacional [50].

O estudo de Kececioglu e Ravi abordou a versão não orientada do problema. Eles obtiveram um algoritmo com fator de aproximação 2 e complexidade de tempo $O(n^2)$, onde n é o número total de genes distintos no genoma. Além disso, eles mostraram que quando é imposta a restrição de que todos os segmentos trocados por uma translocação têm tamanhos idênticos, existe um algoritmo que resolve o problema em tempo linear.

Hannenhalli resolveu a versão orientada do problema de ordenação por translocação, produzindo um algoritmo que executa em tempo $O(n^3)$ [39]. Em 2002, Zhu e Ma produziram um algoritmo com complexidade $O(n^2 \log n)$ [73]. Em 2004, Li *et al.* publicaram um algoritmo que realiza o cálculo da distância de translocação de permutações orientadas em tempo linear [53]. Recentemente, Wang *et al.* desenvolveram um algoritmo que resolve o problema de ordenação por translocação em tempo $O(n^2)$ [70].

O problema de ordenação de permutações não orientadas utilizando translocações continua em aberto.

3.4 Combinações de Eventos

Combinações de diferentes eventos de rearranjo também são consideradas por muitos pesquisadores. Por exemplo, Bafna e Pevzner sugeriram o problema de ordenação de per-

mutações considerando reversões e transposições. Assim, calcular a distância de reversão e transposição de um par de permutações é obter o número mínimo de eventos de reversão e transposição que transforma uma permutação em outra. De maneira similar, a ordenação por reversões e transposições é o problema de se encontrar a menor série de reversões e transposições que transforma uma permutação na identidade.

Walter, Meidanis e Dias apresentaram um algoritmo de aproximação com fator 3 para calcular a distância de reversão e transposição de permutações não orientadas e um algoritmo de aproximação com fator 2 para as permutações orientadas. Ambos os algoritmos possuem complexidade $O(n^2)$ [68].

Gu *et al.* apresentaram uma heurística gulosa para o problema de ordenar permutações orientadas por reversões, transposições e transposições invertidas [37]. Uma transposição invertida é um evento de rearranjo no qual, além da transposição, um dos fragmentos transpostos também sofre uma reversão. Posteriormente, eles apresentaram um algoritmo de aproximação com fator 2 e complexidade $O(n^2)$ [38].

Lin e Xue apresentaram um algoritmo unificado de aproximação com complexidade $O(n^2)$ e fator de aproximação 2 para a ordenação de permutações orientadas por reversões e transposições e para ordenação de permutações orientadas por reversões, transposições e transposições invertidas [55].

Wang e Warnow desenvolveram uma técnica chamada `IEBP` (Inverso do número esperado de *breakpoints*) para estimar o que eles chamam de *verdadeira distância evolucionária* entre dois genomas (orientados, não orientados, circulares ou lineares) [72]. A *verdadeira distância evolucionária* é, segundo os autores, o número mínimo de reversões, transposições e transposições invertidas necessárias para transformar um genoma em outro. Wang refinou a técnica e apresentou um método mais preciso, para genomas orientadas, denominado `Exact-IEBP` [71].

Alguns cientistas observaram na prática que transposições, em geral, ocorrem com a metade da frequência de ocorrência de reversões [13]. Isso motivou a análise de problemas onde os eventos recebem pesos diferentes.

Com base no algoritmo exato de Hannenhalli e Pevzner para ordenação de permutações orientadas por reversões [42], Eriksen apresentou um esquema de aproximação de tempo polinomial (PTAS) para o problema, com a restrição de que as permutações sejam orientadas e circulares [35].

Dias e Meidanis estudaram o problema de ordenação por fissão, fusão e transposições onde as transposições possuíam peso duas vezes maior do que os pesos dos eventos de fusão e fissão [26]. O estudo resultou em um algoritmo polinomial com complexidade de tempo $O(n^2)$ para encontrar a série de fusões, fissões e transposições de peso mínimo quando os genomas são representados como sequências de genes circulares e orientadas. Este algoritmo foi o primeiro algoritmo polinomial envolvendo transposições.

Kececioglu e Ravi consideraram o estudo do problema de ordenação de genomas por reversões e translocações. Eles apresentaram um algoritmo de aproximação de fator 2 para a ordenação de permutações não orientadas por reversões e translocações. Eles também apresentaram um algoritmo com fator 1.5 para a versão orientada do problema. Ambos os algoritmos possuem complexidade $O(n^2)$, onde n é o número de genes distintos no genoma [50].

Quando os números de cromossomos de dois genomas são diferentes, inevitavelmente é

necessária a utilização dos eventos de fusão ou fissão. Esse fato abre espaço para a definição de um problema ainda mais amplo: a ordenação de genomas por reversões, translocações, fusões e fissões, que consiste em encontrar a menor série destes eventos para transformar um genoma em outro.

Hannenhalli e Pevzner consideraram a versão orientada do problema e produziram um algoritmo polinomial exato com complexidade $O(n^4)$ [43].

4 Enumeração de Soluções para o Problema de Distância de Reversões

Os algoritmos mencionados anteriormente produzem, como saída, uma solução ótima que transforma uma permutação em outra. No entanto, o espaço de soluções ótimas é frequentemente imenso e não existe garantia de que a solução produzida seja aquela que realmente aconteceu ao longo da evolução das duas espécies comparadas.

Seja $d(\pi)$ a distância de reversão da permutação π em relação à permutação identidade. Uma sequência de reversões $s = \rho_1\rho_2 \dots \rho_i$ é chamada de *optimal i-sequence* quando $d(\pi \cdot \rho_1\rho_2 \dots \rho_i) = d(\pi) - i$. Note que se $i = d(\pi)$, então s é uma solução ótima. Baseado neste conceito, Siepel propôs em 2003 um algoritmo eficiente para enumerar todas as soluções ótimas possíveis para a ordenação de uma permutação orientada [66]. O algoritmo proposto permite que o conjunto de todas as *optimal 1-sequences* de uma permutação seja calculado em tempo $O(n^3)$.

O conjunto de todas as *optimal i-sequences* pode ser calculado a partir do conjunto de $(i-1)$-*sequences* através da iteração do mesmo algoritmo. A utilização desta solução resulta em um algoritmo capaz de listar todo o conjunto de soluções ótimas em tempo $O(n^{2n+3})$.

A enumeração de todo o conjunto de soluções é um resultado tão útil quanto listar uma única solução. Como o espaço de solução pode ser muito grande, a simples enumeração das soluções sem qualquer classificação não facilita o estudo comparativo dos dois genomas.

O trabalho de Bergeron *et al.* [9] apresenta uma forma de classificar o conjunto de soluções ótimas. Através da utilização do conceito de *traces*, as soluções ótimas podem ser agrupadas em classes de equivalência e uma solução representativa pode ser escolhida para representar a classe.

Se uma sequência de reversões for identificada como uma palavra no alfabeto $\mathcal{A}$ de reversões, Bergeron *et al.* definem uma relação de equivalência sobre estas palavras: se ρ e θ são reversões e não se sobrepõem, então as palavras $\rho\theta$ e $\theta\rho$ são equivalentes. Dizemos que, nesse caso, ρ e θ comutam. Baseada nesta relação, qualquer palavra contendo $\rho\theta$ como sub-palavra é equivalente a mesma palavra que substitui $\rho\theta$ por $\theta\rho$.

Por exemplo, para a permutação $(4, -3, -1, 2)$, podemos considerar como uma solução a sequência de reversões $\{1,3,4\}\{2,4\}\{2,3\}\{3\}$. Neste caso, a reversão $\{3\}$ comuta com todas as outras reversões e nenhuma das outras reversões comutam entre si. Assim, podemos dizer que as sequências $\{1,3,4\}\{2,4\}\{3\}\{2,3\}$, $\{1,3,4\}\{3\}\{2,4\}\{2,3\}$ e $\{3\}\{1,3,4\}\{2,4\}\{2,3\}$ são soluções equivalentes pela comutação de $\{3\}$ com todas as outras reversões.

Se considerarmos a sequência de reversões $\{1\}\{1,2\}\{4\}\{1,2,3,4\}$, temos que $\{4\}$ e $\{1,2\}$ comutam e, por isso, $\{1\}\{1,2\}\{4\}\{1,2,3,4\}$ é equivalente a $\{1\}\{4\}\{1,2\}\{1,2,3,4\}$.

Na realidade, neste exemplo, todas as reversões comutam entre si e, por isso, todas as permutações destas quatro reversões também são soluções.

Uma classe de equivalência de sequências ótimas de reversão sobre a relação de equivalência descrita acima é denominada *trace*. Bergeron *et al.* propõem que para uma permutação orientada π, o conjunto de todas as reversões ótimas é uma união de *traces*.

Como consequência, temos que se o conjunto de soluções que devem ser enumeradas é muito grande, o conjunto de *traces* pode ser um resultado mais relevante para o problema de ordenação por reversões.

Um elemento s de um *trace* T está em sua forma normal se ele pode ser decomposto em sub-palavras $s = u_1|\ldots|u_m$ tais que:

- todo par de elementos de uma sub-palavra u_i comutam;
- para todo elemento ρ de uma sub-palavra u_i $(i > 1)$, existe pelo menos um elemento θ da palavra u_{i-1} tal que ρ e θ não comutam;
- toda sub-palavra u_i é uma palavra crescente não vazia com relação à ordem lexicográfica induzida por $\mathcal{A}$.

Segundo um teorema de Cartier e Foata [20], para qualquer *trace* existe uma única palavra que está na forma normal. Assim, é possível representar os *traces* através das suas representações na forma normal. No caso da permutação $(4, -3, -1, 2)$, por exemplo, temos dois *traces* de soluções ótimas: $\{1\}\{1,2\}\{1,2,3,4\}\{4\}$ e $\{1,3,4\}\{3\}|\{2,4\}|\{2,3\}$. Neste exemplo, com apenas duas formas normais de *traces* é possível descrever o conjunto completo de 28 soluções de uma maneira compacta.

O trabalho de Braga *et al.* realiza uma combinação dos conceitos apresentados por Siepel [66] e Bergeron *et al.* [9] para produzir um algoritmo capaz de produzir os *traces* de todas as soluções ótimas e contar o número total de soluções ótimas possíveis [14].

Para qualquer *optimal i-sequence* s, se utilizarmos as relações de equivalência deduzidas a partir da comutação de reversões, podemos definir um *trace* que contém s. Esse *trace* é denominado *i-trace*.

Um *k-trace* T' é um prefixo de um *i-trace* T $(k \leq i)$ se T' contém um *k-sequence* que é prefixo de uma *i-sequence* de T. Isto é equivalente a dizer que cada *k-sequence* de T' é um prefixo de uma *i-sequence* de T.

Todo prefixo de tamanho k de uma *optimal i-sequence* está em um *k-trace* de *optimal k-sequences*. Assim, ao invés de realizar a enumeração de todas as *i-sequences* para posterior cálculo e comparação de todos os *traces*, torna-se mais vantajoso o processo de enumeração e comparação direta de todos os *i-traces*.

A enumeração de todos os *i-traces* pode ser feita de maneira similar à obtenção da forma normal de um *(i+1)-trace* a partir de um *i-trace*. Este método pode ser aplicado para construir todas os *i-traces* de maneira incremental sem a necessidade de se computar todas as soluções. Sem custo adicional, é possível computar o número de sequências que são representadas por cada *i-trace*.

A cada iteração do algoritmo, um conjunto $\mathcal{T}$ armazena todas as formas normais e tamanhos de *i-traces* de soluções para π. Para $i = 1$, cada *1-trace* é gerado com auxílio do algoritmo descrito por Siepel. O tamanho de cada *1-trace* é 1.

Para $2 \leq i \leq d(\pi)$, $\mathcal{T}$ contém todas as formas normais e os tamanhos dos $(i-1)$-*traces*. Cada *i-trace* possui um prefixo neste conjunto, pois um prefixo de tamanho $i-1$ de uma

optimal i-sequence é uma *optimal* $(i-1)$*-sequence*. Assim, cada *i-trace* é obtido a partir de um $(i-1)$*-trace* com o algoritmo de Siepel.

A cardinalidade de um *i-trace* T é a soma das cardinalidades de seus $(i-1)$ prefixos. Essa propriedade permite que a cardinalidade de cada *trace* seja computada e que assim, o número total de soluções de uma permutação π possa ser obtido.

Nos testes realizado por Braga *et al.*, o algoritmo, que tem complexidade exponencial, apresentou bom desempenho e o fator limitante de sua execução é o consumo de memória. Em um computador pessoal com 1Gb de RAM, o algoritmo consegue lidar com permutações com $d(\pi)$ inferior a 20. Apesar do espaço de solução ter caído drasticamente com a utilização de *traces*, ele pode ser muito grande para ser tratado por biólogos, principalmente quando grandes permutações são estudadas. Assim, os autores acreditam que o limite do algoritmo coincide com o limite da utilidade da solução e que uma outra estrutura de dados teria que ser inventada para solucionar o problema das grandes permutações.

Em um trabalho mais recente, Braga *et al.* implementam uma extensão do algoritmo que permite a redução do espaço da solução através da eliminação da necessidade de se enumerar todo o conjunto de *traces* [15].

A extensão do algoritmo baseia-se numa variação do estudo de ordenação por reversões denominado "ordenação perfeita". Nesta variação, trabalha-se com o conceito de *intervalo comum*.

Um *intervalo comum* de uma permutação π é um intervalo de π que também é um intervalo da permutação identidade. Estes intervalos são utilizados para modelar grupo de genes homólogos que podem ser encontrados nas duas espécies representadas pelas permutações. A idéia aqui é que se genes se encontram juntos em ambas as espécies, então é provável que o ancestral comum a elas também possuía os mesmos genes agrupados e que, portanto, eles não foram separados pela evolução.

Sendo assim, uma sequência de reversões $\rho_1 \dots \rho_k$ que ordena uma permutação π é dita "perfeita" se não existe reversão em $\rho_1, \dots, \rho_k$ que sobreponha um intervalo comum de π.

Para integrar o conceito de segmentos conservados ao conceito de *traces*, os autores fazem a seguinte proposição: todo *trace* de soluções para uma permutação por reversões com sinais contém somente soluções perfeitas ou não possui nenhuma solução perfeita.

Se uma sequência $s = \rho_1 \dots \rho_k$ que ordena uma permutação π é perfeita, então, por definição nenhum dos elementos $\rho_1, \dots, \rho_k$ sobrepõem um intervalo comum de π. Assim, qualquer sequência com o mesmo conjunto de reversões, mas em diferentes ordens, é perfeita. Este é o caso para todas as sequências de um *trace*. Portanto, se um uma sequência de um *trace* é perfeita, todas as sequências representadas por ele são perfeitas também.

Um *trace* perfeito é aquele que contém somente soluções perfeitas de tamanho $d(\pi)$. Este *trace* nem sempre existe o que significa que em cenários parsimoniosos, intervalos comuns podem ser quebrados [27].

A modificação imposta no algoritmo para encontrar os *traces* perfeitos consiste na interrupção da exploração de um *i-trace* quando uma reversão encontrada sobrepõe um intervalo comum. Deste modo, é possível reduzir o tempo de execução e diminuir consideravelmente a quantidade de memória necessária.

5 Alinhamento de Sequências com Reversões

Alinhamentos de sequências são frequentemente usados para comparação de sequências biológicas. Eles são associados a um conjunto de operações de edição que transformam um sequência na outra.

Normalmente, as operações consideradas são *substituição* (mutação) de um símbolo por outro, *inserção* de um símbolo e *remoção* de um símbolo. Se associarmos pesos para estas operações, podemos utilizar um algoritmo de programação dinâmica clássico de complexidade $O(n^2)$ que calcula o conjunto de operações de edição de custo mínimo e exibe o alinhamento associado. No entanto, outros importantes eventos biológicos como as reversões não são detectadas por estes algoritmos.

Uma reversão pode ser definida como uma operação de edição que substitui qualquer segmento por seu complemento reverso. Assim, é possível definir uma nova classe de problemas: dadas duas sequências e pesos fixos para cada tipo de operação de edição, o alinhamento com inversões é um problema de otimização que busca o custo mínimo total de uma sequência de operações que transforma uma sequência em outra. Adicionalmente, espera-se que as sequências de edições e/ou o alinhamento correspondente sejam obtidos.

Em 1992, Schöniger e Waterman [64] descrevem uma simplificação ao problema exigindo que todas as regiões envolvidas nas operações de inversões não se sobreponham. Esta simplificação levou ao problema de *alinhamento com inversões que não se sobrepõem* e estes autores apresentaram um algoritmo simples de programação dinâmica com complexidade $O(n^6)$. Eles também introduziram algumas heurísticas que reduziam o tempo de execução médio para algo entre $O(n^2)$ e $O(n^4)$.

Jiang *et al.* [21] mostram que o problema de decisão associado ao alinhamento com inversões para um alfabeto de tamanho ilimitado é NP-difícil.

Outros estudos independentes [28–30, 36] produziram algoritmos exatos para este problema com complexidade $O(n^4)$ de tempo e $O(n^2)$ de espaço.

Em 2005, Vellozo, Alves e do Lago apresentaram um algoritmo que resolve o problema em tempo $O(n^3 \log n)$ [1]. Um ano depois, os mesmos autores apresentaram um algoritmo que possui complexidade $O(n^3)$ de tempo e $O(n^2)$ de espaço [67].

6 Estágio no Exterior

Dentro do escopo do projeto de Doutorado, programamos um período de um ano de estágio no exterior (Bolsa Sanduíche a ser solicitada para a Capes).

O estágio, que tem previsão de início em Março de 2008, será desenvolvido nas dependências do grupo `BAMBOO-BAOBAB`, liderado pela Professora Doutora Marie-France Sagot, pesquisadora de prestígio internacional nas áreas de Bioinformática e Biologia Computacional. O laboratório do grupo `BAMBOO-BAOBAB` se situa em Lyon, França, dentro da *Université Claude Bernard (Lyon I)*.

Os pesquisadores que são os autores de trabalhos que utilizamos como base desse projeto passaram pelo Laboratório do grupo `BAMBOO-BAOBAB`. Marília Dias Vieira Braga desenvolveu toda a pesquisa nas dependências do laboratório e está concluindo neste ano de 2008 seu doutoramento, financiado pelo programa Alban da União Européia, sob o tema "Estrutura e Dinâmica de Genomas: Modelos e Algoritmos" tendo como orientadores os professores

Marie-France Sagot e Eric Tanier. Augusto Fernandes Vellozo concluiu seu doutoramento em 2007 na USP com o projeto "Alinhamento de Sequências com Rearranjo". Neste ano de 2008, Vellozo está no laboratório do grupo `BAMBOO-BAOBAB` como pesquisador visitante.

Acreditamos, portanto, que a realização deste estágio nas dependências do laboratório do grupo será um intercâmbio muito frutífero para ambas as partes. Poderemos trabalhar diretamente com pessoas familiarizadas com os problemas que pretendemos abordar e, além disso, estaremos dando continuidade a trabalhos desenvolvidos por pesquisadores do laboratório.

Em Fevereiro de 2008, visitamos o laboratório do grupo e, desde então, mantemos contato frequente com a Professora Doutora Marie-France Sagot, que apóia a realização do estágio.

7 Projeto

O objetivo do projeto desenvolvido neste doutorado é promover a investigação em dois tópicos principais de pesquisa: *Enumeração de Soluções de Distância de Reversão* e *Alinhamento de Sequências Utilizando Eventos de Rearranjo*.

