Alinhamento de seqüências com rearranjos

Augusto Fernandes Vellozo

TESE APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO DE DOUTOR
EM
CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Alair Pereira do Lago

Durante a elaboração deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, março de 2007.

Resumo

Uma das tarefas mais básicas em bioinformática é a comparação de seqüências feita por algoritmos de alinhamento, que modelam as alterações evolutivas nas seqüências biológicas através de mutações como inserção, remoção e substituições de símbolos. Este trabalho trata de generalizações nos algoritmos de alinhamento que levam em consideração outras mutações conhecidas como rearranjos, mais especificamente, inversões, duplicações em *tandem* e duplicações por transposição.

Alinhamento com inversões não tem um algoritmo polinomial conhecido e uma simplificação para o problema que considera somente inversões não sobrepostas foi proposta em 1992 por Schöniger e Waterman [55]. Em 2003, trabalhos independentes propuseram algoritmos com tempo[1] $O(n^4)$ [20, 31, 21] para alinhar duas seqüências com inversões não sobrepostas. Desenvolvemos dois algoritmos que resolvem este mesmo problema: um em tempo $O(n^3 \log n)$ [4] e outro em tempo $O(n^3)$ [62], ambos em memória $O(n^2)$.

Em 1997, Benson propôs um modelo de alinhamento que reconhecesse também as duplicações em *tandem*. Ele propôs dois algoritmos exatos para alinhar duas seqüências com duplicações em *tandem*: um em tempo $O(n^5)$ e memória $O(n^2)$, e outro em tempo $O(n^4)$ e memória $O(n^3)$. Propomos um algoritmo para alinhar duas seqüências com duplicações em *tandem* em tempo $O(n^3)$ e memória $O(n^2)$. Propomos também um algoritmo para alinhar duas seqüências com *transposons* (um tipo mais geral que a duplicação em *tandem*), em tempo $O(n^3)$ e memória $O(n^2)$.

[1] Neste caso, n denota o tamanho máximo dentre as duas seqüências a serem alinhadas.

Abstract

Sequence comparison done by alignment algorithms is one of the most fundamental tasks in bioinformatics. The evolutive mutations considered in these alignments are insertions, deletions and substitutions of nucleotides. This work treats of generalizations introduced in alignment algorithms in such a way that other mutations known as rearrangements are also considered, more specifically, we consider inversions, duplications in tandem and duplications by transpositions.

Alignment with inversions does not have a known polynomial algorithm and a simplification to the problem that considers only non-overlapping inversions were proposed by Schöniger and Waterman [55] in 1992. In 2003, independent works proposed algorithms with $O(n^4)$ [20, 31, 21] time[2] to align two sequences with non-overlapping inversions. We developed two algorithms to solve this problem: one in $O(n^3 \log n)$ [4] time and other in $O(n^3)$ [62] time, both in $O(n^2)$ memory.

In 1997, Benson proposed a model of alignment that also recognized tandem duplication. He proposed two exact algorithms to align two sequences with tandem duplication: one in $O(n^5)$ time and $O(n^2)$ memory, and other in $O(n^4)$ time and $O(n^3)$ memory. We propose one algorithm to align two sequences with tandem duplication in $O(n^3)$ time and $O(n^2)$ memory. We also propose one algorithm to align two sequences with transposons (a type of duplication more general than tandem duplication), in $O(n^3)$ time and $O(n^2)$ memory.

[2]In this case, n denotes the maximal length of the two aligned sequences.

Sumário

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Atualmente, devido ao grande número de projetos de seqüenciamentos desenvolvidos e finalizados, temos uma enorme quantidade de dados moleculares disponíveis, principalmente de DNA, RNA e proteína. Processar estes dados para extrair informações relevantes é um grande desafio.

Uma das maneiras de manipular, estudar e estruturar estes dados moleculares é através do uso de seqüências de símbolos. No caso do DNA os símbolos são as letras que representam as bases de nucleotídeos (A, C, T e G), no caso das proteínas os símbolos são letras que representam os 20 aminoácidos elementares.