7.1 Enumeração de Soluções de Distância de Reversão

Para o tópico de Enumeração de Soluções de Distância de Reversão, partiremos da base construída por Braga *et al.* e apresentada nos trabalhos "The Solution Space of Sorting by Reversals" [14] e "Exploring the solution space of sorting by reversals with experiments and an application to evolution" [15].

Nestes trabalhos, Braga *et al.* apresentam um método para a enumeração de todos os *traces* de soluções de distância de reversão. Este método funciona bem para permutações que possuem $d(\pi)$ menor do que 20 (em um computador pessoal com 1GB de RAM), oferecendo aos biólogos uma gama maior de possibilidades para a análise da distância evolucionária entre espécies diferentes. O maior limite da aplicação está na quantidade de memória necessária para a representação dos dados intermediários no cálculo do imenso conjunto de soluções. Os próprios autores acreditam que para otimizar o código implementado seria necessária a criação de uma nova estrutura de dados. Por outro lado, um solução adotada por eles foi a adição de restrições biológicas com o objetivo de reduzir o tamanho do conjunto de soluções possíveis.

O laboratório do grupo `BAMBOO-BAOBAB` tem um interesse especial no estudo evolucionário de espécies simbiontes. Utilizando o algoritmo de enumeração de reversões, os pequisadores desenvolveram um trabalho inicial foi com bactérias do gênero *Rickettsia*.

O grupo tem interesse de estudar também o gênero *Wolbachia*, que também é composto por bactérias simbiontes. Em uma primeira etapa do projeto, aplicaremos para o gênero *Wolbachia* os métodos empregados no gênero *Rickettsia*. Esta etapa será de fundamental importância para obtenção de um melhor entendimento do problema, identificação de possíveis melhorias para aperfeiçoamento do algoritmo ou, até mesmo, criação de novos métodos para solucionar o problema.

Após a realização da etapa inicial, trabalharemos na pesquisa de alternativas para redução do consumo de memória do algoritmo. O objetivo aqui é ampliar o limite atual

do algoritmo para que ele seja capaz de trabalhar com permutações com distância de reversão maiores do que 20. Trabalhando em cooperação com os pesquisadores do grupo `BAMBOO-BAOBAB`, estudaremos também a possibilidade de incluir outras restrições biológicas.

Finalmente, um tópico de pesquisa mais avançado que pretendemos abordar é a fusão do conceito de *traces* com o algoritmo de aproximação para ordenação de permutações utilizando transposições, proposto por Elias e Hartman e que possui fator de aproximação 1.375 [34]. A nossa idéia aqui é de gerar um algoritmo capaz de listar os *traces* de soluções para o problema de ordenação de permutações por transposições com auxílio do algoritmo de aproximação.

7.2 Alinhamento de Sequências Utilizando Eventos de Rearranjo

Para o tópico de Alinhamento de Sequências Utilizando Eventos de Rearranjo, pretendemos estender o trabalho desenvolvido por Vellozo *et al.* e apresentado no artigo "Alignment with Non-overlapping Inversions in $O(n^3)$-Time" [67]. Neste trabalho, os autores introduzem um novo algoritmo de complexidade $O(n^3)$, que permite a realização do alinhamento de sequências utilizando inversões que não se sobrepõem.

Numa fase inicial do estudo deste tópico, pretendemos investigar alguns pontos interessantes que não foram abordados pelos autores.

O primeiro ponto de estudo seria adaptar o algoritmo ou criar novos algoritmos que realizam o alinhamento considerando um esquema de penalização afim para os buracos, algo que ainda não é contemplado pelo algoritmo. A motivação para se utilizar uma função linear do tipo $w(k) = g + hk$, onde k é o número de buracos, g é a pontuação para se abrir o buraco e h é a pontuação para se estender o buraco, está no fato de que, geralmente, a existência de buracos contíguos de tamanho k é mais frequente do que a existência de k buracos isolados [65].

Pinter e Skiena desenvolveram um trabalho para ordenação de permutações não orientadas utilizando reversões [59]. Para o procedimento de ordenação, cada reversão recebia um peso dependente do comprimento das sequências revertidas. Eles obtiveram como resultado um algoritmo de aproximação que utilizava o comprimento das reversões como critério para otimização. A motivação para esta abordagem está no fato de que pequenos rearranjos são mais comuns do que os grandes. Isso significa que um rearranjo de uma pequena região do genoma tem uma probabilidade maior de ocorrência do que o rearranjo de uma grande região. Assim, a idéia seria criar uma função que pontue as inversões conforme os tamanhos que elas apresentem.

Bender *et al.* publicaram um trabalho para ordenação de permutações utilizando reversões com pesos [7]. Neste trabalho, o peso da reversão é generalizado pela função $f(l) = l^\alpha$, onde l é o comprimento da reversão e α é uma constante. Com base nesta função, os autores produziram um algoritmo para aproximação do custo ótimo para se ordenar um dada entrada.

Baseados no conceito utilizado nos trabalhos de Pinter e Skiena e de Bender *et al.*, estudaremos a adoção de diferentes funções e implementaremos as adaptações necessárias para que o algoritmo incorpore a utilização de funções que atribuam pesos para as reversões, proporcionais aos seus tamanhos.

O algoritmo desenvolvido por Vellozo *et al.* utiliza espaço de memória $O(n^2)$. Logo, um tópico de pesquisa interessante é a alteração do algoritmo para que ele seja capaz de utilizar uma quantidade menor de memória sem afetar negativamente a complexidade de tempo. Para alcançar este objetivo, utilizaremos as idéias introduzidas por Miller e Myers na implementação do algoritmo de alinhamento global de sequências que utiliza espaço linear [57].

Em uma fase mais avançada do estudo, pretendemos adicionar ao algoritmo a capacidade de realizar alinhamentos de sequências considerando, além das reversões, as transposições.

8 Cronograma

Nesta seção, detalharemos o cronograma de atividades do projeto. Este cronograma pode ser dividido em três etapas distintas. A primeira etapa, que pode ser vista na Tabela 1, refere-se as atividades realizadas nos primeiros semestres do curso: disciplinas obrigatórias, levantamento bibliográfico de artigos relacionados aos temas da pesquisa e preparação para o exame de qualificação específico. Nesta primeira etapa, também ocorreu a nossa visita ao laboratório do grupo `BAMBOO-BAOBAB`.

A segunda etapa agrupa as atividades previstas para a fase preparatória de familiarização com os algoritmos pesquisados (Setembro de 2008 a Fevereiro de 2009). Nesta etapa agrupamos, também, as atividades previstas para o estágio no exterior. A distribuição das atividades desta etapa pode ser vista na Tabela 2.

Finalmente, a terceira etapa reúne as atividades reservadas para o retorno ao Brasil. Reservamos para esse período os tópicos de pesquisa mais avançados pois envolvem a modificação ou criação de algoritmos que incorporem o evento de transposição. Além destas atividades de pesquisa, a Tabela 3 também lista a programação de final de doutorado: término da escrita da tese, defesa e entrega da versão final.

1. Disciplinas obrigatórias;
2. Levantamento bibliográfico;
3. Visita ao laboratório do grupo `BAMBOO-BAOBAB`;
4. Preparação para Exame de Qualificação Específico;
5. Aplicação do algoritmo de enumeração de soluções de ordenação de permutações orientadas utilizando reversões ao gênero *Wolbachia*;
6. Incorporação de pontuação afim ao algoritmo de alinhamento de sequências com reversões que não se sobrepõem;
7. Estágio no Exterior:
   (a) Redução do consumo de memória do algoritmo de enumeração de soluções de ordenação de permutações orientadas utilizando reversões;
   (b) Redução do consumo de memória do algoritmo de alinhamento de sequências com reversões que não se sobrepõem;
   (c) Incorporação de função de peso para reversões no algoritmo de alinhamento de sequências com reversões que não se sobrepõem;

Tabela 1: Cronograma de Atividades : Agosto 2006 – Agosto 2008

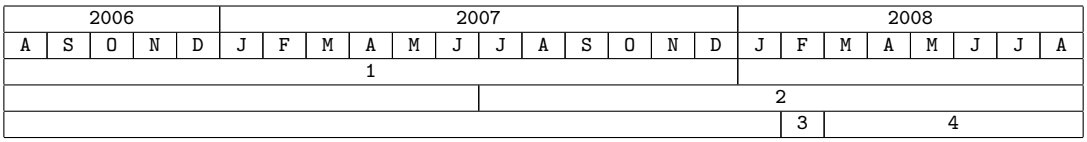


Tabela 2: Cronograma de Atividades : Setembro 2008 – Fevereiro 2010

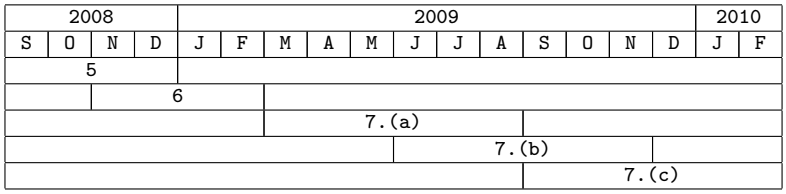
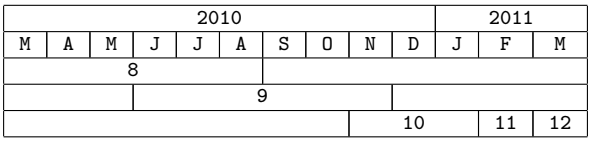


Tabela 3: Cronograma de Atividades : Março 2010 – Março 2011



8. Desenvolvimento de algoritmo de alinhamento de sequências utilizando transposições;
9. Criação de algoritmo para enumeração de soluções do problema de ordenação de permutações utilizando transposições;
10. Conclusão da escrita da tese;
11. Defesa;
12. Entrega da versão final da tese.

Referências

[1] C. E. R. Alves, A. P. do Lago, and A. F. Vellozo. Alignment with non-overlapping inversions in $O(n^3 \log n)$-time. In *Proceedings of GRACO 2005*, volume 19 of *Electronic Notes in Discrete Mathematics*, pages 365–371, Amsterdam, 2005. Elsevier.

[2] 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. In *Proceedings of the Seventh Workshop on Algorithms and Data Structures (WADS'01)*. Springer Verlag, 2001.

[3] V. Bafna and P. A. Pevzner. Sorting by reversals: Genome rearrangements in plant organelles and evolutionary history of X chromosome. *Molecular Biology and Evolution*, 12(2):239–246, 1995.

[4] 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, January 1995.

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

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

[7] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting with length-weighted reversals. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 919–928, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[8] A. Bergeron. A Very Elementary Presentation of the Hannenhalli-Pevzner Theory. In *Proceedings of the 12th Annual Symposium of the Combinatorial Pattern Matching (CPM'2001)*, volume 2089 of *Lecture Notes in Computer Science*, pages 106–117, Berlin, Germany, September 2001. Springer-Verlag.

[9] A. Bergeron, C. Chauve, T. Hartman, and K. Saint-Onge. On the Properties of Sequences of Reversals that Sort a Signed Permutation. In *Proceedings of the JOBIM 2002*, pages 99–108, 2002.

[10] A. Bergeron and F. Strasbourg. Experiments in Computing Sequences of Reversals. In *WABI '01: Proceedings of the First International Workshop on Algorithms in Bioinformatics*, pages 164–174, London, UK, 2001. Springer-Verlag.

[11] P. Berman and S. Hannenhalli. Fast Sorting by Reversal. In D. S. Hirschberg and E. W. Myers, editors, *Proceedings of Combinatorial Pattern Matching (CPM'96), 7th Annual Symposium*, volume 1075 of *Lecture Notes in Computer Science*, pages 168–185, Laguna Beach, USA, January 1996. Springer.

[12] 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)*, Lecture Notes in Computer Science, Rome, Italy, September 2002. Springer.

[13] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric Genome Rearrangement. *Journal of Computational Biology*, 172:11–17, 1996.

[14] M. D. V. Braga, M.-F. Sagot, C. Scornavacca, and E. Tanier. The Solution

Space of Sorting by Reversals. In *Bioinformatics Research and Applications*, volume 4463, pages 293–304. Springer Berlin / Heidelberg, 2007. Proceedings of the International Symposium on Bioinformatics Research and Applications 2007 (ISBRA 2007).

[15] M. D. V. Braga, M.-F. Sagot, C. Scornavacca, and E. Tanier. Exploring the solution space of sorting by reversals with experiments and an application to evolution. *Transactions on Computational Biology and Bioinformatics*, 2008.

[16] A. Caprara. Sorting by Reversals is Difficult. Technical report, DEIS - Operations Research Group, University of Bologna, April 1996.

[17] A. Caprara. Formulations and Hardness of Multiple Sorting by Reversals. In S. Istrail, P. A. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB'99)*, pages 84–93, Lyon, France, 1999. ACM Press.

[18] A. Caprara. On the Tightness of the Alternating-Cycle Lower Bound for Sorting by Reversals. *Journal of Combinatorial Optimization*, 3:149–182, 1999.

[19] A. Caprara and G. Lancia. Experimental and Statistical Analysis of Sorting by Reversals. In D. Sankoff and J. H. Nadeau, editors, *Comparative Genomics: Empirical and Analyitical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families*. Kluwer Academic Publishers, Le Chantecler, Canada, September 2000.

[20] P. Cartier and D. Foata. *Problèmes combinatoires de commutation et réarrangements*. Number 85 in Lecture Notes in Mathematics. Springer-Verlag, Berlin, 1969.

[21] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of Orthologous Genes via Genome Rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[22] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, November 1996.

[23] D. A. Christie. A 3/2-Approximation Algorithm for Sorting by Reversals. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 244–252, San Francisco, USA, January 1998.

[24] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, Glasgow University, 1998.

[25] A. J. Clutterbuck and S. J. O'Brien. *Genetics Maps: Locus Maps of Complex Genomes*. Cold Spring Harbor Laobratory Press, New York, 6 edition, 1993.

[26] 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, November 2001. IEEE Computer Society.

[27] Y. Diekmann, M.-F. Sagot, and E. Tanier. Evolution under reversals: parsimony and conservation of common intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):301–309, 2005.

[28] A. P. do Lago, C. A. Kulikowski, E. Linton, J. Messing, and I. Muchnik. Comparative genomics: simultaneous identification of conserved regions and their

rearrangements through global optimization. In *The Second University of São Paulo/Rutgers University Biotechnology Conference*, Rutgers University Inn and Conference Center, New Brunswick, NJ, August 2001.

[29] A. P. do Lago, I. Muchnik, and C. Kulikowski. An $O(n^4)$ algorithm for alignment with non-overlapping inversions. In *Second Brazilian Workshop on Bioinformatics, WOB 2003*, Macaé, RJ, Brazil, 2003.

[30] A. P. do Lago, I. Muchnik, and C. A. Kulikowski. A sparse dynamic programming algorithm for alignment with non-overlapping inversions. *Theoretical Informatics and Applications*, 39:175–189, 2005.

[31] T. Dobzhansky and A. H. Sturtevant. Inversions in the third chromosome of wild races of *Drosophila pseudoobscura*, and their use in the study of the history of the species. *Proceedings of the National Academy of Science*, 22:448–450, July 1936.

[32] T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *Drosophila pseudoobscura*. *Genetics*, 23(1):28, 1938.

[33] N. El-Mabrouk and D. Sankoff. Genome Rearrangement. In T. Jiang, T. Smith, Y. Xu, and M. Zhang, editors, *Current Topics in Computational Biology*. MIT Press, Cambridge, 2001.

[34] I. Elias and T. Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions. In *Proceedings of 5th Workshop on Algorithms in Bioinformatics (WABI'05)*, volume 3692 of *Lecture Notes in Bioinformatics*, pages 204–215, 2005.

[35] N. Eriksen. $(1 + \epsilon)$-Approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517–529, October 2002.

[36] Y. Gao, J. Wu, R. Niewiadomski, Y. Wang, Z-Z. Chen, and G. Lin. A Space Efficient Algorithm for Sequence Alignment with Inversions. In *Computing and Combinatorics, 9th Annual International Conference COCOON 2003*, volume 2697 of *Lecture Notes in Computer Science*, pages 57–67. Springer Berlin / Heidelberg, 2003.

[37] Q.-P. Gu and K. Iwata. A Heuristic Algorithm for Genome Rearrangements, December 1997. Poster in The Eighth Workshop on Genome Informatics (GIW'97).

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

[39] S. Hannenhalli. Polynomial-time Algorithm for Computing Translocation Distance Between Genomes. *Discrete Applied Mathematics*, 71:137–151, 1996.

[40] S. Hannenhalli, C. Chappey, E. V. Koonin, and P. A. Pevzner. Genome Sequence Comparison and Scenarios for Gene Rearrangements: a Test Case. *Genomics*, 30:299–311, 1995.

[41] S. Hannenhalli and P. A. Pevzner. Towards a Computational Theory of Genome Rearrangements. *Lecture Notes in Computer Science*, 1000:184–200, 1995.

[42] S. Hannenhalli and P. A. Pevzner. Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals). In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 178–189, Las Vegas, USA, May 1995.

[43] S. Hannenhalli and P. A. Pevzner. Transforming Men into Mice (Polynomial Algorithm for Genomic Distance Problem). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 581–592, Los Alamitos, USA, October 1995. IEEE Computer Society Press.

[44] S. Hannenhalli and P. A. Pevzner. To Cut... or Not to Cut (applications of comparative physical maps in molecular evolution). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 304–313, Atlanta, USA, January 1996.

[45] 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, January 1999.

[46] L. S. Heath and J. P.Vergara. Sorting by Bounded Block-Moves. *Discrete Applied Mathematics, Second Special Issue on Computational Biology*, 88:181–206, 1998.

[47] L. S. Heath and J. P. Vergara. Sorting by Short Block-Moves. *Algorithmica*, June 2000. Online publication.

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

[49] H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals. *SIAM Journal on Computing*, 29(3):880–892, January 2000.

[50] J. D. Kececioglu and R. Ravi. Of Mice and Men: Algorithms for Evolutionary Distances Between Genomes with Translocation. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 604–613, New York, USA, January 1995. ACM Press.

[51] J. D. Kececioglu and D. Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13:180–210, January 1995.

[52] G. M. Landau and M. Ziv-Ukelson. On the common substring alignment problem. *Journal of Algorithms*, 41(2):338–354, 2001.

[53] G. Li, X. Qi, X. Wang, and B. Zhu. *Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, chapter A Linear-Time Algorithm for Computing Translocation Distance between Signed Genomes, pages 323–332. Springer Berlin / Heidelberg, 2004.

[54] Z. Li and L. Wang. Algorithmic Approaches for Genome Rearrangement: A Review. *IEEE Transactions on Systems, Man, and Cybernetics*, 36(5):636–648, September 2006.

[55] G.-H. Lin and G. Xue. Signed Genome Rearrangement by Reversals and Transpositions: Models and Approximations. In *Fifth Annual International Computing and Combinatorics Conference (COCOON'99)*, pages 71–80, 1999.

[56] J. Meidanis, M. E. M. T. Walter, and Z. Dias. A Lower Bound on the Reversal and Transposition Diameter. Technical Report IC-00-16, Institute of Computing - University of Campinas, October 2000.

[57] E. W. Myers and W. Miller. Optimal alignments in linear space. *Comput. Applic. Biosci.*, 4:11–17, 1988.

[58] J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 27:87–97, 1988.

[59] R. Y. Pinter and S. Skiena. Genomic Sorting with Length-Weighted Reversals. *Genome Informatics*, 13:103–111, 2002.