À medida que o número de novos genomas completos aumenta, a comparação entre seqüências longas de DNA de espécies próximas torna-se mais importante para nosso entendimento da estrutura da seqüência do DNA. Devido a isto, a análise genômica comparativa [41], apesar de ser um novo campo na bioinformática, está se desenvolvendo rapidamente. Em muitas espécies próximas, a ordem dos genes é preservada para intervalos curtos [47]. Nesses casos, os genes são mais conservados do que as regiões intergênicas. Portanto, a ordem da seqüência de genes é muito útil para detectar reordenamentos cromossômicos como inversões. Estes tipos de comparações ganham maior significância à medida que mais segmentos de genomas ortólogos, fortemente relacionados pela evolução, são seqüenciados.

Desde a finalização do rascunho do genoma humano novos projetos de seqüenciamento têm sido desenvolvidos para comparação com o genoma humano. Muitos programas computacionais têm sido usados para esse propósito como VISTA [45, 22], GLASS [9], Mummer [19], PipMaker [56], e também BLAST 2 Sequences [60].

Essas comparações genômicas muitas vezes dependem fortemente da comparação de trechos menores dos genomas. Em outros estudos em bioinformática, assim como nas comparações genômicas, a comparação de seqüências biológicas é muito utilizada. Uma das maneiras de se executar estas comparações de seqüências biológicas é através do alinhamento de seqüências, os quais são definidos na seção 3.1.

Na história da evolução vários eventos introduzem mudanças nas seqüências do DNA. Alguns eventos biológicos típicos são as *substituições*, *remoções* e *inserções* de nucleotídeos. Portanto, qualquer comparação de seqüências precisa levar em consideração a possibilidade da ocorrência desses eventos, se é esperado identificar uma alta similaridade entre duas seqüências. Procedimentos de alinhamento típicos tentam identificar que partes não mudam e onde se localizam esses eventos biológicos. Após, apresentam um alinhamento ótimo de acordo com algum critério de otimização e sistema de pontuação associado aos eventos.

Alinhamentos podem ser associados a um conjunto de operações de edição que transformam uma seqüência em outra. Normalmente as únicas operações de edição consideradas são a *substituição* de um símbolo em outro, a *inserção* de um símbolo e a *remoção* de um símbolo. Se os custos são associados a cada operação, existe um procedimento de programação dinâmica clássico em $O(n^2)$[1] que computa o conjunto mínimo de operações de edição com o custo total mínimo e apresenta o alinhamento associado, que tem boa qualidade e alta semelhança para custos realistas.

Além desses eventos que resultam em alterações em um único símbolo na seqüência biológica, iremos considerar também, para a obtenção de alinhamentos, outros eventos, os rearranjos, que agem sobre um fragmento da seqüência. Consideraremos também, além das inserções, remoções e substituições de um símbolo, os eventos de inversão e de duplicação para a obtenção de um alinhamento ótimo de duas seqüências. Estes não são os únicos tipos de rearranjos, mas são os mais importantes.

A seção 1.1 descreve algumas mutações que alteram as seqüências biológicas e que pretendemos analisar nos alinhamentos. Esta seção contém muitos conceitos e definições que foram obtidos de [33].

O capítulo 2 contém definições e algoritmos que serão utilizados neste texto. A seção 2.1 fornece conceitos básicos e elementares em combinatória

[1]Nas análises das complexidades dos algoritmos, consideraremos neste texto que n é o comprimento da maior seqüência analisada.

das palavras e apresenta notações que serão utilizadas neste texto. As seções 2.2 e 2.3 mostram alguns tipos de matrizes (matrizes monotônicas e matrizes totalmente monotônicas) que foram utilizadas em [5], e que têm propriedades que auxiliam na solução do problema da obtenção dos elementos máximos de cada coluna de uma matriz. Trazem também algoritmos apresentados por Aggarwal et al. [2] que resolvem este problema para estes tipos de matrizes. Estes algoritmos são utilizados nos algoritmos propostos para a obtenção de alinhamentos com inversões não sobrepostas. Na seção 2.4 é apresentado a matriz de monge, que é um tipo de matriz com muitas propriedades descritas em [12] e que serão utilizadas, novamente, nos algoritmos propostos para alinhamentos com inversões não sobrepostas. A seção 2.5 contém as definições das operações de edição sobre seqüências. Estas operações de edição serão utilizadas como uma representação dos eventos biológicos que alteram as seqüências biológicas e que pretendemos detectar nos alinhamentos de seqüências. Na seção 2.6 é apresentado o conceito de grafo de edição, no qual são baseados os algoritmos de alinhamentos propostos.