[60] N. Saheb. Concurrency measure in commutation monoids. *Discrete Applied Mathematics*, 24:223–236, 1989.

[61] D. Sankoff. Edit distance for genome comparison based on non-local operations. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the Third Annual Symposium of the Combinatorial Pattern Matching (CPM'92)*, number 664 in Lecture Notes in Computer Science, pages 121–135, Tucson, USA, 1992. Springer-Verlag.

[62] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. F. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy Science, USA*, 89:6575–6579, 1992.

[63] J. P. Schmidt. All Shortest Paths in Weighted Grid Graphs and its Application to Finding All Approximate Repeats in Strings. In *Israel Symposium on Theory of Computing Systems*, pages 67–77, 1995.

[64] M. Schöniger and M. S. Waterman. A local algorithm for DNA sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54(4):521–536, 1992.

[65] J. C. Setubal and J. Meidanis. *Introduction to Computional Molecular Biology*. PWS Publishing Company, 1997.

[66] A. C. Siepel. An Algorithm to Enumerate Sorting Reversals. *Journal of Computational Biology*, 10(3–4):575–597, 2003.

[67] A. F. Vellozo, C. E. R. Alves, and A. P. do Lago. Alignment with Non-overlapping Inversions in $O(n^3)$-Time. In *Algorithms in Bioinformatics*, volume 4175 of *Lecture Notes in Computer Science*, pages 186–196. Springer-Verlag Berlin Heidelberg, 2006. Proceedings of the WABI 2006.

[68] M. E. M. T. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *Proceedings of the String Processing and Information Retrieval (SPIRE'98)*, 1998.

[69] 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)*, September 2000.

[70] L. Wang, D. Zhu, X. Liu, and S. Ma. An $O(n^2)$ algorithm for signed translocation. *Journal of Computer and System Sciences*, 70(3):284–299, 2005.

[71] L.-S. Wang. Exact-IEBP: A New Technique for Estimating Evolutionary Distances between Whole Genomes. *Lecture Notes in Computer Science*, 2149:175–188, 2001.

[72] L.-S. Wang and T. Warnow. Estimating true evolutionary distances between genomes. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 637–646, New York, NY, USA, 2001. ACM.

[73] D. M. Zhu and S. H. Ma. Improved polynomial-time algorithm for computing translocation distance between genomes. *Chinese J. Comput.*, 25(2):189–196, 2002. In Chinese.

A Revisão Bibliográfica

Nesta revisão bibliográfica iremos extrair os objetivos de cada artigo e citar os principais tópicos de interesse para o trabalho a ser desenvolvido durante o projeto de doutorado.

A escolha dos textos foi feita de modo que eles englobassem aspectos relacionados, direta ou indiretamente, ao nosso estudo. Dessa maneira, visamos a aquisição de um conhecimento mais amplo sobre o contexto da pesquisa e sobre as soluções existentes para os problemas ligados à área de Rearranjo de Genomas.

A.1 The Solution Space of Sorting by Reversals [14]

Em genômica comparativa, algoritmos que ordenam permutações através de reversões são frequentemente utilizados para proposição de cenários evolucionários no estudo de mutações genômicas entre espécies. Um dos maiores problemas destes algoritmos é que eles produzem como saída apenas uma solução dentro de um conjunto enorme de soluções ótimas. Além disso, não existem critérios que diferenciem estas soluções entre si.

O trabalho de Bergeron *et al.* [9] apresenta uma forma de classificar o conjunto de soluções ótimas. Através da utilização do conceito de *traces*, as soluções ótimas podem ser agrupadas em classes de equivalência e uma solução representativa pode ser escolhida para representar a classe. Contudo, até então, o único algoritmo disponível para o cálculo do conjunto completo de soluções ótimas representativas era a enumeração de todas as permutações possíveis para realização da classificação.

Este artigo tem o objetivo de propor um algoritmo que seja capaz de produzir uma solução representativa para cada classe e de contar o número de soluções em cada classe, com uma complexidade prática e teórica melhor do que o método da enumeração de todas as soluções possíveis.

Se uma sequência de reversões for identificada como uma palavra no alfabeto $\mathcal{A}$ de reversões, Bergeron *et al.* [9] definem uma relação de equivalência sobre estas palavras: se ρ e θ são reversões e não se sobrepõem, então as palavras $\rho\theta$ e $\theta\rho$ são equivalentes. Dizemos que, nesse caso, ρ e θ comutam. Baseada nesta relação, qualquer palavra contendo $\rho\theta$ como sub-palavra é equivalente a mesma palavra que substitui $\rho\theta$ por $\theta\rho$.

Por exemplo, para a permutação $(4, -3, -1, 2)$, podemos considerar como uma solução a sequência de reversões $\{1,3,4\}\{2,4\}\{2,3\}\{3\}$. Neste caso, a reversão $\{3\}$ comuta com todas as outras reversões e nenhuma das outras reversões comutam entre si. Assim, podemos dizer que as sequências $\{1,3,4\}\{2,4\}\{3\}\{2,3\}$, $\{1,3,4\}\{3\}\{2,4\}\{2,3\}$ e $\{3\}\{1,3,4\}\{2,4\}\{2,3\}$ são soluções equivalentes pela comutação de $\{3\}$ com todas as outras reversões.

Se considerarmos a sequência de reversões $\{1\}\{1,2\}\{4\}\{1,2,3,4\}$, temos que $\{4\}$ e $\{1,2\}$ comutam e, por isso, $\{1\}\{1,2\}\{4\}\{1,2,3,4\}$ é equivalente a $\{1\}\{4\}\{1,2\}\{1,2,3,4\}$. Na realidade, neste exemplo, todas as reversões comutam entre si e, por isso, todas as permutações destas quatro reversões também são soluções.

Uma classe de equivalência de sequências ótimas de reversão sobre a relação de equivalência descrita acima é denominada *trace*. Bergeron *et al.* propõem que para uma permutação sinalizada π, o conjunto de todas as reversões ótimas é uma união de *traces*.

Como consequência, temos que se o conjunto de soluções que devem ser enumeradas é muito grande, o conjunto de *traces* pode ser um resultado mais relevante para o proble-

ma de ordenação por reversões. Porém, ainda é necessária uma forma de representação compacta para os *traces*.

Um elemento s de um *trace* T está em sua forma normal se ele pode ser decomposto em sub-palavras $s = u_1|\dots|u_m$ tais que:

- todo par de elementos de uma sub-palavra u_i comutam;
- para todo elemento ρ de uma sub-palavra u_i $(i > 1)$, existe pelo menos um elemento θ da palavra u_{i-1} tal que ρ e θ não comutam;
- toda sub-palavra u_i é uma palavra crescente não vazia com relação à ordem lexicográfica induzida por $\mathcal{A}$

Segundo um teorema de Cartier e Foata [20], para qualquer *trace* existe uma única palavra que está na forma normal. Assim, é possível representar os *traces* através das suas representações na forma normal.

No caso da permutação $(4, -3, -1, 2)$, por exemplo, temos dois *traces* de soluções ótimas: $\{1\}\{1,2\}\{1,2,3,4\}\{4\}$ e $\{1,3,4\}\{3\}|\{2,4\}|\{2,3\}$. Neste exemplo, com apenas duas formas normais de *traces* é possível descrever o conjunto completo de 28 soluções de uma maneira compacta.

Bergeron *et al.* não apresentam um algoritmo que permita a representação de todos os *traces* das soluções de uma ordenação por reversões. Assim, o melhor algoritmo para listagem de todos os *traces*, até então, era a enumeração de todas as soluções para cálculo do *trace* associado e adição deste a uma listagem de *traces* encontrados.

Seja $d(\pi)$ a distância da permutação π em relação à permutação identidade. Uma sequência de reversões $s = \rho_1\rho_2\dots\rho_i$ é chamada de *optimal i-sequence* quando $d(\pi \cdot \rho_1\rho_2\dots\rho_i) = d(\pi) - i$. Note que se $i = d(\pi)$, então s é uma solução ótima.

Para enumeração de todas as soluções ótimas podemos utilizar o algoritmo proposto por Siepel [66] que permite que o conjunto de todas as *optimal 1-sequences* de uma permutação seja calculado em tempo $O(n^3)$.

O conjunto de todas as *optimal i-sequences* pode ser calculado a partir do conjunto de $(i-1)$-*sequences* através da iteração do mesmo algoritmo. A utilização desta solução resulta em um algoritmo capaz de listar todo o conjunto de soluções ótimas em tempo $O(n^{2n+3})$.

Para qualquer *optimal i-sequence* s, se utilizarmos as relações de equivalência deduzidas a partir da comutação de reversões, podemos definir um *trace* que contém s. Esse *trace* é denominado *i-trace*.

Dada uma sequência de reversões $s = \rho_1\rho_2\dots\rho_d$ para uma permutação π com distância de reversão d, a forma normal do *trace* T que contém s é construída a partir da iteração de um inteiro 1 até d, em que a cada passo i, o elemento ρ_i, representado pelo conjunto ordenado de seus valores, é adicionado à forma normal $(i-1)$-*trace*, que já contém os elementos $\rho_1\dots\rho_{i-1}$.

O *trace* construído da maneira descrita acima é então comparado com o conjunto de formas normais de *traces* para eliminação de redundância.

Considerando-se a complexidade de tempo de todas as operações envolvidas nesta construção, temos que para a determinação do conjunto de todas as formas normais de *traces* associado à uma permutação π, a complexidade de tempo é $O(n^{2n+3})$. Esta complexidade nos mostra que este método não pode ser aplicado para grandes permutações.

No lugar de enumerar todas as soluções ótimas para então construir o conjunto de *traces*, a alternativa proposta neste artigo opera de forma a realizar a enumeração de *traces*.

Um *k-trace* T' é um prefixo de um *i-trace* T ($k \leq i$) se T' contém um *k-sequence* que é prefixo de uma *i-sequence* de T. Isto é equivalente a dizer que cada *k-sequence* de T' é um prefixo de uma *i-sequence* de T.

Facilmente podemos ver que todo prefixo de tamanho k de uma *optimal i-sequence* está em um *k-trace* de *optimal k-sequences*. Assim, ao invés de realizar a enumeração de todas as *i-sequences* para posterior cálculo e comparação de todos os *traces*, torna-se mais vantajoso o processo de enumeração e comparação direta de todos os *i-traces*.

A enumeração de todos os *i-traces* pode ser feita de maneira similar à obtenção da forma normal de um *(i+1)-trace* a partir de um *i-trace*. Este método pode ser aplicado para construir todas os *i-traces* de maneira incremental sem a necessidade de se computar todas as soluções. Sem custo adicional, é possível computar o número de sequências que são representadas por cada *i-trace*.

A cada iteração do algoritmo, um conjunto $\mathcal{T}$ armazena todas as formas normais e tamanhos de *i-traces* de soluções para π. Para $i = 1$, cada *1-trace* é gerado com auxílio do algoritmo descrito por Siepel. O tamanho de cada *1-trace* é 1.

Para $2 \leq i \leq d(\pi)$, $\mathcal{T}$ contém todas as formas normais e os tamanhos dos $(i-1)$-*traces*. Cada *i-trace* possui um prefixo neste conjunto, pois um prefixo de tamanho $i-1$ de uma *optimal i-sequence* é uma *optimal* $(i-1)$-*sequence*. Assim, cada *i-trace* é obtido a partir de um $(i-1)$-*trace* com o algoritmo de Siepel.

A cardinalidade de um *i-trace* T é a soma das cardinalidades de seus $(i-1)$ prefixos. Essa propriedade permite que a cardinalidade de cada *trace* seja computada e que assim, o número total de soluções de uma permutação π possa ser obtido.

Sejam N o número de *d-traces*, onde $d = d(\pi)$, e k_{max} a largura máxima de um *d-trace* (de acordo com a definição utilizada no trabalho). A complexidade de tempo deste algoritmo para enumeração de todos os *traces* é $O(Nn^{k_{max}+4})$.

Alguns testes foram feitos com dados simulados aleatoriamente e dados biológicos. Estes testes comprovaram a vantagem do novo método em relação ao ganho de velocidade para o cálculo das soluções. Por exemplo, para um conjunto de dados composto pelos cromossomos X do rato e do homem ($n = 16$ e $d = 10$) temos que o número total de soluções é 2419750 e o número total de *traces* é 418. Enquanto o algoritmo de cálculo de todas as soluções para posterior cálculo dos *traces* gastava aproximadamente 13 minutos, o algoritmo para enumeração de *traces* concluía o processamento em aproximadamente 10 segundos.

O algoritmo se mostrou muito rápido. No entanto, ele está limitado a permutações pequenas devido à necessidade de memória durante o processamento. Em um computador pessoal com 1Gb de RAM, o algoritmo consegue lidar com permutações com $d(\pi)$ inferior a 20.

Apesar do espaço de solução ter caído drasticamente com a utilização de *traces*, ele pode ser muito grande para ser tratado por biólogos, principalmente quando grandes permutações são estudadas. Assim, o limite do algoritmo coincide com o limite da utilidade da solução. Os autores do artigo acreditam que uma outra estrutura de dados teria que ser inventada para solucionar o problema das grandes permutações.

A.2 Exploring the solution space of sorting by reversals, with experiments and an application to evolution [15]

O objetivo deste artigo é mostrar a evolução do trabalho apresentado anteriormente em *The Solution Space of Sorting by Reversals* [14] e apresentar uma aplicação da solução proposta na análise de um conhecido evento de evolução genômica.

Em sua parte inicial o artigo repete a apresentação dos algoritmos e análises de complexidades efetuadas no trabalho anterior.

A seguir, inicia-se a análise de uma extensão do algoritmo que visa reduzir o espaço de busca ao adotar restrições que eliminam a necessidade de exploração de todo o conjunto de *traces*.

Para isso, a extensão do algoritmo baseia-se numa variação do estudo de ordenação por reversões denominado "ordenação perfeita". Nesta variação, trabalha-se com o conceito de *intervalo comum*.

Um *intervalo comum* de uma permutação π é um intervalo de π que também é um intervalo da permutação identidade. Estes intervalos são utilizados para modelar grupo de genes homólogos que podem ser encontrados nas duas espécies representadas pelas permutações. A idéia aqui é que se genes se encontram juntos em ambas as espécies, então é provável que o ancestral comum a elas também possuía os mesmos genes agrupados e que, portanto, eles não foram separados pela evolução.

Sendo assim, uma sequência de reversões $\rho_1 \dots \rho_k$ que ordena uma permutação π é dita "perfeita" se não existe reversão em $\rho_1, \dots, \rho_k$ que sobreponha um intervalo comum de π.

O objetivo, então, é encontrar cenários ótimos com respeito ao princípio de segmentos conservados a partir da aplicação deste conceito aos *traces*.

Neste sentido, o artigo faz a seguinte proposição: todo *trace* de soluções para uma permutação por reversões com sinais contém somente soluções perfeitas ou não possui nenhuma solução perfeita.

Se uma sequência $s = \rho_1 \dots \rho_k$ que ordena uma permutação π é perfeita, então, por definição nenhum dos elementos $\rho_1, \dots, \rho_k$ sobrepõem um intervalo comum de π. Assim, qualquer sequência com o mesmo conjunto de reversões, mas em diferentes ordens, é perfeita. Este é o caso para todas as sequências de um *trace*. Portanto, se uma sequência de um *trace* é perfeita, todas as sequências representadas por ele são perfeitas também.

Um *trace* perfeito é aquele que contém somente soluções perfeitas de tamanho $d(\pi)$. Este *trace* nem sempre existe, o que significa que em cenários parcimoniosos, intervalos comuns podem ser quebrados [27].

A modificação imposta no algoritmo para encontrar os *traces* perfeitos consiste na interrupção da exploração de um *i-trace* quando uma reversão encontrada sobrepõe um intervalo comum. Deste modo, é possível reduzir o tempo de execução e diminuir consideravelmente a quantidade de memória necessária.

No caso da não existência de *traces* perfeitos, é possível determinar um valor de corte que determina o número máximo de reversões que podem sobrepor um intervalo comum para que *traces* quase-perfeitos sejam encontrados.

Uma investigação foi conduzida para avaliar a redução do espaço de soluções quando apenas soluções perfeitas ou quase-perfeitas são consideradas. Para isso, 100 permutações aleatórias com $n = 12$ e distância de reversão $2 \leq d \leq 11$ foram geradas. Para cada

permutação, foram computados o número de reversões que sobrepunham intervalos comuns para cada *trace* (se o número é igual a zero, então o *trace* é perfeito).

A partir da escolha de *traces* quase-perfeitos que minimizassem o número de reversões que sobrepunham os intervalos, foram calculados a taxa de soluções e *traces* que são perfeitos ou quase-perfeitos.

Em média, nas 100 permutações analisadas, o espaço de solução é dividido por 3 quando se aplica o critério de *traces* quase-perfeitos. Outra observação interessante, é que o conjunto de *traces* em média é dividido por 20, mostrando a redução no espaço de soluções.

O artigo mostra uma aplicação prática do algoritmo na análise de um evento de evolução genômica envolvendos os cromossomos humanos X e Y.

Acredita-se que os cromossomos X e Y evoluíram a partir de um par de autossomos idênticos. A evolução destes cromossomos é que deu a origem ao processo de diferenciação sexual: o par XX feminino e o par XY masculino. Devido ao mecanismo de recombinação, a organização feminina favoreceu a conservação do cromossomo X. No caso masculino, a evolução do par XY causou a divergência do cromossomo Y à medida que ele perdeu gradualmente a capacidade de recombinar com o cromossomo X.

Os cromossomos X e Y ainda dividem uma região "pseudo-autossomal" em uma de suas extremidades onde a recombinação acontece como nos autossomos. Teorias atuais sugerem que a região pseudo-autossomal, que originalmente cobria os cromossomos inteiros, foi alterada por algumas grandes reversões no cromossomo Y. As extremidades destas reversões se localizam em cada um dos lados do limite desta região pseudo-autossomal. Os sucessivos limites da região pseudo-autossomal no cromossomo X, dos pontos de origem aos pontos onde estão localizados agora, representam o que é denomidado de *strata* evolucionários dos cromossomos sexuais.

A análise dos cromossomos sugere a presença de cinco *strata* no cromossomo X. Estes *strata* são ordenados de acordo com o tempo de criação. Assim, o *stratum* mais próximo do limite da região autossomal recebe número 5 e o mais próximo da extremidade do cromossomo X recebe o número 1.

Para uma permutação orientada $\pi = (\pi_1, \ldots, \pi_n)$, um *k-strata* é definido como uma partição de π em um conjunto ordenado $B = (I_k, I_{k-1}, \ldots, I_1)$ de k intervalos tais que,

$$\begin{aligned} I_k &= \{|\pi_1|, \ldots, |\pi_{n_k}|\}, \\ I_{k-1} &= \{|\pi_{n_k+1}|, \ldots, |\pi_{n_k+n_{k-1}}|\}, \\ &\vdots \\ I_1 &= \{|\pi_{n_k+\ldots+n_2+1}|, \ldots, |\pi_{n_k+\ldots+n_1}|\} \end{aligned}$$

onde n_i é o tamanho do intervalo I_i. Os intervalos são ordenados por suas posições, mas são indexados em ordem decrescente no sentido do início para o fim da permutação.

Dizemos que uma sequência ótima de reversões $r = \rho_1\rho_2\ldots\rho_d$ produz um *k-strata* $B = (I_k, I_{k-1}, \ldots, I_1)$ em π se r possui uma subsequência $b = \theta_1\theta_2\ldots\theta_k$, tal que para $1 \leq i \leq k$, a reversão θ_i contém o intervalo I_i e, para qualquer $j > i$, não existe elemento I_j que esteja em θ_i.

Adicionalmente, para qualquer duas reversões consecutivas θ_i e θ_{i+1} de b, se ρ é uma reversão que ocorre entre θ_i e θ_{i+1} em r, então ρ é um subconjunto de $I_1 \cup I_2 \ldots \cup I_i$. As reversões contidas em b são chamadas **grandes reversões** (cada grande reversão produz

um novo *stratum*) e as reversões não contidas em b são chamadas **pequenas reversões**. Uma sequência de reversões que produz um *k-strata* possui k grandes reversões e $d - k$ pequenas reversões.

Se T é um *trace* de uma sequência ótima de reversões para π, definimos *B-induced subtrace* T_B um subconjunto de T tal que $T_B = \{s | s \in T$ e s produz o *k-strata* de B em $\pi\}$.

Baseado no conceito acima, uma pequena modificação foi feita no algoritmo para que, dado um *k-strata* B, a saída seja o conjunto de soluções que produzem B.

A modificação foi feita no primeiro passo onde o algoritmo de Siepel é utilizado. Após realizar a busca por todas as próximas reversões, apenas aquelas que são compatíveis com o *k-strata* são escolhidas. A primeira reversão é fixada e corresponde ao primeiro *stratum* (ou primeira grande reversão). A seguir, para cada passo, suponha que o *stratum* I_k foi movido por uma grande reversão e que o *stratum* I_{k+1} não foi movido. Aqui é possível escolher entre realizar uma grande reversão incluindo I_{k+1} e nenhum elemento dos *strata* seguintes, ou realizar uma pequena reversão, contida em $I_1 \cup \ldots \cup I_k$. Esta modificação não alterou a complexidade total do algoritmo.

O processo de construção de *traces* não foi alterado. Contudo, como consequência do processo de seleção de reversões a serem aplicadas, o algoritmo constrói de fato os *B-induced subtraces*. No final da execução, temos como saída um conjunto não vazio de *B-induced subtraces* (representados em suas formas normais) e seus tamanhos para uma dada permutação e um *k-strata* B.

Análises foram feitas com os dados dos genes compartilhados pelos últimos três *strata* dos cromossomos humanos X e Y (*strata* 3, 4 e 5) que resultam na permutação $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$.

O conjunto completo de soluções ótimas pode ser representado por seis *traces* que agrupam, no total, 31752 soluções.

Se considerarmos o cenário de ordenação que respeita a formação dos três últimos *strata*, $B = (\{3, 4\}, \{5, 6, 7, 8, 9, 10, 11, 12\}, \{1, 2\})$, a saída produzida lista um único *B-induced subtraces* que agrupa 420 soluções.

Aqui, mais importante do que a redução de 31752 para 420 soluções, é a obtenção de um único *B-induced subtraces*. Isto indica que o cenário observado, considerando-se os três *strata*, tem grande probabilidade de ser correto.

Apesar deste resultado, os limites entre os *strata* pode não ser claro e eventualmente, outras hipóteses podem ser avaliadas pelo algoritmo. Por exemplo, suponha um cenário que possui dois *strata*: $B' = (\{3, 4\}, \{1, 2, 5, 6, 7, 8, 9, 10, 11, 12\})$. A execução do algoritmo produz novamente um único *B-induced subtraces* que agrupa 2520 soluções diferentes.

Finalmente, o artigo conclui levantando como problema o limite relacionado ao uso de memória do algoritmo e a validade de usar alguns conceitos para impor restrições que ajudem a diminuir o conjunto de soluções.

A.3 Alignment with Non-overlapping Inversions in $O(n^3)$-Time [67]

O objetivo deste artigo é apresentar um algoritmo para a realização de alinhamentos com inversões que não se sobrepõem em tempo $O(n^3)$.

Alinhamentos de sequências são frequentemente usados para comparação de sequências biológicas. Eles são associados a um conjunto de operações de edição que transformam um sequência na outra.

Normalmente, as operações consideradas são *substituição* (mutação) de um símbolo por outro, *inserção* de um símbolo e *remoção* de um símbolo. Se associarmos pesos para estas operações, podemos utilizar um algoritmo de programação dinâmica clássico de complexidade $O(n^2)$ que calcula o conjunto de operações de edição de custo mínimo e exibe o alinhamento associado.

Outros importantes eventos biológicos como as inversões não são detectadas por estes algoritmos. Nós podemos definir uma nova operação de edição chamada inversão que substitui qualquer segmento por seu complemento reverso e, assim, definir um novo problema no estudo de alinhamentos: dados duas sequências e pesos fixos para cada tipo de operação de edição, o alinhamento com inversões é um problema de otimização que busca o custo mínimo total de uma sequência de operações de edição, incluindo inversões, que transforma uma sequência em outra. Além disso, é esperado também que as sequências de edições e/ou o alinhamento correspondente sejam obtidos. Neste trabalho, os autores reduzem o problema de otimização, descrito acima, no problema de maximização da pontuação de similaridade existente entre as sequências.

Em 1992, Schöniger e Waterman [64] descrevem uma simplificação ao problema exigindo que todas as regiões envolvidas nas operações de inversões não se sobreponham. Esta simplificação levou ao problema de *alinhamento com inversões que não se sobrepõem* e estes autores apresentaram um algoritmo simples de programação dinâmica com complexidade $O(n^6)$. Eles também introduziram algumas heurísticas que reduziam o tempo de execução médio para algo entre $O(n^2)$ e $O(n^4)$.

Contudo, Jiang *et al.* [21] mostram que o problema de decisão associado ao alinhamento com inversões para um alfabeto de tamanho ilimitado é NP-difícil.

Outros estudos independentes [28–30, 36] produziram algoritmos exatos para este problema com complexidade $O(n^4)$ de tempo e $O(n^2)$ de espaço. Os autores deste artigo apresentaram anteriormente [1] um algoritmo com complexidade $O(n^3 \log n)$ de tempo e, agora, apresentam um algoritmo com complexidade $O(n^3)$ de tempo e $O(n^2)$ de espaço.

O alinhamento de duas sequências s e t, utilizando o algoritmo tradicional de programação dinâmica, é representado pela inserção de buracos $(-)$ em determinadas posições de cada sequência e pelo alinhamento de cada símbolo ou espaço de s com o seu correspondente em t. Se $s[i]$ e $t[j]$ são símbolos de s e t, respectivamente, então um par $(s[i], t[j])$ é uma substituição (*mismatch*) de $s[i]$ por $t[j]$ (se ambos forem diferentes, caso contrário temos um *match*). Um par $(-, t[j])$ é uma inserção de um símbolo $t[j]$ em t. Finalmente, um par $(s[i], -)$ representa a remoção de $s[i]$ em s.

Além destas três operações, agora uma quarta operação é utilizada. A operação de *inversão* representa o evento em que uma subsequência é removida, invertida, complementada e inserida de volta na mesma posição. No caso deste estudo, existe a restrição que impede a sobreposição das inversões. Isso significa, que apesar de ser possível a ocorrência de múltiplas inversões em s, qualquer símbolo de s pode estar contido no máximo em uma região invertida.

Segundo a notação utilizada neste trabalho, $\overline{s}$ representa a sequência invertida de s e $\overline{s[a..b]}$ é a subsequência invertida de s que começa na posição a e termina na posição b.

Note que, a e b são tomadas de s e não de $\overline{s}$

Sejam s e t duas sequências de comprimentos n e m respectivamente. Considere $V = \{(i,j)|0 \leq i \leq n, 0 \leq j \leq m\}$ e $E = E_H \cup E_D \cup E_V$, onde:

- $E_H = \{e_H^{i,j} = ((i,j-1),(i,j))|0 \leq i \leq n, 0 \leq j \leq m\}$ é o conjunto de arestas horizontais que terminam no vértice (i,j);
- $E_D = \{e_D^{i,j} = ((i-1,j-1),(i,j))|0 \leq i \leq n, 0 \leq j \leq m\}$ é o conjunto de arestas diagonais que terminam no vértice (i,j);
- $E_V = \{e_V^{i,j} = ((i-1,j),(i,j))|0 \leq i \leq n, 0 \leq j \leq m\}$ é o conjunto de arestas verticais que terminam no vértice (i,j).

Considere também a função $\omega \colon E \longrightarrow \mathbb{R} \cup \{-\infty\}$ que associa a cada aresta $e \in E$ um peso $\omega(e)$. O grafo direcionado $G = (V, E, \omega)$ é o grafo de edição de s e t.

O peso $\omega(e_V^{i,j})$ indica a pontuação da remoção da letra $s[i]$ quando $s[1..i-1]$ é alinhado com $t[1..j]$. O peso $\omega(e_H^{i,j})$ aponta a pontuação da inserção da letra $t[j]$ quando $s[1..i]$ é alinhado com $t[i..j-1]$. Finalmente, o peso $\omega(e_D^{i,j})$ nos dá a pontuação da substituição da letra $s[i]$ pela letra $t[j]$ quando $s[1..i-1]$ é alinhado com $t[1..j-1]$. Os pesos são definidos normalmente por uma função $\phi \colon \Sigma \cup \{-\} \times \Sigma \cup \{-\} \longrightarrow \mathbb{R} \cup \{-\infty\}$, $- \notin \Sigma$, tais que $\omega(e_V^{i,j}) = \phi(s[i], -)$, $\omega(e_H^{i,j}) = \phi(-, t[j])$ e $\omega(e_V^{i,j}) = \phi(s[i], t[j])$, onde Σ é o alfabeto de símbolos usados nas sequências.

Um caminho que vá de $(0,0)$ até (i,j) em G corresponde a um e somente um alinhamento de $s[1..i]$ com $t[1..j]$ quando utilizamos o algoritmo tradicional (sem inversões) de programação dinâmica. A pontuação do alinhamento é dada pela soma dos pesos das arestas contidas no caminho.

Para este estudo, foi definido uma versão estendida do grafo de edição. Nesta versão, $E = E_H \cup E_D \cup E_V \cup E_X$ onde $E_X = \bigcup_{i=0}^{n} \bigcup_{j=0}^{m} E_X^{i,j}$ e $E_X^{i,j}$ é o conjunto de arestas estendidas que terminam no vértice (i,j), definido da seguinte maneira:

- $E_X^{i,j} = \{e_{i',j'}^{i,j} = ((i',j'),(i,j)|0 \leq i' \leq i \leq n, 0 \leq j' \leq j \leq m \text{ e } (i',j') \neq (i,j)\}$

A função ω também é estendida para que associe pesos às arestas estendidas, que representam alinhamentos tradicionais ótimos de subsequências de t com subsequências invertidas de s. Da mesma maneira que nos grafos de edição, um caminho ótimo no grafo de edição estendido é um caminho com peso máximo.

Seja $\overline{G} = (V, \overline{E}, \overline{\omega})$ o grafo de edição de $\overline{s}$ e t. Este grafo é utilizado para avaliação dos alinhamentos de subsequências de t contra subsequências invertidas de s. Em $\overline{G}$, os pesos $\overline{\omega}(e_H^{i,j})$, $\overline{\omega}(e_D^{i,j})$ e $\overline{\omega}(e_V^{i,j})$ correspondem, respectivamente, as pontuações de inserção de $t[j]$, substituição de $\overline{s}[i] = \overline{s[n+1-i]}$ por $t[j]$ e remoção de $\overline{s}[i] = \overline{s[n+1-i]}$.

Seja $G = (V, E, \omega)$ o grafo de edição estendido de s e t, tal que

- $\omega(e_H^{i,j}) =$ pontuação de inserção de $t[j]$,
- $\omega(e_V^{i,j}) =$ pontuação de remoção de $s[i]$,
- $\omega(e_D^{i,j}) =$ pontuação de substituição de $s[i]$ por $t[j]$,
- $\omega(e_{i',j'}^{i,j}) = \overline{\omega}_{(n-i,j')}^{(n-i',j)} + \omega_{inv}$,

onde ω_{inv} é o valor da penalidade para inversões e $\overline{\omega}^{(n-i',j)}_{(n-i,j')}$ é o peso de um caminho ótimo de $(n-i,j')$ até $(n-i',j)$ em $\overline{G}$. Em outras palavras, $\overline{\omega}^{(n-i',j)}_{(n-i,j')}$ é a pontuação do alinhamento tradicional entre $\overline{s[i'+1..i]}$ e $t[j'+1..j]$.

Como existe uma relação um para um entre caminhos em G e alinhamentos com inversões sem sobreposições de s com t, o peso de um caminho ótimo de $(0,0)$ até (n,m) em G é a pontuação de um alinhamento ótimo com inversões, que não se sobrepõem, de s e t.

Baseado nisso, a matriz B é definida de maneira que $B[i,j] = \omega^{i,j}_{0,0}$ é o peso de um caminho ótimo de $(0,0)$ até (i,j) em G, $0 \leq i \leq n$ e $0 \leq j \leq m$.

Dados i' e i, tais que $0 \leq i' \leq i \leq n$, a matriz $Out^i_{i'}[1..m, 1..m]$ de G é definida da seguinte maneira: $Out^i_{i'}[j',j] = B[i',j'] + \omega^{i,j}_{i',j'}$, se $0 \leq j' \leq j \leq m$, e $Out^i_{i'}[j',j] = -\infty$, se $0 \leq j < j' \leq m$. Nesta matriz, o elemento $Out^i_{i'}[j',j]$ armazena a pontuação do alinhamento ótimo de $s[1..i]$ contra $t[1..j]$ onde $\overline{s[i'+1..i]}$ é alinhado com $t[j'+1..j]$.

Seja G um grafo de edição. Dados i' e o vértice (i,j) de G, $0 \leq i' \leq i$, $hDif^{i,j}_{i'}$ de G é um vetor de tamanho j em que $hDif^{i,j}_{i'}[j'] = \omega^{i,j}_{i',j'} - \omega^{i,j-1}_{i',j'}$, $0 \leq j' < j$. Este vetor é não decrescente, propriedade que é utilizada pelo algoritmo.

O número de vezes que $hDif^{i,j}_{i'}[j']$ é incrementado quando $hDif^{i,j}_{i'}$ é percorrido de $j' = 0$ até $j-1$ é denominado $\Psi H^{i,j}_{i'}$.

Normalmente, o esquema de pontuação adotado no alinhamento utiliza valores inteiros: r para premiar uma igualdade de símbolos, q para penalizar uma diferença de símbolos e E para um buraco. Usando a notação do grafo de edição, $\omega(e^{i,j}_D) = r$ se $s[i] = t[j]$, $\omega(e^{i,j}_D) = q$ se $s[i] \neq t[j]$ e $\omega(e^{i,j}_H) = \omega(e^{i,j}_V) = E$, para todo par (i,j). Geralmente, $2E \leq q < r$ e, nesses casos, $\Psi H^{i,j}_{i'} \leq r - 2E$. Portanto, $\Psi H^{i,j}_{i'}$ é limitado por uma constante.

Dados i' e i, $0 \leq i' \leq i \leq n$, a coluna j da matriz $BLH^i_{i'}$, $0 \leq j \leq m$, é um vetor de tamanho $\Psi H^{i,j}_{i'}$, tal que, $BLH^i_{i'}[\alpha, j]$ é o α-ésimo j' onde $hDif^{i,j}_{i'}[j'] \neq hDif^{i,j}_{i'}[j'-1]$, para j' de 1 até $j-1$. Os elementos desta matriz são chamados de pontos de borda [52].

A função $buildBLH(\overline{G}, BLH, i')$ constrói a matriz $BLH^i_{i'}$ e é implementada segundo algoritmo previamente descrito [63, Seção 6]. Esta função executa em tempo $O(m)$. Como cada coluna da matriz de pontos de borda possui $O(1)$ elementos, esta função constrói cada coluna $BLH^i_{i'}$ baseada na respectiva coluna da matriz $BLH^i_{i'+1}$ em tempo constante.

A função $getMaxOut(BLH, B, i')$ retorna um vetor com o máximo valor de cada coluna de $Out^i_{i'}$ em tempo $O(m)$. Ela foi desenvolvida de acordo com algoritmo previamente descrito [52, Seção 6.2].

Os autores afirmam que, com base na utilização destas funções, é possível verificar que o algoritmo é correto e executa em tempo $O(n^2 m)$ ($O(n^3)$ se $m = O(n)$).

Testes foram realizados para verificar o desempenho do algoritmo.

O primeiro teste utilizou dois pares de sequências dos cromossomos 7 e 6 do homem e do chimpanzé, respectivamente. O primeiro par possuía sequências de 867 bp e o algoritmo reportou como resultado 98,6% de identidade e uma inversão envolvendo os trechos $95119414 - 95120280$ do cromossomo 7 e $96726825 - 96727087$ do cromossomo 6.

O segundo par era formado por sequências de 95319 bp. Para que o algoritmo pudesse processá-lo de maneira mais rápida, o seguinte procedimento foi adotado: as sequências foram quebradas em fragmentos de 100 bp cada. Estes fragmentos foram submetidos ao alinhamento tradicional de modo que todo fragmento de genoma humano fosse alinhado com todo fragmento de genoma do chimpanzé nos sentidos invertido e não invertido. O

Algoritmo A.1: Algoritmo $O(n^3)$ que constrói a matriz B

Entrada: Sequências s e t
Saída: Matriz B

```
1  início
2      para i ← 0 até |s| faça
3          /* Obtém caminho ótimo terminado com arestas não estendidas */
4          se i = 0 então
5              B[0,0] ← 0
6          senão
7              B[0,0] ← B[i−1,0] + ω(e_V^{i,j})
8          fim
9          para j ← 1 até |t| faça
10             se i = 0 então
11                 B[0,j] ← B[0,j−1] + ω(e_H^{i,j})
12             senão
13                 aux ← max(B[i,j−1] + ω(e_H^{i,j}), B[i−1,j] + ω(e_V^{i,j}))
14                 B[i,j] ← max(aux, B[i−1,j−1] + ω(e_D^{i,j}))
15             fim
16         fim
17         /* Obtém caminho ótimo terminado com arestas estendidas */
18         para i' ← i até 0 faça
19             BLH ← buildBLH(Ḡ, BLH, i')
20             maxOut_{i'}^{i} ← getMaxOut(BLH, B, i')
21             para j ← 0 até |t| faça
22                 B[i,j] ← max(B[i,j], maxOut_{i'}^{i}[j] + ω_inv)
23             fim
24         fim
25     fim
26     retorna B
27 fim
```

algoritmo foi utilizado considerando as sequências como sequências de fragmentos, ao invés de sequências de pares de bases. Uma correspondência entre fragmentos ocorria quando o alinhamento tradicional correspondente apresentava pontuação maior que um valor de corte. O algoritmo mostrou $94,8\%$ de identidade e uma inversão envolvendo os trechos $80553522 - 80588821$ do cromossomo humano 7 e $81781455 - 81816854$ do cromossomo 6 do chimpanzé.

O algoritmo também foi testado com dados simulados a partir de sequências de DNA geradas aleatoriamente. As sequências, com tamanho médio de 700 bp, foram criadas de modo que os pares apresentassem porcentagem de *indels* entre 5% e 10%, porcentagem de substituições entre 5% e 15%, e número de inversões sem sobreposição variando no intervalo entre 1 e 15. Os autores dizem que os resultados foram consistentes e que todas as inversões foram detectadas como esperado.