O capítulo 3 é sobre alinhamentos usuais de duas seqüências, ou seja alinhamentos que não consideram duplicações nem inversões. Na seção 3.1, descreveremos e definiremos os alinhamentos usuais de duas seqüências. Muitos conceitos desta seção foram obtidos de [50] e [32]. Nas seções 3.2 e 3.3, descreveremos como os alinhamentos usuais se relacionam com operações de edição e grafos de edição, respectivamente. Na seção 3.4 descreveremos e daremos algumas noções sobre alguns tipos de alinhamentos usuais (local, global e semiglobal).

O capítulo 4 é sobre alinhamentos com inversões, ou seja alinhamentos que consideram também os eventos de inversão. A seção 4.1 traz uma introdução e um histórico sobre alinhamentos com inversões e uma definição de alinhamentos com inversões sem a restrição da não sobreposição. Já a seção 4.2 descreve e define um alinhamento com inversões não sobrepostas. As seções 4.3 e 4.4 são baseadas nos conceitos apresentados por Jeanette Schmidt [54]. Estas seções mostram uma estrutura de dados com árvores binárias, que armazenam as pontuações dos alinhamentos dos prefixos de uma seqüência contra todos os fatores de outra seqüência, e um algoritmo que constrói estas estruturas de dados. Estas estruturas serão utilizadas no algoritmo de alinhamento com inversões não sobrepostas da seção 4.5. A seção 4.5 mostra um algoritmo que propomos em [4] para resolver o problema da obtenção da pontuação de um alinhamento ótimo com inversões não sobrepostas em tempo $O(n^3 \log n)$ e espaço $O(n^2)$. Esse mesmo pro-

blema é resolvido pelo algoritmo que propomos em [62] e que apresentamos na seção 4.6. Este algoritmo, para os sistemas de pontuação comumente utilizados em alinhamentos, leva tempo $O(n^3)$ e utiliza espaço $O(n^2)$ para executar.

O capítulo 5 é sobre alinhamentos com duplicações, ou seja alinhamentos que consideram os eventos de duplicação. A seção 5.1 traz uma introdução sobre alinhamentos com duplicações e uma definição de alinhamentos com duplicações, sem a restrição das duplicações precisarem ser encadeadas (*tandem*). A seção 5.2 mostra as recorrências para construir as matrizes que serão utilizadas no algoritmo proposto, e apresentado nesta seção, para obter a pontuação de um alinhamento ótimo com duplicações. O algoritmo que obtém a pontuação de um alinhamento ótimo com duplicações apresentado nesta seção executa em tempo $O(n^3)$ e utiliza espaço $O(n^2)$. As seções 5.3 e 5.4 são equivalentes às seções 5.1 e 5.2, porém são para alinhamentos com duplicações em *tandem*.

O capítulo 6 traz uma conclusão sobre os algoritmos apresentados neste texto e no capítulo 7 são apresentados alguns trabalhos e problemas com os quais há interesse em se trabalhar no futuro.

1.1 Um pouco de biologia: mutações

Durante o decorrer do tempo algumas seqüências de DNA podem sofrer alterações. Estas alterações nas seqüências de DNA chamamos de *mutações*. As mutações são um tipo de evento biológico que ocorre devido a ação de agentes mutagênicos, que podem ser físicos (por exemplo radiação ionizante e raios ultra-violeta), químicos (por exemplo substâncias cancerígenas) e biológicos (por exemplo vírus e bactérias). Também podem haver mutações por falhas ocasionais (ou pelo menos desconhecidas), por exemplo algumas mutações ocorridas no processo de divisão das células. Alguns agentes mutagênicos apesar do efeito nocivo às células humanas, são aproveitados pela ciência e algumas vezes utilizado na medicina para diagnósticos e tratamentos de doenças.

Muitas vezes as células que sofreram uma mutação no seu código genético não persistem e morrem (ou param de metabolizar) sem replicar a mutação. Porém, outras vezes as mutações na seqüência de DNA de uma célula não interferem na sua sobrevivência ou até provocam a apresentação de novas características que melhoram a sua sobrevivência. Isto possibilita que esta

mutação seja propagada através da divisão celular.