Finalmente, foram implementados em Java, para realização de testes comparativos, os algoritmos $O(n^3 \log n)$ e $O(n^4)$ descritos em trabalhos desenvolvidos anteriormente pelos autores [1, 29]. Também foi implementado o algoritmo esparso descrito em [30] que tem complexidade $O(r^2 \log^2 r)$, onde $r = O(n^2)$ é o número de igualdades entre símbolos de uma sequência contra símbolos de outra sequência. Os testes mostraram que o algoritmo desenvolvido neste trabalho é mais rápido que todos os outros algoritmos. O algoritmo 1 perde apenas do algoritmo esparso em casos onde são alinhadas sequências de fragmentos com número de correspondências baixo.

A.4 On the Properties of Sequences of Reversals that Sort a Signed Permutation [9]

A distância de reversão é o número mínimo de operações de reversões necessárias para transformar uma permutação orientada, que representa o genoma de uma espécie, em uma outra permutação associada a uma segunda espécie. Ela é amplamente aceita como uma boa estimativa da distância evolucionária entre duas espécies.

Os algoritmos relacionados a este conceito trabalham com dois problemas principais. O primeiro refere-se ao cálculo da distância de reversão. O segundo, mais complexo, visa a obtenção de uma sequência de reversões que realize a ordenação das permutações de acordo com a distância de reversão. O objetivo aqui, é fornecer aos biólogos uma sequência de eventos que apontem uma pista sobre a história evolucionária existente entre as duas espécies.

O objetivo deste artigo é fornecer evidências experimentais e teóricas em relação a enorme quantidade de soluções ótimas associadas a uma dada permutação. Além disso, o artigo trabalha argumentos que justifiquem a utilização de restrições biológicas secundárias para que os algoritmos possam apontar com maior probabilidade a real história evolucionárias existente entre as espécies.

Para prover base algorítmica adequada para o estudo de ordenações por reversões utilizando restrições biológicas, o trabalho também analisa a estrutura e as propriedades das sequências ótimas utilizando o conceito de *trace monoids* [20].

A primeira etapa da pesquisa abordada neste artigo procurou gerar, utilizando diferentes estratégias, diversos exemplos de sequências ótimas. O objetivo aqui é obter um entendimento da estrutura e das propriedades do enorme conjunto de sequências ótimas.

As diferentes estratégias para construção de sequências ótimas foram exploradas com o pressuposto de que existe ao menos uma sequência orientada de reversões que ordena uma permutação chamada "permutação sem obstáculos" (*without hurdles*). Esta restrição inicial exclui um subconjunto de permutações, difíceis de ordenar, que aparecem com uma frequência muito baixa, tanto em permutações aleatórias como em permutações associadas com dados biológicos. Permutações que não possuem elementos negativos são exemplos destas permutações difíceis de ordenar. Note que uma permutação sem obstáculos sempre possui ao menos uma reversão orientada.

Mesmo que a permutação não possua obstáculos, certas escolhas de reversões podem criar uma permutação difícil de ordenar. Assim, os autores propõem a seguinte definição: seja π uma permutação sem obstáculos. Uma reversão r é segura se r é uma reversão orientada de π tal que $r \cdot \pi$ é também uma permutação sem obstáculos.

Evidências experimentais, apresentadas no trabalho de Adam Siepel [66], mostram que, tipicamente, quase todas as reversões orientadas são seguras e que diversas restrições secundárias podem ser utilizadas para gerar sequências ótimas com características particulares.

Assim, numa primeira série de experimentos realizadas com permutações aleatórias, os autores avaliaram a taxa de sucesso de várias estratégias para seleção de reversões seguras. As estratégias selecionadas foram:

1. *Pontuação máxima*: Esta estratégia escolhe uma reversão orientada que maximiza o número de reversões orientadas na permutação resultante. Para permutações sem obstáculos, esta estratégia sempre encontra uma sequência ótima [8], e fornece um modo de identificar permutações sem obstáculos.
2. *Comprimento máximo*: Esta estratégia escolhe uma reversão orientada que maximiza o número de elementos revertidos.
3. *Comprimento mínimo*: Esta estratégia escolhe uma reversão orientada que minimiza o número de elementos revertidos.
4. *Aleatório*: Esta estratégia escolhe uma reversão orientada aleatoriamente.
5. *Pontuação mínima*: Esta estratégia escolhe uma reversão orientada que minimiza o número de reversões orientadas na permutação resultante. Segundo os autores, do ponto de vista teórico, esta estratégia deve ser uma das piores.

Os autores afirmam que ficaram surpresos em observar que a maioria das estratégias funcionam bem. A Tabela 4 apresenta as taxas de sucessos obtidas para um conjunto de 10.000 permutações orientadas aleatórias de 128 elementos. As estratégias foram aplicadas em cada uma das 10.000 permutações e o processamento parava quando a permutação resultante continha apenas elementos positivos. O sucesso era considerado quando a permutação resultante era a identidade.

A taxa sucesso de 100% para a estratégia de pontuação máxima é consequência da sua eficiência na ausência de obstáculos e do fato de que permutações com obstáculos são quase inexistentes em permutações com 128 elementos [18].

O valor médio de $d(\pi)$ obtido para o conjunto de 10.000 permutações foi de $124,33$. Baseados neste valor, os autores afirmam que as taxas obtidas, para as demais estratégias, são maiores do que as esperadas. Como cada experimento envolve a escolha, em média, de

Estratégia	Taxa de sucesso (em %)
Pontuação máxima	100
Comprimento máximo	79
Comprimento mínimo	64
Aleatório	62
Pontuação mínima	12

Tabela 4: Taxa de sucesso das estratégias (em porcentagem)

124 reversões, a escolha de uma única reversão não segura pode arruinar a sequência de ordenação.

Para explicar o sucesso obtido pelas estratégias, os autores apresentam uma análise teórica que resulta no seguinte teorema:

Teorema A.1. *Se π é uma permutação aleatória de n elementos, e se r é uma reversão orientada aleatória de π, então a probabilidade de r ser uma reversão não segura é $O(1/n^2)$.*

Após a apresentação deste teorema, o artigo introduz o conceito de *trace monoids*.

Sejam A um alfabeto e $\theta \subseteq A \times A$ uma relação simétrica e irreflexiva em A. Quando um par (x, y) está em θ, dizemos que as letras x e y comutam. Duas palavras w e w' sobre o alfabeto A são equivalentes com respeito à relação θ se podemos transformar w em w' através da aplicação de uma sequência finita de comutações. Por exemplo, se $A = \{a, b, c\}$, $\theta = \{(a, b), (a, c)\}$, as palavras *abacbbca* e *bcababac* são equivalentes pois podemos transformar uma em outra através das seguintes comutações.

$$\underline{ab}acbbca \rightarrow ba\underline{ac}bbca \rightarrow b\underline{ac}abbca \rightarrow bcaabb\underline{ca} \rightarrow bca\underline{ab}bac \rightarrow bcababac$$

O *trace monoid* T, associado a A e θ, é o conjunto de classes de equivalência de palavras sobre A com respeito à relação θ. Tais classes de equivalência são denominadas *traces*. Por exemplo, se $A = \{a, b, c\}$ e $\theta = \{(a, b), (a, c)\}$, os conjuntos de palavras $\{aabc, abac, baac, abca, baca, bcaa\}$ e $\{bc\}$ são dois *traces*.

O conceito de *trace monoids* foi introduzido por Cartier e Foata [20] com o nome de *monoids* parcialmente comutativos, relacionados a problemas de combinatória. Posteriormente, mostrou-se que o conceito possuía conexões com teoria de linguagens formais e com sistemas concorrentes.

Utilizando o conceito apresentado acima, os autores trabalham de forma a mostrar que o conjunto de sequências de reversões ótimas que ordenam uma permutação podem ser descritas em termos de *traces*.

Uma reversão pode ser identificada pelo conjunto de elementos que estão sendo invertidos na permutação. A sequência de reversões é, portanto, uma palavra sobre o alfabeto $\mathcal{P}_n$ de todos os subconjuntos não vazios de $\{1, \ldots, n\}$. Duas letras S e T deste alfabeto comutam se uma destas condições são observadas:

1. $S \cap T = \emptyset$
2. $S \subset T$ ou $T \subset S$

Denotamos por T_n o *trace monoid* associado ao alfabeto $\mathcal{P}_n$ e à relação de comutação descrita acima.

Proposição A.1. *Seja π uma permutação orientada. O conjunto de sequências ótimas de reversões que ordenam π é a união de* traces *pertencentes ao* trace monoid T_n.

Prova Precisamos mostrar que se uma sequência $R = r_1 r_2 \dots r_d$ de reversões ordena π, então todas as sequências no *trace* de R também ordenam a permutação. Sejam S e T os conjuntos correspondentes a duas reversões consecutivas em R, e π' a permutação obtida antes da aplicação de T.

Se $S \cap T = \emptyset$, então as duas reversões afetam intervalos disjuntos na permutação π'. Assim, a troca de ordens de S e T produz uma sequência de reversões bem definida que ordena π'.

Se $S \subset T$, como T é bem definido em π' e S é bem definido em $T\pi'$, os elementos de S devem ser também consecutivos em π', logo $TS\pi'$ é bem definida. O conjunto T pode ser descrito como três intervalos consecutivos ASB de π', e ambos $ST\pi'$ e $TS\pi'$ transformam estes intervalos em $-BS - A$. □

Entre os conceitos fundamentais relacionados aos *trace monoids* está a *forma normal de um trace*. Como um *trace* é um conjunto de palavras (finito, mas possivelmente enorme), seria interessante obter uma forma única e canônica que o caracterizasse completamente. Assim, o conceito de forma normal consiste na definição de uma propriedade de palavras que é verificada por exatamente uma palavra em todo *trace*.

Sejam A um alfabeto, θ um conjunto de regras de comutação sobre A e w uma palavra de comprimento n em A. Uma palavra w está na *Forma Normal de Cartier-Foata* se w pode ser decomposta em subpalavras consecutivas, $u_1|\dots|u_m$, tais que:

- todo par (x, y) de letras em u_i comutam, para todo $i \in [1..m]$;
- para $i > 1$, e para toda letra x em u_i, existe ao menos uma letra y em u_{i-1} tal que x e y não comutam;
- u_i é uma palavra crescente não vazia na ordem lexicográfica induzida por A.

Teorema A.2 (Cartier-Foata [20])**.** *Seja T um* trace monoid. *Para todo* trace $t \in T$, *existe uma única palavra em t que está na Forma Normal de Cartier-Foata.*

O número de subpalavras em Forma Normal de Cartier-Foata de uma palavra é denominado como sendo sua altura de pilha (*stack height*). Por exemplo, para a permutação $\pi = (0\ -4\ 1\ -3\ 6\ -7\ -5\ 2\ 8)$, temos que $d(\pi) = 6$, e que o conjunto de sequências de ordenação ótimas é composto pela união de 4 *traces* pertencentes a T_7:

$$\{2,5,6,7\}\{6,7\}\{7\}|\{1,2,3\}\{1,2,3,4\}\{2,3\}$$
$$\{2,5,7\}|\{1,2,3,6\}\{1,2,3,4,6\}\{2,3,6\}|\{3,4,5\}\{3,4,5,6\}$$
$$\{2,5,7\}|\{1,2,3,4,6\}|\{1,3,4,5\}\{1,3,4,5,6\}|\{1,2,3\}\{2,3\}$$
$$\{2,5,7\}|\{2,3,6\}|\{3,5\}\{3,5,6\}|\{1,2,3\}\{1,2,3,4\}$$

As alturas de pilha são, respectivamente 2, 3, 4 e 4. O maior *trace* é o primeiro que agrupa 180 elementos. O segundo *trace* tem tamanho 12 e os outros dois têm tamanho 4.

Após a apresentação destes conceitos os autores iniciam uma seção que discute uma série de questões e argumentações sobre a utilização de *traces* em aplicações biológicas.

Desta discussão, os autores concluem que os algoritmos associados à teoria de *trace monoids* são úteis para lidar com um *trace*. Contudo, na maioria dos casos, o conjunto de soluções ótimas é composto por um conjunto de *traces*.

Gerar o conjunto completo de sequências ótimas é muito custoso e até o momento da escrita do artigo, não existiam algoritmos que fossem capazes de gerar um elemento para cada *trace* pertencente ao conjunto de soluções ótimas de uma permutação. Assim, os autores deixam este como um assunto aberto para pesquisa.

Outro resultado citado refere-se à relação entre o valor da altura da pilha de um *trace*. Segundo o campo de computação distribuída, este parâmetro está fortemente associado à possibilidade de um programa ser paralelizado [60]. Se considerarmos as letras do alfabeto A como tarefas que devem ser processadas por um sistema paralelo e o conjunto de regras de comutação como o conjunto de pares de tarefas que podem ser processadas em paralelo, um *trace* é a descrição de um programa para este sistema. Neste contexo, a subpalavra u_i na Forma Normal de Cartier-Foata é o conjunto de tarefas que podem ser processadas ao mesmo tempo pelo sistema. A subpalavra u_{i+1} é o conjunto de tarefas que podem ser processadas uma vez que todas as tarefas de u_i foram concluídas. A taxa entre o tamanho de uma palavra e a sua altura de pilha é chamada grau de concorrência e é considerada por muitos autores como uma boa medida da habilidade do programa correspondente ser paralelizado.

Os autores estudaram um panorama geral da altura de pilha dos *traces* para verificar se cenários evolucionários altamente paralelizáveis são plausíveis ou não. Assim, uma nova estratégia de escolha de reversões foi criada de modo que ela "promovesse" a comutação:

- *Comutação*: Esta estratégia escolhe, entre todas as reversões orientadas definidas em π, o subconjunto máximo no qual todas as reversões comutam entre si. Todas as reversões deste subconjunto são aplicadas em π. O processo é repetido até que a permutação resultante seja positiva.

A Tabela 5 compara as alturas de pilhas média obtidas a partir da ordenação de 10.000 permutações aleatórias, de tamanhos entre 32 e 224, obtidas de acordo com as estratégias *Comprimento máximo*, *Pontuação máxima* e *Comutação*.

Note que a estratégia *Comprimento máximo* tem um impacto muito negativo em relação à comutação de elementos. Isso significa que ela conduz a histórias evolucionárias quase determinísticas. Por outro lado, a estratégia que promove comutação reduz drasticamente os valores de altura de pilha. Ela conduz a um número de sequências equivalentes que cresce fatorialmente com uma função crescente do tamanho.

Finalmente, o artigo conclui que apesar da distância de reversão fornecer uma boa estimativa da distância entre duas espécies, o número de sequências ótimas correspondentes à distância é tipicamente muito grande para ser útil na determinação da história evolucionária real.

Enquanto alguns parâmetros possuem significado biológico claro como, por exemplo, reversões curtas *versus* reversões longas, outros parâmetros como a altura de pilha de um

n	*Comprimento máximo*	*Pontuação máxima*	*Comutação*
32	19,3	10,2	7,4
64	42,9	19,6	11,6
96	66,7	28,3	14,8
128	91,2	36,7	17,6
160	115,5	45,3	20,0
192	139,9	52,8	22,3
224	164,6	60,1	24,2

Tabela 5: Altura de pilha média segundo as estratégias *Comprimento máximo*, *Pontuação máxima* e *Comutação*.

trace não possui uma fácil interpretação evolucionária.

Como trabalhos futuros, os autores desejam identificar outros parâmetros combinatórios, relacionados ao conjunto de sequências ótimas, com foco especial sobre aqueles que os biólogos considerem como relevantes evolucionariamente. Eles apontam como um conceito interessante a utilização de segmentos conservados, que possuem ricas interações com o problema de ordenação por reversões.

A.5 An Algorithm to Enumerate Sorting Reversals for Signed Permutations [66]

O problema de encontrar todas as sequências de tamanho mínimo que ordenam uma permutação é uma generalização natural do problema de ordenação por reversões.

Este problema, que o autor denomina *ASR* (*all sorting reversals problem*), pode ser facilmente reduzido para o problema de encontrar todas as reversões de uma permutação que, quando aplicadas, diminuem a distância de permutação.

O objetivo deste artigo é apresentar um algoritmo que soluciona o problema de enumerar todas as reversões que ordenam uma permutação. Dados experimentais também são apresentados e mostram que, apesar de o algoritmo não apresentar ganhos em termos assintóticos, pois ele tem complexidade de tempo $O(n^3)$ para permutações de tamanho n, o algoritmo pode resolver o problema usando força bruta em tempo $\Theta(n^3)$ e, na prática, o desempenho é dramaticamente melhor do que a apresentada pela melhor alternativa conhecida.

O artigo traz em sua parte inicial uma longa introdução de conceitos e definições utilizadas no trabalho. Na sequência, o autor apresenta uma versão simplicada do problema denominada Modelo Livre de Fortalezas (*Fortress-Free Model – FFM*) Este modelo é utilizado para mostrar uma série de critérios que uma reversão de cada classe deve apresentar para ser uma reversão que ordena uma permutação. Feito isso, o ator reintroduz o conceito de fortalezas e os resultados obtidos para o *FFM* são generalizados para o caso geral.

Sejam π e ϕ duas permutações orientadas de tamanho n, tais que $\pi = (\pi_1, \pi_2 \ldots, \pi_n)$ e $\phi = (\phi_1, \phi_2 \ldots, \phi_n)$. Deixe a permutação não orientada $\pi' = (\pi'_0, \pi'_1 \ldots, \pi'_{2n+1})$ ser definida de modo que $\pi'_0 = 0$, $\pi'_{2n+1} = 2n + 1$ e para todo i $(1 \leq i \leq n)$, $\pi'_{2i} = 2\pi_i$, $\pi'_{2i-1} = 2\pi_i - 1$ se $\pi_i > 0$ ou $\pi'_{2i} = 2|\pi_i| - 1$, $\pi'_{2i-1} = 2|\pi_i|$ se $\pi_i < 0$. Deixe $\phi' = (\phi'_0, \phi'_1 \ldots, \phi'_{2n+1})$ ser

definida da mesma maneira com repeito à ϕ.

Dois elementos π_i e π_{i+1} são denominados *adjacentes* em π, e os elementos correspondentes π'_{2i} e π'_{2i+1} são nomeados *adjacentes* em π' (similarmente para ϕ e ϕ').

O grafo de *breakpoint* $\mathcal{B}$ de π com respeito a ϕ é construído através do arranjo em linha da sequência de $2n + 2$ vértices, correspondentes aos elementos de π' (os vértices devem obedecer a ordem de π') e da conexão de todo par de vértices que representa uma adjacência em π' com uma aresta preta e de todo par de vértices que representa uma adjacência em ϕ' com uma aresta cinza. O processo de se visitar os vértices de $\mathcal{B}$ na ordem em que aparecem é denominado como *percurso* do grafo de *breakpoint*. O percurso no grafo pode começar em qualquer vértice, mas deve incluir todos os vértices e pode ser feito em qualquer direção.

Um ciclo em $\mathcal{B}$ é uma sequência de vértices $(v_0, v_1, v_2, \ldots, v_{2m}, v_{2m+1}, v_{2m+2})$, tais que $m \geq 0$, $v_{2m+2} = v_0$ e, para todo $0 \leq i \leq m$, v_{2i} e v_{2i+1} estão conectados por uma aresta preta, e v_{2i+1} e v_{2i+2} estão conectados por uma aresta cinza. O percurso de um ciclo é feito visitando-se os vértices e as arestas na ordem desta sequência. Como no percurso de $\mathcal{B}$, o percurso de um ciclo pode começar em qualquer ponto e ser feito em qualquer direção.

Um dado percurso em um ciclo induz a toda aresta preta uma ordenação positiva ou negativa dos vértices de $\mathcal{B}$. Por exemplo, suponha que o ciclo contém os vértices 0, 10, 3, 1, 2 e 11 e que o percurso começa ao longo da aresta preta que liga os vértices 0 e 10. Neste caso, o percurso induz uma ordenação positiva dos vértices na aresta $(0, 10)$ e uma ordenação negativa na aresta $(11, 2)$. Em relação as arestas pretas e cinzas que conectam os vértices de um ciclo, dizemos que elas pertencem ao ciclo.

Seja o *grafo de sobreposição* $\mathcal{O} = (V, E)$ para $\mathcal{B}$ ser definido de modo que existe um vértice distinto $v_e \in V$ para toda aresta cinza e em $\mathcal{B}$. Dois vértices v_e e v'_e são conectados por uma aresta ($\{v_e, v'_e\} \in E$) se e somente se e e e' se sobrepõem em $\mathcal{B}$. Um *componente conexo* em $\mathcal{O}$ tem o significado usual e pode ser chamado simplesmente de *componente*.

Uma aresta preta é orientada se ela gera um número ímpar de vértices em $\mathcal{B}$ e não orientada, caso contrário. Um ciclo em $\mathcal{B}$ e um componente conexo em $\mathcal{O}$ são orientados se eles possuem pelo menos uma aresta cinza orientada. Ciclos e componentes são não orientados se eles não são orientados e não são triviais. Um ciclo trivial consiste de uma única aresta cinza e uma única aresta preta e correspondem a uma adjacência compartilhada pelas permutações π e ϕ. Um componente é trivial se ele consiste de um vértice único e isolado em $\mathcal{O}$.

As arestas cinzas de um ciclo sempre pertencem ao mesmo componente conexo e, por isso, podemos dizer que o ciclo pertence ao componente. Em alguns casos, é útil agrupar componentes triviais e orientados. O autor se refere a esses componentes como *componentes benignos*.

Todo componente orientado pode ser classificado como *obstáculo* ou um *não obstáculo protegido*. Um obstáculo é um componente não orientado que não separa outros componentes não orientados. Um não obstáculo protegido, por sua vez, separa outros componentes não orientados. Um componente u separa dois outros componentes v e w se, em um percurso de $\mathcal{B}$, é impossível passar de um vértice pertencente a v para um vértice pertencente a w sem encontrar um vértice de u. Note que, enquanto o conceito de separação é usado em relação a componentes não orientados, ele também se aplica a componentes benignos. Um obstáculo é chamado de *super obstáculo* se, onde ele for eliminado, um não obstáculo protegido ermegirá como um obstáculo. Caso contrário, este obstáculo é um *obstáculo*

simples.

Segundo o teorema da dualidade de Hannenhalli e Pevzner [42], a distância $d(\pi, \phi)$ entre π e ϕ é dada por $d(\pi, \phi) = n + 1 - c + h + f$, onde c é o número de ciclos e h é o número de obstáculos em $\mathcal{B}$. O parâmetro f é igual a 1 se existe uma *fortaleza* em $\mathcal{B}$ e igual a 0, caso contrário. Uma fortaleza existe se e somente se existe um número ímpar de obstáculos e todos eles são super obstáculos. A função $d(\pi, \phi)$ é, às vezes, representadas pela variável d quando π e ϕ são claros no contexto.

Uma reversão $\rho_{i,j}$ $(1 \leq i < j \leq n)$ aplicada a $\pi = (\pi_1, \ldots, \pi_n)$ transforma π em $\rho_{i,j}(\pi) = (\pi_1, \ldots, -\pi_j, \ldots, -\pi_i, \ldots, \pi_n)$. Uma reversão ρ é uma *sorting reversal* se e somente se $d(\rho(\pi), \phi) = d(\pi, \phi) - 1$. O autor utiliza o termo Δd para indicar a quantia $d(\rho(\pi), \phi) - d(\pi, \phi)$ e, similarmente, os termos Δc, Δh e Δf para indicar mudanças causadas por uma dada reversão aos números de ciclos, obstáculos e fortalezas ($\Delta d = -\Delta c + \Delta h + \Delta f$). Uma dada reversão pode apenas aumentar em 1 unidade, diminuir em 1 unidade ou deixar inalterada a distância de reversão de π para ϕ. Assim, $\Delta d \in \{-1, 0, 1\}$. As extremidade i e j de uma reversão $\rho_{i,j}$ correspondem às i-ésima e $j + i$-ésima arestas de $\mathcal{B}$. Dizemos que uma reversão ρ atua nestas arestas.

Setubal e Meidanis [65] apresentaram uma distinção alternativa àquela existente entre arestas cinzas orientadas e não orientadas, baseada em arestas pretas. Eles definem que duas arestas pretas que pertencem ao mesmo ciclo são *convergentes* se, em um percurso do ciclo, estas arestas induzem a mesma ordenação dos vértices de $\mathcal{B}$. Caso contrário, os vértices são *divergentes*. Facilmente pode ser mostrado que uma aresta cinza orientada sempre conecta nós que unem arestas pretas divergentes e que uma aresta cinza não orientada sempre conecta nós que unem arestas pretas convergentes (se a aresta cinza não orientada faz parte de um ciclo trivial, então ela conecta nós que unem a mesma aresta preta e, assim, consideramos que toda aresta preta é convergente com ela mesma). Uma reversão quebra um ciclo se e somente se ela atua em arestas pretas divergentes do ciclo. Este resultado é mais forte do que o correspondente para arestas cinzas, que diz que uma reversão quebra um ciclo se (e não, se e somente se) ela atua em arestas pretas correspondentes a uma aresta cinza orientada. Uma reversão não muda o número de ciclos se e somente se ela atua em arestas convergentes do mesmo ciclo. Uma reversão combina dois ciclos se e somente se ela atua em arestas pertencentes a diferente ciclos.

Quando uma aresta cinza possui um vértice dentro do intervalo da reversão e outro fora, dizemos que ela é *afetada* pela reversão. Um componente é afetado por uma reversão se e somente se ele possui ao menos uma aresta cinza que é afetada. É possível mostrar que uma reversão faz com que uma aresta mude de orientação (mudar de orientada para não orientada ou vice-versa) se e somente se ela afeta a aresta (duas arestas pretas unindo uma aresta cinza mudam de convergente para divergente ou vice-versa se e somente se a aresta cinza é afetada). Como resultado, uma reversão fará com que um componente orientado se transforme em não orientado, ou vice-versa, se ela afeta o componente. Esta observação fornece uma simples explicação para os fenômenos de corte de obstáculos (*hurdle cutting*) e união de obstáculos (*hurdle merging*) descritos por Hannenhalli e Pevzner [42].

O autor denota todos componentes conexos, os ciclos dentro deles e as arestas pretas dentro dos ciclos como descrito a seguir: Sejam $M = \{m_i\}$ o conjunto de componentes em $\mathcal{O}$, $C_i = \{c_{i,j}\}$ o conjunto de ciclos que pertence a m_i e $B_{i,j} = \{b_{i,j,k}\}$ o conjunto de arestas pretas pertencentes ao ciclo $c_{i,j}$. Esta notação permite que as definições acima

sejam descritas de maneira conscisa.

Quaisquer duas arestas pretas pertencentes ao mesmo ciclo, $b_{i,j,k}$ e $b_{i,j,l}$, podem ser convergentes ou divergentes. Se existem arestas preta divergentes $b_{i,j,k}$ e $b_{i,j,l}$, então o ciclo $c_{i,j}$ é orientado, caso contrário é não orientado, ao menos que ele possua uma única aresta preta ($|B_{i,j}| = 1$), caso em que $c_{i,j}$ é trivial. Se existe $c_{i,j} \in C_i$ tal que $c_{i,j}$ é orientado, então m_i é orientado. Caso contrário, m_i é não orientado a não ser que $|C_i| = 1$ e o único elemento de C_i é trivial, caso em que m_i é, também, trivial. Se m_i é não orientado, então m_i pode ser um obstáculo ou um não obstáculo protegido, dependendo da condição de ele separar ou não outros componentes não orientados.

Para a apresentação do modelo livre de fortalezas o autor apresenta uma série de lemas e definições que direcionam para uma classificação exaustiva de reversões.

Lema A.1. *Suponha que ρ é uma reversão que atua em duas arestas pretas (de um grafo de* breakpoint*) $b_{i,j,k}$ e $b_{i',j',k'}$ que pertencem, respectivamente, aos ciclos $c_{i,j}$ e $c_{i',j'}$ e componentes conexos m_i e $m_{i'}$. Então, uma e somente uma das seguintes afirmações é verdadeira:*

1. *($i = i'$ e $j = j'$) $b_{i,j,k}$ e $b_{i',j',k'}$ pertencem ao mesmo ciclo, $c_{i,j}$, e:*
   (a) *$c_{i,j}$ é orientado e m_i é orientado; ou*
   (b) *$c_{i,j}$ é não orientado e $b_{i,j,k}$ e $b_{i',j',k'}$ são convergentes, e m_i pode ser orientado ou não orientado.*
2. *($i = i'$ e $j \neq j'$) $b_{i,j,k}$ e $b_{i',j',k'}$ pertencem a ciclos diferentes do mesmo componente m_i, que pode ser orientado ou não orientado.*
3. *($i \neq i'$) $b_{i,j,k}$ e $b_{i',j',k'}$ pertencem a diferentes componentes m_i e $m_{i'}$.*

O modelo *FFM* do problema *ASR* é definido em termos de uma medida de distância modificada $d'(\pi, \phi)$, que ignora fortalezas. Especificamente, seja $d'(\pi, \phi) = n + 1 - c + h$ e $\Delta d' = d'(\rho(\pi), \phi) - d'(\pi, \phi)$. Então, o modelo *FFM* é definido como a versão do problema *ASR* onde reversões que ordenam (*sorting reversals*) são aquelas que produzem $\Delta d' = -1$.

Lema A.2 (Conservação da distância)**.** *Sob o modelo* FFM*, uma reversão é uma* sorting reversal *(reversão ordenadora) se e somente se uma das condições a seguir é observada:*

1. *$\Delta c = -1$ e $\Delta h = -2$;*
2. *$\Delta c = 0$ e $\Delta h = -1$;*
3. *$\Delta c = 1$ e $\Delta h = 0$.*

Considerandos o modelo *FFM* os próximos lemas discutem os casos 1a, 1b, 2 e 3 do Lema A.1.

Lema A.3 (Caso 1a)**.** *Sob o modelo* FFM*, uma reversão ρ atua em duas arestas pretas pertencentes ao mesmo ciclo orientado é uma* sorting reversal *se e somente se as arestas são divergentes e ρ não altera o número de obstáculos.*

Lema A.4 (Caso 1b)**.** *Sob o modelo* FFM*, uma reversão ρ que atua em duas arestas pretas pertencentes ao mesmo ciclo não orientado $c_{i,j}$ é uma* sorting reversal *se e somente o componente m_i, ao qual $c_{i,j}$ pertence, é um obstáculo simples.*

Quando reversão afeta um obstáculo da maneira como descrita no Lema A.4, dizemos que ela *corta* o obstáculo.

Lema A.5 (Caso 2)**.** *Sob o modelo* FFM*, uma reversão ρ não pode ser uma* sorting reversal *se ρ atua em duas arestas pretas pertencentes a dois ciclos diferentes de um mesmo componente.*

Definição A.1. *Dois obstáculos u e v formam um* super obstáculo duplo – SOD *se e somente se dois outros componentes orientados w e p coexistem com u e v tais que:*

1. *p separa u e v de w mas não separa u e v;*
2. *todo componente não orientado ou separa u e v ou é separado de u e v por p;*
3. *p não separa qualquer dois outros componentes que ele separa de u e v.*

Definição A.2. *Dizemos que um obstáculo que separa um componente benigno de outros componentes não orientados é o* obstáculo separador *do componente benigno.*

Lema A.6 (Caso 3)**.** *Sob o modelo* FFM*, uma reversão ρ que atua em duas arestas negras pertencentes a dois diferentes componentes u e v é uma* sorting reversal *se e somente se todas as condições abaixo são observadas:*

1. *cada um dos componentes u e v é um obstáculo ou é um componente benigno que possui um obstáculo separador;*
2. *u e v não são dois componentes benignos que compartilham o mesmo obstáculo separador;*
3. *u e v ou seus obstáculos separadores não formam um* SOD*.*

O caso geral difere do modelo *FFM* quando $\Delta f \neq 0$. Neste caso, duas possibilidades devem ser consideradas.

A primeira possibilidade, tratada pelo Lema A.7, ocorre quando uma fortaleza não existe antes da aplicação de uma reversão candidata ρ. Aqui, é possível que ρ seja uma *sorting reversal* sob o modelo *FFM*, mas introduz uma fortaleza e, por isso, não é uma *sorting reversal* no caso geral. O segundo caso reflete a situação contrária: ρ não é uma *sorting reversal* sob o modelo *FFM*, mas elimina uma fortaleza e, por isso, é uma *sorting reversal* no caso geral.

Lema A.7. *Um reversão ρ que obedece os critérios de* sorting reversal *sob o modelo* FFM *irá introduzir uma fortaleza se e somente se uma das seguinter condições for verdadeira:*

1. *ρ atua em arestas pretas divergentes do mesmo ciclo orientado, e ρ introduz ao menos um componente não orientado tal que o número de obstáculos se torna ímpar e todos eles são super obstáculos;*
2. *ρ corta o único obstáculo simples e existem um número ímpar de super obstáculos;*
3. *ρ atua sobre duas arestas pretas pertencentes a diferentes componentes tais que dois obstáculos são destruídos, e o conjunto de obstáculos é alterado de maneira que permaneça um número ímpar de elementos, todos super obstáculos.*

Para analisar a segunda possibilidade, precisamos introduzir algumas definições.

Definição A.3. *Dois componentes não orientados u e v são adjacentes em um grafo de* breakpoint $\mathcal{B}$ *se e somente se ao menos um par de vértices de u e v possui entre eles, quando o grafo de* breakpoint *é percorrido, nenhum vértice pertencente a outro componente não orientado.*

Definição A.4. *Seja U o conjunto de componentes não orientados em um grafo de breakpoint $\mathcal{B}$. O grafo de obstáculo de $\mathcal{B}$ é um grafo $\mathcal{H} = (V, E)$ tal que $V = U$ e $E = \{\{v_i, v_j\} | v_i, v_j \in V\}$ e v_i, v_j são adjacentes em $\mathcal{B}$.*

O grafo de obstáculos pode ser facilmente contruído percorrendo-se uma vez o grafo de *breakpoint*. Ele possui diversas propriedades úteis. Por exemplo, obstáculos, não obstáculos protegidos e super obstáculos podem ser identificados a partir de propriedades locais de vértices do grafo de obstáculos:

1. Suponha que u, v e w são três componentes não orientados e u', v' e w' são os vértices correspondentes no grafo de obstáculos. Então, w separa u e v se e somente se não existe caminho no grafo de obstáculos entre u' e v' que não passe por w'. Por conveniência, dizemos que o nó w' separa os nós u' e v'.
2. Um vértice no grafo de obstáculos não separa outros vértices se e somente se ele pertence a um ciclo e possui grau 2, ou ele tem um grau menor do que 2. Este vértice corresponde a um obstáculo.
3. Um vértice no grafo de obstáculos separa outros vértices se e somente se ele pertence a um ciclo e possui grau maior do que 2, ou se ele não pertence a um ciclo e tem grau 2. Este vértice corresponde a um não obstáculo protegido.
4. Um vértice corresponde a um super obstáculo se e somente se possui grau 1 e seu vizinho possui grau 3 e pertence a um ciclo, ou tem grau 2 e não pertence a um ciclo.

Existem diversas estruturas de alto-nível em um grafo de obstáculos, contudo para o problema, o mais importante é a identificação de *sorting reversals* no caso de uma fortaleza.

Definição A.5. *Uma* cadeia de obstáculo *é uma cadeia de vértices no grafo de obstáculos composto por um obstáculo e por nenhum ou mais outros vértices, tais que todo vértice v na cadeia é um obstáculo, tem grau* 2 *e não pertence a um ciclo, ou é o último vértice na cadeia e, ou pertence a um ciclo ou tem grau maior do que* 2.

Definição A.6. *Se uma cadeia de obstáculos tem uma extremidade que não é um obstáculo, o vértice nesta extremidade é a* âncora *da cadeia.*

Uma cadeia de obstáculos que possui obstáculos nas duas extremidades é uma *cadeia não ancorada* (esta cadeia deve ter exatamente dois obstáculos). Se uma cadeia não ancorada existe, ela deve [encompass] o grafo de obstáculos inteiro; contudo, se existe ao menos uma cadeia ancorada, todas as outras cadeias devem ser ancoradas.

Uma cadeia de obstáculos que possui um super obstáculo é uma cadeia de super obstáculo. Tal cadeia não pode ter nenhum obstáculo simples. Ela deve ser ancorada e ter um único super obstáculo ou não ser ancorada e ter dois super obstáculos.

SODs podem ser identificados rapidamente a partir do grafo de obstáculos usando os conceitos de cadeia de obstáculos e âncoras. Especificamente, dois obstáculos u e v formam um *SOD* se e somente se u e v pertencem a uma cadeia de obstáculos ancorada por vértices w e x, tais que w e x pertencem a um ciclo de três vértices, o terceiro vértice do ciclo tem grau 3 e ambos w e x tem grau 3 no máximo.

Definição A.7. *Um super obstáculo é um* protetor único *se ele pertence a uma cadeia de obstáculos ancorada de tamanho* 2*.*

Definição A.8. *O vizinho de um protetor único é um* pseudo-obstáculo*.*

Lema A.8. *Uma reversão ρ que corta um pseudo-obstáculo ou um protetor único não muda o número total de obstáculos, mas faz com que um super obstáculo seja substituído por um obstáculo simples.*

Lema A.9. *Uma reversão ρ eliminará uma fortaleza e será uma* sorting reversal *se e somente se existe uma fortaleza e se uma das seguintes condições é verdadeira:*

1. *ρ atua em arestas divergentes do mesmo ciclo e introduz ao menos um novo componente não orientado tal que o número de obstáculos é incrementado em exatamente mais* 1 *obstáculo;*
2. *ρ corta um protetor único ou um pseudo-obstáculo;*
3. *ρ afeta dois componentes u e v tais que um deles é um super obstáculo ou um componente benigno que possui um super obstáculo separador. Deixe u ser este componente, o obstáculo u' ser u ou o obstáculo separador de u, e k ser a cadeia de obstáculo de u'. Uma das afirmações a seguir deve ser verdadeira em relação a v:*
   (a) *v é a âncora da cadeia k;*
   (b) *v é um não obstáculo protegido não pertencente à cadeia k;*
   (c) *v é um componente benigno que não possui obstáculo separador, e nenhum componente na cadeia k separa v de outro componente na cadeia k;*
   (d) *v é um super obstáculo ou um componente benigno com um super obstáculo separador, e v ou seu super obstáculo separador formam um* SOD *com u'.*

Os Lemas A.7 e A.9 permitem que os Lemas A.3, A.4, A.5 e A.6 sejam generalizados para acomodar fortalezas.