Vamos classificar as mutações em dois grupos distintos:

Mutações pontuais : são mutações que alteram a seqüência de DNA em poucas bases ou apenas uma base. Neste trabalho, vamos considerar para este grupo de mutações somente as mutações que alteram as seqüências em apenas uma base. As principais mutações deste grupo são:

- **substituição** : neste tipo de mutação um nucleotídeo é substituído por outro na seqüência de DNA. Se a mudança envolve uma purina e uma pirimidina então este tipo de mutação é chamado de transversão, senão é chamado de transição;
- **inserção** : neste tipo de mutação uma base de nucleotídeo é inserida na seqüência de DNA;
- **remoção** : neste tipo de mutação uma base de nucleotídeo é removida da seqüência de DNA;

Mutações por rearranjo : são mutações que alteram a seqüência de DNA em várias bases. As principais mutações deste grupo são:

- **inversão** : neste tipo de mutação um fragmento da seqüência de DNA é envolvido. Os nucleotídeos deste fragmento tem a sua ordem alterada para a reversa e são trocados por seus nucleotídeos complementares, ou seja, quando uma inversão ocorre, um fragmento da seqüência é transformado no seu complemento reverso. O nucleotídeo complementar da adenina (A) é a timina (T) e vice versa. O nucleotídeo complementar da guanina (G) é a citosina (C) e vice versa. Por exemplo, o complemento reverso do fragmento $ACTG$ é $CAGT$. A figura 1.1 mostra o que acontece numa inversão de uma seqüência.
- **duplicação** : neste tipo de mutação um fragmento de DNA de uma seqüência é copiado e inserido na própria seqüência;
- **excisão** : neste tipo de mutação um fragmento de DNA, o qual foi inserido na seqüência de DNA anteriormente, é removido. Consideraremos neste trabalho que o fragmento inserido foi devido a uma mutação de duplicação.

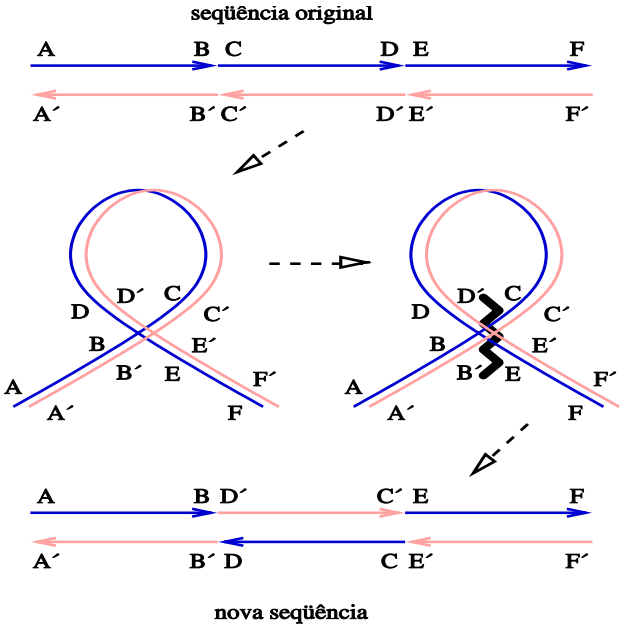

Figura 1.1: Exemplo de inversão numa seqüência de DNA

Em organismos multicelulares, as mutações presentes em células reprodutivas (gametas) de um indivíduo são propagadas para os descendentes deste indivíduo, podendo até gerar novas espécies. Em algumas plantas, as mutações sofridas no DNA de células somáticas (não reprodutivas) podem também ser transmitidas para seus dependentes.

As mutações em seqüências de DNA afetam não somente as seqüências de DNA, mas também as de RNA e as de aminoácidos de uma proteína.

As mutações atuam de forma crucial na evolução das espécies.