Teorema A.3 (Generalização do Lema A.3)**.** *Uma reversão ρ que atua em duas arestas pretas pertencentes ao mesmo ciclo orientado é uma* sorting reversal *se e somente se as arestas são divergentes e uma das seguintes afirmações é verdadeira:*

1. *ρ não introduz um componente não orientado;*
2. *Não existem fortalezas ($f = 0$) e ρ introduz ao menos um componente não orientado, mas não muda o número de obstáculos e não gera a existência de um número ímpar de obstáculos onde todos são super obstáculos.*
3. *Existe uma fortaleza ($f = 1$) e ρ introduz ao menos um componente não orientado tal que o número de obstáculos é mantido o mesmo ou é incrementado em exatamente mais um obstáculo.*

Teorema A.4 (Generalização do Lema A.4)**.** *Uma reversão ρ que atua em duas arestas pretas pertencentes ao mesmo ciclo não orientado $c_{i,j}$ é uma* sorting reversal *se e somente se uma das seguintes condições é verdadeira sobre o componente m_i ao qual o ciclo $c_{i,j}$ pertence:*

1. *Não existe nenhuma fortaleza ($f = 0$), m_i é um obstáculo simples e ou m_i não é o único obstáculo simples ou existe um número par de super obstáculos.*
2. *Existe uma fortaleza ($f = 1$) e m_i é um protetor simples ou um pseudo-obstáculo.*

Teorema A.5 (Generalização do Lema A.5)**.** *Uma reversão ρ não pode ser uma* sorting reversal *se ρ atua sobre duas arestas pretas pertencentes a diferentes ciclos do mesmo componente.*

Teorema A.6 (Generalização do Lema A.6)**.** *Uma reversão ρ que atua sobre arestas pretas pertencentes a diferentes componentes u e v é uma* sorting reversal *se e somente se uma das duas condições abaixo é verdadeira:*

1. *Não existe fortaleza ($f = 0$) e todas as condições abaixo são verificadas:*
   (a) *Cada um dos componentes u e v é um obstáculo ou é um componente benigno que possui um obstáculo separador;*
   (b) *u e v não são componentes benignos que compartilham o mesmo obstáculo separador;*
   (c) *u e v ou seus obstáculos separadores não forma um* SOD*;*
   (d) *A eliminação de obstáculos associados com u e v não deixa um número ímpar de obstáculos que sejam super obstáculos.*
2. *Existe uma fortaleza ($f = 1$) e uma das duas condições abaixo são observadas:*
3. *Cada um dos componentes u e v é um super obstáculo ou é um componente benigno que possui um obstáculo separador, ou u e v não são componentes benignos compartilhando o mesmo obstáculo separador, ou u e v ou seus obstáculos separadores não formam um* SOD*;*
4. *O caso 3 do Lema A.6 se aplica.*

Os Teoremas A.3, A.4, A.5 e A.6 conduzem à descrição do algoritmo que resolve o problema *ASR* (`find_all_sorting_reversals`).

Um passo importante deste algoritmo é a rotina que enumera *sorting reversals* que quebram ciclos (`get_rev_split_cycles`). Este passo tem a função de determinar se uma reversão que quebra um ciclo introduz novos componentes não orientados. Como esta rotina deve ser aplicada para toda reversão candidata da classe que quebra ciclos e esta é a classe mais frequente, ela é extremamente custosa. O autor nomeou a rotina que efetivamente faz o trabalho como `detect`.

Duas versões de `detect` foram implementadas e comparadas em performance. A primeira versão implementa o algoritmo `connected_components` de Bader *et al.* [2], que testa se mais componentes não orientados estão presentes depois do que antes da reversão. Esta rotina gasta tempo linear no número de vértices afetados pelo componente.

A segunda versão baseou-se na extensão de idéias apresentadas por Bergeron [8] e Bergeron e Strasbourg [10] para modelagem de mudanças no grafo de sobreposição induzidas por uma reversão. A chave deste algoritmo está no Lema A.10.

Algoritmo A.2: Algoritmo `find_all_sorting_reversals`

Entrada: Duas permutações orientadas de tamanho n, π e ϕ.
Saída: Uma lista L de todas as reversões que ordenam π com respeito à ϕ.

início
  Construa o grafo de *breakpoint* $\mathcal{B}$ de π com respeito à ϕ;
  Identifique todas as arestas pretas, ciclos e componentes conexos em $\mathcal{B}$. Deixe $e_{i,j,k}$ representar a k-ésima aresta preta pertencente ao j-ésimo ciclo do i-ésimo componente;
  Defina o valor $o_{i,j,k}$ correspondente a cada $e_{i,j,k}$ tal que $o_{i,j,k} \in \{+1, -1\}$ e $o_{i,j,k_1} \cdot o_{i,j,k_2} = -1$ se e somente se e_{i,j,k_1} e e_{i,j,k_2} são divergentes;
  Classifique cada componente como orientado, trivial ou não orientado;
  Construa o grafo de obstáculos $\mathcal{H}$; utilize $\mathcal{H}$ para classificar cada componente orientado como um obstáculo simples, um protetor simples, um pseudo-obstáculo, um (não protetor único) super obstáculo ou um (não pseudo-obstáculo) obstáculo protegido. Identifique também quaisquer *SODs* e anote para cada membro, de um *SOD*, o seu parceiro;
  Sejam c o número de ciclos em $\mathcal{B}$, h o número de obstáculos e s o número de super obstáculos; Atribua $f = 1$ se $h = s$ e s é ímpar. Caso contrário, $f = 0$;
  Inicie a lista L;
  `append_all(`L`,get_revs_split_cycles());`
  `append_all(`L`,get_revs_cut_hurdles());`
  se $f = 0$ **então**
    `append_all(`L`,get_revs_merge_nofort());`
  senão
    `append_all(`L`,get_revs_merge_fort());`
  fim
  retorna L;
fim

Algoritmo A.3: Algoritmo `get_revs_split_cycles`

Entrada: Todos $e_{i,j,k}, o_{i,j,k}$ e os valores h, s e f recebidos do algoritmo `find_all_sorting_reversals`; assuma a existência da função `detect` que retorna uma lista de novos componentes não orientados introduzidos por uma dada reversão ou retorna o conjunto vazio ($\emptyset$) se nenhum componente não orientado é introduzido.

Saída: Uma lista M de todas as *sorting reversals* que quebram ciclos.

```
1  início
2      Inicie a Lista M;
3      /* Enumera todas as arestas divergentes do mesmo ciclo */
4      para cada e_{i,j,k_1} e e_{i,j,k_2} tais que o_{i,j,k_1} · o_{i,j,k_2} = −1 faça
5          P ← detect(ρ(e_{i,j,k_1}, e_{i,j,k_2}));
6          se P = ∅ então
7              /* Nenhum componente não orientado novo (Teo. A.3-1) */
8              append(M, ρ(e_{i,j,k_1}, e_{i,j,k_2}));
9          fim
10         /* Ao menos um componente não orientado novo (Teo. A.3-2 e 3) */
11         Conte o novo número de obstáculos (h') e determine se uma fortaleza existe na
           nova permutação (f');
12         se h' + f' = h + f então
13             /* Ambos os Casos 2 e 3 devem ser aplicados */
14             append(M, ρ(e_{i,j,k_1}, e_{i,j,k_2}));
15         fim
16     fim
17     retorna M;
18 fim
```

Algoritmo A.4: Algoritmo `get_revs_cut_hurdles`

Entrada: Todos $e_{i,j,k}, o_{i,j,k}$ e os valores h, s e f recebidos do algoritmo `find_all_sorting_reversals`.

Saída: Uma lista M de todas as *sorting reversals* que cortam obstáculos (ou pseudo-obstáculos).

```
1  início
2      Inicie a Lista M;
3      se f = 0 então
4          /* Teo. A.4-1 */
5          H ← {i | componente i é um super obstáculo };
6          se s = 2a + 1 e h = 2a + 2 (para um inteiro positivo a) então
7              /* Evita uma fortaleza */
8              retorna M;
9          fim
10     senão
11         /* Teo. A.4-2 */
12         H ← {i| componente i é um pseudo-obstáculo ou um protetor único};
13     fim
14     /* Enumera todos pares de arestas pertencentes ao mesmo ciclo de um
       componente em H */
15     para cada e_{i,j,k_1} e e_{i,j,k_2} tais que i ∈ H e k_1 ≠ k_2 faça
16         append(M, ρ(e_{i,j,k_1}, e_{i,j,k_2}));
17     fim
18     retorna M
19 fim
```

Algoritmo A.5: Algoritmo `get_revs_merge_nofort`

Entrada: Todos $e_{i,j,k}, o_{i,j,k}$ e os valores h, s e f recebidos do algoritmo `find_all_sorting_reversals`.

Saída: Uma lista M de todas as *sorting reversals* que unem componentes separados, assumindo que não existem fortalezas.

```
1  início
2      Inicie a Lista M;
3      Encontro todos obstáculos separadores: Seja S_i = {j | j é um componente benigno
       cujo obstáculo separador é o componente i};
4      H ← {i | componente i é um obstáculo };
5      /* Enumera todos pares de obstáculos */
6      para cada i, k ∈ H tais que i ≠ k faça
7          se Obstáculo i e k formam um SOD então
8              /* Evita um SOD (Teo. A.6-1c) */
9              continue;
10         senão se (s = 2a + 1, h = 2a + 3 [para um inteiro positivo a], e os obstáculos i e k
           são ambos obstáculos simples) ou (s = 2a + 2, h = 2a + 3 [para um inteiro positivo
           a], e um dos obstáculos i e k é um super obstáculo) então
11             /* Evita uma fortaleza (Teo. A.6-1d) */
12             continue;
13         senão
14             para cada j ∈ {i} ∩ S_i faça
15                 /* j é i ou é separado por i */
16                 para cada l ∈ {k} ∩ S_k faça
17                     /* l é k ou é separado por k; Teo. A.6-1 deve se aplicar a l e j
                       */
18                     para cada e_{j,y1,z1}, e_{l,y2,z2} faça
19                         append(M, ρ(e_{j,y1,z1}, e_{l,y2,z2}));
20                     fim
21                 fim
22             fim
23         fim
24     fim
25     retorna M
26 fim
```

Algoritmo A.6: Algoritmo `get_revs_merge_fort`

Entrada: Todos $e_{i,j,k}, o_{i,j,k}$ e os valores h, s e f recebidos do algoritmo `find_all_sorting_reversals`.

Saída: Uma lista M de todas as *sorting reversals* que unem componentes separados, assumindo que existe uma fortaleza.

início
  Inicie a Lista M;
  Encontre todos obstáculos separadores: Seja $S_i = \{j \mid j$ é um componente benigno cujo obstáculo separador é o componente $i\}$; e seja $S_\neg = \{j \mid j$ é um componente benigno que não possui obstáculo separador $\}$;
  $H \leftarrow \{i \mid i$ é um obstáculo$\}$ e $U \leftarrow \{j \mid j$ é um componente não orientado$\}$;
  Marque cada componente não orientado com sua cadeia de obstáculo: seja $\gamma_i = i$ se $j \in U, i \in H$ e j pertence à cadeia de obstáculos de i; seja $\gamma_j = -1$ se j não pertence a nenhuma cadeia de obstáculos;
  Identifique a âncora de cada cadeia: seja $\alpha_i = j$ se $j \in U$ é âncora da cadeia de $i \in H$. Inicie `mark`$[i]$ com o valor para todo $i \in H$;
  para cada $i \in H$ **faça**
    `mark`$[i] \leftarrow 1$; `/* Marque` i `como visitado */`
    `/* Outros obstáculos e componentes benignos que eles separam, excluindo parceiros de` *SODs* `(Teo. A.6-2a) */`
    $V \leftarrow \{j \mid k \in H$ e `mark`$[k] = 0$ e k não é parceiro de i (SOD) e $j \in \{k \cap S_k\}$;
    `/* A âncora da cadeia de` i `(Teo. A.6-2b e Lema A.9-3a) */`
    $W \leftarrow \{\alpha_i\}$;
    `/* Obstáculos não protegidos que não estão na cadeia de` i `(Teo. A.6-2b e Lema A.9-3b) */`
    $X \leftarrow \{j \mid k \in U$ e $j \notin H$ e $\gamma_j \neq i\}$;
    `/* Componentes benignos que não possuem obstáculos separadores e que não são separados, de um componente na cadeia de` i`, por outro componente (Teo. A.6-2b e Lema A.9-3c) */`
    $Y \leftarrow \{j \mid j \in S_\neg$ e $\nexists k, l$ tais que $\gamma_k = i, \gamma_l = i$ e k separa j de $l\}$;
    `/* Parceiros de` i `em` *SODs* `e componentes benignos que eles separam (Teo. A.6-2b e Lema A.9-3d) */`
    $Z \leftarrow \{j \mid k$ é um parceiro de i em um SOD e `mark`$[k] = 0$ e $j \in \{k\} \cap S_k\}$;
    para cada $j \in \{i\} \cap S_i$ **faça**
      `/*` j `é` i `ou é separado por` i `*/`
      para cada $l \in V \cap W \cap X \cap Y \cap Z$ **faça**
        `/*` l `é tal que Teo. A.6-2 se aplica a` j `e` l `*/`
        para cada $e_{j,y_1,z_1}, e_{j,y_2,z_2}$ **faça** `append(`M`,` $\rho(e_{j,y_1,z_1}, e_{l,y_2,z_2})$`);`
      fim
    fim
  fim
  retorna M
fim

Lema A.10. *O efeito de qualquer reversão em um grafo de sobreposição é complementar o subgrafo de todos vértices correspondentes às arestas cinzas afetadas.*

O autor não detalha a descrição da segunda versão do algoritmo `detect`. No entanto ele afirma que este algoritmo obtém um novo grafo de sobreposição, que seria resultado do uso de uma reversão candidata, a partir da utilização do Lema A.10 e das técnicas introduzidas por Bergeron e Strasbourg, que permitem que o grafo de sobreposição seja representado por uma matriz de bits e mudanças aplicadas a ela são modeladas com a utilização de operações `bitwise` *or*, *e* e *ou exclusivo*. Este algoritmo gasta tempo $O(k^2)$, onde k é o número de vértices no componente afetado.

O algoritmo `find_all_sorting_reversals` gasta tempo $O(n^3)$ ou $O(n^4)$ dependendo da escolha entre a primeira e a segunda versão da rotina `detect`.

A enumeração de todas as reversões por força bruta pode ser feita em tempo $\Theta(n^3)$. Contudo, testes realizados pelo autor mostraram que, na prática, o novo algoritmo oferece ganho considerável de performance.

Os testes consideram dados divididos em três classes. A primeira classe foi construída com pares aleatórios de permutações orientadas. Para cada par um dos membros sofria um número específico de reversões aleatórias. A segunda classe era similar a primeira, mas ao invés da aplicação de reversões aleatórias, um procedimento foi adotado para introduzir componentes não orientados. Finalmente, a terceira classe era composta por permutações, especialmente selecionadas, que apresentassem configurações que dificilmente seriam encontradas nas outras duas classes.

A corretude do teste era avaliada comparando-se os resultados obtidos pelo algoritmo `find_all_sorting_reversals` com os resultados do algoritmo que enumerava as reversões por força bruta.

As permutações testadas possuíam n no intervalo entre 25 e 100 e número de reversões aleatórias r no intervalo entre 0% a 100% de n.

Os resultados mostraram que as duas versões do novo algoritmo apresentaram grande ganho de performance em relação ao algoritmo de força bruta. Para $r \leq 10$, o ganho de velocidade foi de 400 vezes. Observou-se também que a segunda versão do algoritmo `detect` é significativamente mais rápida que a primeira.

Finalmente, o autor conclui que enumerar todas as soluções que ordenam uma reversão orientada é de algum interesse teórico. Contudo, segundo ele o algoritmo será de maior utilidade se aplicado para solução de problemas onde restrições adicionais possam ser adotadas com o objetivo de reduzir o tamanho do conjunto resposta.

O autor também sugere uma extensão do problema que seria a enumeração de todas as soluções que ordenam uma permutação π e excedam o valor $d(\pi)$ em uma pequena constante k. De acordo com o autor, este novo problema poderia ser facilmente resolvido através da adoção da capacidade de enumerar reversões neutras, ou seja, reversões que não alteram a distância de reversão.

A.6 A very elementary presentation of the Hannenhalli-Pevzner theory [8]

O objetivo deste artigo é fornecer uma apresetação elementar para a teoria apresentada por Hannenhalli e Pevzner [42].

Uma reversão $\rho(i,j)$ transforma uma permutação $\pi = (\pi_1 \ \ldots \ \underline{\pi_i \ \pi_{i+1} \ \ldots \ \pi_j} \ \ldots \ \pi_n)$ para $\pi' = (\pi_1 \ \ldots \ \pi_j \ \ldots \ \pi_{i+1} \ \pi_i \ \ldots \ \pi_n)$. A *distância de reversão* entre duas permutações é o número mínimo de reversões que transforma uma permutação em outra. O problema de se encontrar o número mínimo de reversões que ordenam uma permutação é NP-difícil [16].

Na versão orientada do problema, os elementos da permutação possuem sinais positivos ou negativos e uma reversão $\rho(i,j)$ transforma a permutação π na permutação $\pi' = (\pi_1 \ \ldots \ -\pi_j \ \ldots \ -\pi_{i+1} \ -\pi_i \ \ldots \ \pi_n)$. Ao contrário do problema mais geral, esta versão pode ser resolvida em tempo polinomial, como descrito no trabalho de Hannenhalli e Pevzner [42]. Nesse trabalho, o conceito de *fortaleza* foi introduzido para explicar casos especialmente resistentes ao processo de ordenação. Hannenhalli e Pevzner utilizam inúmeras construções intermediárias e alguns trabalhos produziram simplificações para elas [2,11,49].

Todos os critérios para a escolha de uma *reversão segura* envolvem a construção de uma permutação de $2n$ pontos e a análise de ciclos e/ou componentes conexos no grafo associado a esta permutação. Neste artigo, o autor apresenta um tratamento elementar para a ordenação de *componentes orientados* de uma permutação, além de fornecer uma nova definição do conceito de *obstáculos*, que simplifica aquela apresentada por Kaplan *et al.* [49].

O problema de calcular a distância entre duas permutações é frequentemente tratado como o problema de calcular $d(\pi)$, a distância de reversão entre a permutação π e a permutação identidade $(1 \ 2 \ \ldots \ n)$. Neste artigo, o trabalho foca na reconstrução de uma possível sequência de reversões que realiza $d(\pi)$, problema conhecido como *ordenação por reversão*. Aqui, assume-se que a permutação recebe dois elementos adicionais, 0 e $n+1$ (sempre positivos), de modo que $\pi = (0 \ \pi_1 \ \ldots \ \pi_i \ \pi_{i+1} \ \ldots \ \pi_j \ \ldots \ \pi_n \ n+1)$.

Um *par orientado* (π_i, π_j), $i < j$, é um par de inteiros consecutivos e com sinais opostos tais que, $|\pi_i| - |\pi_j| = \pm 1$. Tais pares são úteis porque indicam reversões que podem criar elementos consecutivos na permutação. Por exemplo, o par $(1, -2)$ da permutação $(0 \ 3 \ 1 \ 6 \ 5 \ -2 \ 4 \ 7)$ induz a reversão do intervalo $(6 \ 5 \ -2)$, gerando a permutação $(0 \ 3 \ 1 \ 2 \ -5 \ -6 \ 4 \ 7)$ e fazendo com que os elementos 1 e 2 fiquem consecutivos. A reversão induzida por um par orientado (π_i, π_j) será $\rho(i, j-1)$ se $\pi_i + \pi_j = +1$, e $\rho(i+1, j)$ se $\pi_i + \pi_j = -1$.

Note que reversões que criam inteiros consecutivos são sempre induzidas por pares orientados. Reversões deste tipo são denominadas *reversões orientadas*. A pontuação de uma reversão orientada é definida como o número de pares orientados que existem na permutação resultante. O fato de as reversões orientadas terem um efeito benéfico na ordenação de uma permutação sugere uma primeira estratégia de ordenação.

Algoritmo A.1. *Enquanto π possuir um par orientado, escolha uma reversão orientada que maximize a pontuação.*

Um aspecto interessante da estratégia deste algoritmo é que ela é suficiente para otimizar a ordenação da maioria das permutações aleatórias e quase todas permutações que têm origem em dados biológicos. Esta estratégia é ótima e os autores provam a seguinte proposição relacionado à ela.

Proposição A.2. *Se a estratégia do Algoritmo A.1 aplica k reversões em uma permutação π, produzindo uma permutação π', então $d(\pi) = d(\pi') + k$.*

A saída do Algoritmo A.1 produz uma permutação de elementos positivos. A maioria das reversões aplicadas a permutações como essas criaram pares orientados. Contudo, a escolha de uma reverão ótima é uma tarefa delicada.

Deixe π ser uma permutação orientada com apenas elementos positivos e assuma que π é *reduzida*, ou seja, que π não possui inteiros consecutivos. Suponha também que π é *framed* por 0 e $n+1$ e considere uma ordem circular induzida pela adoção do 0 como sucessor do $n+1$.

Defina um *framed interval* em π como um intervalo da forma $i\ \pi_{j+q}\ \pi_{j+2}\ \ldots\ \pi_{j+k-1}\ i{+}k$, tal que todos inteiros entre i e $i+k$ pertencem ao intervalo $[i \ldots i+k]$. Por exemplo, a permutação (0 2 5 4 3 6 1 7) é, inteira, um *framed interval*. Contudo, ainda podemos temos o intervalo 2 5 4 3 6, que pode ser ordenado como 2 3 4 5 6 e, por circularidade, o intervalo 6 1 7 0 2, que pode ser ordenado como 6 7 0 1 2.

Definição A.9. *Se π é reduzida, um* obstáculo *em π é um* framed interval *que não contém um* framed interval *menor.*

Proposição A.3. *Obstáculos como os definidos na Definição A.9 são os mesmos obstáculos que são definidos nos trabalhos de Hannenhalli e Pevzner [42] e Kaplan* et al. *[49].*

Quando uma permutação tem apenas um ou dois obstáculos, uma reversão é suficiente para criar pares orientados suficientes para que seja possível completar a ordenação da permutação com o Algoritmo A.1. No trabalho de Hannenhalli e Pevzner [42], duas operações são introduzidas para tratamento destes casos.

A primeira operação é o *corte de obstáulos* que consite na reversão do segmento entre i e $i+1$ de um obstáculo. Esta operação é suficente para tornar possível a ordenação do intervalo com o Algoritmo A.1.

A outra operação é a *união de obstáculos* que age nas extremidades de dois obstáculos $i \ldots i+k \ldots i' \ldots i'+k'$ fazendo a reversão $\rho(i+k, i')$. Se uma permutação tem apenas dois obstáculos, a união deles produzirá uma permutação que pode ser completamente ordenada pelo Algoritmo A.1.

Se a permutação tiver mais de dois obstáculos, as operações acima devem ser aplicadas com cuidado. Inclusive, cortar certos obstáculos pode gerar novos obstáculos.

Definição A.10. *Um* obstáculo simples *é um obstáculo cujo corte diminui o número de obstáculos. Obstáculos que não são simples são denominados* super obstáculos.

Algoritmo A.2. *Se uma permutação possui $2k$ obstáculos, $k \geq 2$, faça a união de dois obstáculos não consecutivos. Se uma permutação possui $2k+1$ obstáculos, $k \geq 1$, então se existir uma obstáculo simples, corte-o; caso contrário, faça a união de dois obstáculos não consecutivos, ou consecutivos se $k = 1$.*

Os algoritmos A.1 e A.2 podem ser utilizados para otimizar a ordenação de qualquer permutação sinalizada. Permutações que não estão reduzidas podem ser sempre reduzidas através da fusão de inteiros consecutivos e renumeração de todos os elementos.

O autor apresenta uma série de lemas e teoremas para discutir os algoritmos propostos. Para isso, ele usa os grafos de *breakpoint* e de *sobreposição*.

O *grafo de breakpoint* de uma permutação π é construído com base na permutação π' associada a π da seguinte maneira: Cada elemento positivo x em π é substituído pela

sequência $2x - 12x$ e cada elemento negativo $-x$ é substituído por $2x2x - 1$. Note que reversões $\rho(i, j)$ de π são simuladas pelas reversões não orientadas $\rho(2i - 1, 2j)$ em π'.

Os elementos de π' são os vértices do grafo de *breakpoint*. Arestas retas unem todos os pares de elementos consecutivos de π' e arestas curvas, também chamadas de *arcos*, unem pares de inteiros consecutivos em π'. Neste grafo, todo componente conexo é um ciclo.

O *suporte* de um arco é o intervalo de elementos de π' que estão entre as extremidades (incluindo as extremidades). Dois arcos se sobrepõem se seus suportes possuem uma intersecção, sem estarem um contido no outro. Um arco é *orientado* se seu suporte possui um número ímpar de elementos, caso contrário, ele é *não orientado*. Note que um arco é orientado se e somente se suas extremidades são imagens de um par orientado na permutação π.

O *grafo de sobreposição* (ou *grafo de sobreposição de arcos*) é o grafo cujo vértices são arcos do grafo de *breakpoint* e arestas unem os vértices que representam arcos que se sobrepõem. Os *vértices orientados* são aqueles que representam os arcos orientados. Neste grafo, um componente conexo que possui ao menos um vértice orientado é chamado *componente orientado*.

Existe uma bijeção natural entre os vértices de um grafo de sobreposição e o pares (não orientados) de inteiros consecutivos x e $x + 1$ na permutação original. Um par de inteiros consecutivos irá gerar quatro inteiros consecutivos na permutação π': $2x - 1$, $2x$, $2x + 1$ e $2x + 2$. O vértice $(2x, 2x + 1)$ está associado ao par x e $x + 1$. O autor utiliza o termo *reversão induzida por um vértice* para indicar uma reversão que é induzida pelo par orientado correspondente na permutação original.

Lema A.11. *Um vértice possui um grau ímpar se e somente se ele é orientado.*

Lema A.12. *Se uma reversão correspondente a um vértice orientado v é aplicada, o efeito no grafo de sobreposição será o de complementar o subgrafo induzido por v e seus vértices adjacentes.*

Lema A.13. *Se uma reversão correspondente a um vértice orientado v é aplicada, cada vértice adjacente a v irá trocar sua orientação.*

Lema A.14. *A pontuação de uma reversão orientada correspondente a um vértice orientado v é dada por $T + U - O - 1$, onde T é o número total de vértices orientados no grafo, U é o número de vértices não orientados adjacentes a v e O é o número de vértices orientados adjacentes a v.*

Teorema A.7 (Hannenhalli e Pevzner [42])**.** *Qualquer sequência de reversões orientadas seguras é ótima.*

A dificuldade em ordenar componentes orientados está na detecção de reversões que sejam seguras. Hannenhalli e Pevzner lidam com o problema através do cálculo de diversas estatísticas sobre ciclos e *breakpoints* de vários grafos. Kaplan *et al.* [49] resolvem o problema através da busca por cliques particulares em grafos de sobreposição. O próximo teorema mostra que a estratégia elementar de escolher a reversão com pontuação máxima é ótima, provando desta maneira a Proposição A.2.

Teorema A.8. *Uma reversão que maximiza a pontuação é segura.*

Prova Suponha que o vértice v tem uma pontuação máxima e que a reversão induzida por v cria um novo componente não orientado C contendo mais de um vértice. Ao menos um dos vértices de C deve ter sido adjacente a v, pois as únicas arestas afetadas pela reversão são aquelas entre vértices adjacentes a v. Deixe w ser um vértice anteriormente adjacente a v e contido em C e considere as seguintes pontuações de v e w:

$\texttt{pontuação}(v) = T + U - O - 1$

$\texttt{pontuação}(w) = T + U' - O' - 1$

Todos vértices não orientados anteriormente adjacentes a v devem estar adjacentes a w. Além disso, um vértice não orientado adjacente a v e não adjacente a w se tornará orientado e conectado a w, contrariando a suposição de que C é não orientado. Logo $U' \geq U$.

Todos vértices orientados anteriormente adjacentens a w devem estar adjacentes a v. Se este não for o caso, um vértice orientado adjacente a w, mas não adjacente a v permanecerá orientado, novamente contrariando a suposição de que C é não orientado. Logo $O' \leq O$.

Agora, se ambos $O' = O$ e $U' = U$, os vértices v e w possuem o mesmo conjunto de vértices adjacentes, e o complemento do subgrafo de v e seus vértices adjacentes irá isolar ambos v e w. Nessas condições, teríamos que $\texttt{pontuação}(w) > \texttt{pontuação}(v)$, o que é uma contradição. □

Para provar a Proposição A.3, assumimos que π é uma permutação positiva e reduzida. Isso equivale a dizer que o grafo de sobreposição não possui componentes orientados, todos eles removidos com ajuda do Algoritmo A.1, e que não existem vértices isolados.

A ordem circular é também utilizada, desta vez sobre o intervalo $[0..2n-1]$. O *span* de um conjunto de vértices X no grafo de sobreposição é o intervalo mínimo que possui, em ordem circular, todos vértices em X.

Obstáculos são definidos por Hannenhalli e Pevzner [42] como componentes não orientados que são minimais com respeito a ordem induzida pela inclusão em *span*. Além disso, Kaplan *et al.* [49] mostram que o *span* de um componente conexo é sempre da forma $[2i, 2j-1]$.

Lema A.15. Framed intervals *na forma* $[i, j]$ *em uma permutação de* n *elementos estão em uma correspondência de um para um com* framed intervals *da forma* $[2i, 2j-1]$ *na permutação não orientada de* $2n$ *elementos correspondente.*

Lema A.16. *Qualquer* framed interval $[2i, 2j-1]$ *é o* span *de uma união de componentes conexos.*

Lema A.17. *O* span $[2i, 2j-1]$ *de um componente conexo é sempre um* framed interval.

Teorema A.9. *Se* π *é reduzida, um componente não orientado é minimal se e somente se seu* span *é um* framed interval *que não contém nenhum outro* framed interval.

Usando o Lema A.15, o Teorema A.9 fornece uma caracterização elementar para o conceito de obstáculos, que não se baseia na construção de um grafo de sobreposição.

O Algoritmo A.1 possui uma implementação direta: para encontrar uma reversão segura, execute cada reversão orientada possível na permutação original e conte o número de reversões orientadas resultantes. Como existem $O(n)$ reversões orientadas em uma permutação de n elementos, e como calcular o número de reversões orientadas em uma

permutação pode ser feito em $O(n)$ operações, isto resulta em um algoritmo $O(n^3)$ para ordenar uma permutação de n elementos.

Contudo, realizar uma reversão ou calcular a sua pontuação, é uma operação "local" no grafo de sobreposição. Isto sugere a possibilidade de paralelização do algoritmo para manter as pontuações e calcular os efeitos das reversões.

Assim, na porção final do artigo, o autor propõem um algoritmo que explora o paralelismo inerente as operações básicas de um processador ao utilizar vetores de bits e manipulá-los com apenas três operações: *ou exclusivo* $\oplus$, *conjunção* $\wedge$ e *negação* $-$.

A.7 A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study [2]

O objetivo deste artigo é apresentar o primeiro algoritmo linear para cálculo da distância de inversão entre permutações orientadas.

Em 1995, Hannenhalli e Pevzner [42] apresentaram o primeiro algoritmo polinomial para ordenação de permutações orientadas por reversões. Este algoritmo possui complexidade $O(n^4)$ e executa em tempo quadrático se for restringido para cálculo apenas da distância de reversão.

Em 1996, Berman e Hannenhalli exploraram propriedades combinatórias do *grafo de breakpoint* para produzir um algoritmo $O(n\alpha(n))$ para cálculo de componentes conexos e um algoritmo $O(n^2\alpha(n))$ para resolução do problema de ordenação [11]. Os autores se referem à estratégia deste algoritmo como estratégia *Union-Find – UF*. Apesar de existirem outras alternativas para cálculo componentes conexos em grafos de intervalos (classe de grafos que incluem o grafo de sobreposição) que possuem complexidade linear, as estruturas que elas utilizam são complexas e sofrem de *overhead* grande o suficiente para tornar a estratégia *UF* mais rápida do que elas.

Segundo os autores, nenhum algoritmo que construa o grafo de sobreposição pode executar em tempo linear. Isto se deve ao fato deste grafo possuir tamanho quadrático. Assim, a estratégia utilizada é a de se construir uma *floresta de sobreposição* tal que dois vértices f e g pertencem a uma mesma árvore na floresta quando eles pertencem exatamente ao mesmo componente conexo de um grafo de sobreposição. Uma floresta de sobreposição possui exatamente uma árvore para cada componente conexo no grafo de sobreposição e, portanto, possui tamanho linear.

O algoritmo proposto para cálculo dos componentes conexos percorre a permutação duas vezes. No primeiro passo, uma floresta trivial é criada de modo que cada nó tem sua própria árvore, nomeada com o início do seu ciclo. No segundo passo, um refinamento da primeira floresta é conduzido através da adição de arestas e da união de árvores da floresta. Esta estratégia é similar a estratégia *UF*, contudo ela não exige que as árvore formadas respeitem parâmetros de forma.

A extensão de um ciclo C é um intervalo $[C.B, C.E]$ tal que, $C.B = min\{i|\pi(i) \in C\}$ e $C.E = max\{i|\pi(i) \in C\}$. A extensão de um conjunto de ciclos $\{C_1, \ldots, C_k\}$ é um intervalo $[B, E]$ com $B = min_{i=1}^{k} C_i.B$ e $E = max_{i=1}^{k} C_i.E$.

Um nó em um grafo de sobreposição (ou floresta) corresponde a um ciclo no grafo de *breakpoint*. A extensão $[f.B, f.E]$ de um nó da floresta de sobrebosição é a extensão do

conjunto de nós na sub-árvore cuja raiz é f. Deixe F_0 ser a floresta trivial configurada no primeiro passo e assuma que o algoritmo processou elementos de 0 até $j-1$ da permutação, produzindo a floresta F_{j-1}. A construção de F_j a partir de F_{j-1} é feita da seguinte forma: Seja f o ciclo que contém o elemento j da permutação. Se j é o início de seu próprio ciclo f, então ele deve deve ser a raiz de uma árvore que possui um único nó, caso contrário, se f sobrepõem outro ciclo g, entao adicionamos um novo arco (g, f) e caculamos a extensão combinada de g e da árvore que possui f como raiz. Dizemos que um árvore que possui raiz em f é *ativa* no estágio j se j está propriamente contido na extensão de f. A extensões das árvores ativas devem ser armazenadas em uma pilha.

A Algoritmo A.7 exibe os passos para a construção da floresta de sobreposição. Neste algoritmo *top* denota o elemento que está no topo da pilha e `push` e `pop` são as operações de pilha para empilhar e desempilhar elementos. A conversão de uma floresta de árvores [up-trees?] em enumerações de componentes conexos é feita em tempo linear através de um simples percurso no vetor, tirando vantagem do fato que o pai de i deve aparecer antes de i no vetor.

Lema A.18. *Na iteração i do laço `Enquanto` (linha 4 do Algoritmo A.7), se a árvore que possui a raiz em top está ativa, i está no ciclo f e $f.B < top.B$, então existe h na árvore que possui a raiz em top tal que h sobrepõem f.*

Teorema A.10. *O Algoritmo A.7 produz uma floresta na qual cada árvore é composta, exatamente, pelos nós que formam um componente conexo.*

Prova Para provar o Teorema A.10 é suficiente mostrar que, após cada iteração, as árvores na floresta correspondem exatamente aos componentes conexos determinados pleos valores obtidos da permutação até aquele ponto. A prova é feita por indução sobre o número de aplicações da laço `Enquanto` (linha 4 do Algoritmo A.7).

O caso base é trivial: cada árvore de F_0 tem um único nó e não existem dois nós que pertençam a um mesmo componente, já que nenhum elemento da permutação foi processado.

Assuma que a invariante está após a $(i-1)$-ésima iteração e deixe i pertencer ao ciclo f. Nós provamos que os nós da árvore que contém i formam o mesmo conjunto de nós do componente conexo que contém i. Outras árvores e componentes conexos não são afetados e, assim, obedece a invariante.

- *Prova de que um nó na árvore contendo i deve estar no mesmo componente conexo de i.* Se temos $i = f.B$, então nada muda no grafo de sobreposição (e nos componentes conexos). Do laço `Enquanto` (linha 4 do algoritmo), também é claro que a floresta permanece inalterada, portanto a invariante é preservada. Por outro lado, se temos que $i > f.B$, então durante o laço, a aresta (top, f) será adicionada à floresta, se $f.B < top.B$ for verdade. Esta aresta irá unir a sub-árvore que tem raiz em f com a sub-árvore que tem raiz em *top* em uma única sub-árvore. Do lema A.18, sabemos que se $f.B < top.B$ é verdadeiro, então existe h na árvore que possui em *top* tal que, h e f sobrepõem. Portanto, a aresta (h, f) deve pertencer ao grafo de sobreposição. Assim, conectando o componente contendo f com aquele que contém *top* e unindo-os em um único componente conexo, a invariante é mantida.

- *Prova de que um nó no mesmo componente conexo de i deve estar na mesma árvore contendo i.* Se (j, i) e (k, l), com $j < k < i < l$, são arestas cinzas nos ciclos f e h, respectivamente, então a aresta (f, h) deve pertencer ao grafo de sobreposição construído a partir das primeiras i entradas da permutação. Neste caso, o algoritmo garante que a aresta (h, f) pertence a floresta de sobreposição. □

Como cada passo do algoritmo gasta tempo linear, então o algoritmo inteiro executa em tempo-linear no pior caso.

Na porção final do artigo os autores apresentam os resultados de experimentos para comparação de velocidade entre o algoritmo proposto o algoritmo de Berman e Hannenhalli (apenas o trecho que realiza o cálculo da distância de inversão). Ambos os algoritmos foram implementados em C e escritos de acordo com técnicas de engenharia de algoritmos para que eles pudessem ser propriamente comparados. Os experimentos apresentam resultados que mostram ganhos de velocidade de até $1,8$ vezes do algoritmo proposto em relação ao outro algoritmo, especialmente em permutações que apresentam distâncias de reversão maiores.

Algoritmo A.7: Algoritmo para contrução da floresta de sobreposição de uma permutação π em tempo linear

Entrada: Permutação π
Saída: `parent`$[i]$, o pai de i na floresta de sobreposição

```
1  início
2      Percorra a permutação, marque cada posição i com C[i].B e calcule a extensão
       [C[i].B, C[i].E];
3      Crie uma pilha vazia;
4      para i ← 0 até 2n + 1 faça
5          se i = C[i].B então
6              push C[i];
7          fim
8          extent ← C[i];
9          enquanto top.B > C[i].B faça
10             extent.B ← min{extent.B, top.B};
11             extent.E ← max{extent.E, top.E};
12             pop top;
13             parent[top.B] ← C[i].B;
14         fim
15         top.B ← min{extent.B, top.B};
16         top.E ← max{extent.E, top.E};
17         se i = top.E então
18             pop top;
19         fim
20     fim
21     Converta cada árvore em uma "enumeração" de seus vértices;
22 fim
```