1.1.1 Repetições

Repetições são elementos preponderantes, especialmente em genomas de organismos eucariontes. Estima-se que mais de 80% dos genomas de planta são compostos por repetições. Existem diversos tipos de repetições em um genoma, provavelmente nem todos já conhecidos. Entre os tipos mais conhecidos estão os satélites (micro ou mini conforme as características de comprimento e outras) que são repetições em *tandem*, isto é, que aparecem uma atrás da outra ao longo do genoma. Outro tipo de repetição muito conhecido

são os ditos elementos transponíveis, ou transposons. Os transposons foram descobertos por Barbara McClintock [17] nos anos 50 estudando o milho. Os elementos transponíveis podem ser definidos como seqüências de DNA moderadamente repetitivas que podem mover-se de um local a outro no genoma e, desta maneira, ter um profundo impacto na estrutura, regulação e função dos genes, bem como na organização dos cromossomos na espécie.

Ser capaz de identificar repetições de maneira sistemática, e portanto exata, dado uma certa definição de repetição, é um problema importante em bioinformática que ainda não foi resolvido de maneira satisfatória ou realmente eficaz, especialmente no caso de genomas de organismos eucariontes. A dificuldade já vem da grande variedade de tipos de repetições. Algumas repetições, como as compridas (onde cada cópia pode atingir centenas de bases), são particularmente difíceis de identificar, por causa do comprimento, e porque há pouca conservação de uma cópia para a outra. Obviamente, o tamanho habitual das seqüências onde as repetições devem ser identificadas, aumenta ainda mais o grau de complexidade do problema.

Algumas doenças humanas são associadas às repetições, tais como: retardação mental *fragile-X* [63], doença de Huntington [1], distrofia miotônica [29] e ataxia de Friedreich [13]. *Tandem repeats* podem estar ligados a regras de regulação gênica [34, 44, 49], ligação DNA-proteína [52, 67] e evolução [36].

O número de cópias num *tandem repeat* pode ser variável entre indivíduos diferentes (polimórfico). Locais polimórficos são úteis em várias tarefas de laboratório [23, 66]. *Tandem repeats* tem sido utilizados para sustentar algumas hipóteses da evolução humana [7, 61] e da evolução de micro-satélites (*tandem repeats* cujo tamanho é de apenas algumas unidades de nucleotídeos) em primatas [46].

Capítulo 2

Definições Gerais

2.1 Combinatória das palavras

Seja Σ um *alfabeto*, um conjunto de *letras*. Qualquer seqüência finita em Σ é também chamada uma *palavra em* Σ ou simplesmente uma *palavra* se o alfabeto for claro. Sejam Σ^* o conjunto de todas as palavras em Σ, incluindo a *palavra vazia* denotada por 1 e Σ^+ o conjunto de todas as palavras em Σ, excluindo a *palavra vazia*. Nós identificamos palavras de comprimento 1 às letras que elas contém. A concatenação $\cdot$ de palavras é uma operação associativa definida sobre Σ^* e será freqüentemente omitida. Seja $w = w[1]w[2]\dots w[k]$ uma palavra. Denotamos por $|w|$ o *comprimento* k de w. Iremos também denotar a palavra w por $w[1\,..\,k]$. Para $1 \leq i \leq j \leq k$, o fator $w[i]w[i+1]\cdots w[j]$ de w é também representado por $w[i\,..\,j]$. Denotaremos $w[i\,..\,j] = 1$ se $i > j$. Seja $x, y, z \in \Sigma^*$. Denotaremos por $x\Sigma^*$ o conjunto $\{xy \mid y \in \Sigma^*\}$, denotaremos por Σ^*x o conjunto $\{yx \mid y \in \Sigma^*\}$ e denotaremos por $\Sigma^*x\Sigma^*$ o conjunto $\{yxz \mid y, z \in \Sigma^*\}$. Diremos que x é um *prefixo de* w se $w \in x\Sigma^*$, diremos que x é um *sufixo de* w se $y \in \Sigma^*x$ e diremos que x é um *fator de* w se $w \in \Sigma^*x\Sigma^*$.

Seja $\_$, também chamada de *inversão*, uma operação qualquer em Σ^* que satisfaça as seguintes propriedades:

1. $\overline{a} \in \Sigma, \forall a \in \Sigma$
2. $\overline{x \cdot y} = \overline{y} \cdot \overline{x}, \forall x, y \in \Sigma^*$

Note que a operação de inversão em Σ^* é definida por seus valores nas letras de Σ. Por exemplo, seja $\Sigma = \{A, C, T, G\}$ e seja $s \in \Sigma^*$ uma seqüência

de DNA qualquer. Se a inversão é definida por $\overline{A} = A$, $\overline{T} = T$, $\overline{C} = C$ e $\overline{G} = G$, ela mapeia s à sua *seqüência reversa*. Por outro lado, se a inversão é definida por $\overline{A} = T$, $\overline{T} = A$, $\overline{C} = G$ e $\overline{G} = C$, ela mapeia s à sua *seqüência reversa complementar*. O último caso é o de interesse para seqüências de DNA em biologia molecular e é o que será considerado nos exemplos e testes deste texto.

2.2 Matriz monotônica

Diremos que M é uma *matriz* $n \times m$ se cada par (i, j), $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$, que chamaremos de posição (i, j), contém um elemento. Dizemos que $M[i, j]$ é o *elemento da matriz* M que está na linha i e na coluna j, ou seja, na posição (i, j).

Seja M uma matriz $n \times m$. Dizemos que um conjunto de posições da matriz é uma *região da matriz*.

Definição 2.1 (Região de (i_1, j_1) a (i_2, j_2)) *Dados i_1, j_1, i_2 e j_2, dizemos que o conjunto de posições $\{(i, j) \mid i_1 \leq i \leq i_2$ e $j_1 \leq j \leq j_2\}$, denotado por $[i_1 \,..\, i_2, j_1 \,..\, j_2]$, é a região de (i_1, j_1) a (i_2, j_2).*

Definição 2.2 (Submatriz) *Sejam M uma matriz $n \times m$ e as seqüências ordenadas $X = (x_1, x_2, \ldots, x_{n'})$ e $Y = (y_1, y_2, \ldots, y_{m'})$ de índices das linhas e das colunas, respectivamente, de M. Seja M' a matriz $n' \times m'$ definida por $M'[i, j] = M[x_{i+1}, y_{j+1}]$, para toda posição (i, j) tal que $0 \leq i < n'$ e $0 \leq j < m'$. Diremos que a matriz M' é uma submatriz de M restrita às linhas de X e às colunas de Y.*

Nesta seção, e na próxima, analisaremos e trabalharemos com matrizes que possuem algumas propriedades específicas, que possibilitam encontrar de forma mais eficaz os elementos que são os máximos de cada coluna destas matrizes, a ser visto no Algoritmo 0 e no Algoritmo 3.

Definição 2.3 (Matriz comparável nas colunas) *Dizemos que a matriz M é uma matriz comparável nas colunas se para cada coluna de M, todos os seus elementos admitem uma ordem total, ou seja, dois elementos quaisquer de uma mesma coluna podem ser comparados.*

Em outras palavras, em uma matriz comparável nas colunas, todos os elementos de uma mesma coluna aceitam as operações de $<$ e $=$ entre eles.

Nesta seção consideraremos que a matriz M é uma matriz $n \times m$ comparável nas colunas.

Diremos que V é um *vetor* de comprimento n se para cada índice i, $0 \leq i \leq n-1$, $V[i]$ é definido. Dizemos que $V[i]$ é o elemento do vetor V que está na posição i.

Definição 2.4 (Vetor I_M de máximos das colunas) *Dada uma matriz M, definimos o vetor I_M da seguinte forma: $I_M[j]$ é o menor índice i tal que $M[i,j]$ é máximo na coluna j. Chamamos o vetor I_M de vetor de máximos das colunas de M.*

Por abuso de linguagem, diremos que o vetor I_M contém os índices das linhas dos elementos máximos de cada coluna, muito embora possa haver outra linha na coluna j, além de $I_M[j]$, que contenha este mesmo elemento máximo.

Observe que para todo i', tal que $0 \leq i' < I_M[j]$, temos que $M[i',j] < M[I_M[j],j]$ e para todo i'', tal que $I_M[j] < i'' < n$, temos que $M[i'',j] \leq M[I_M[j],j]$.

Por exemplo, para a matriz M do exemplo 2.5 o vetor I_M é igual ao vetor $[4,2,1,4,5,3]$ e os elementos máximos de cada coluna estão destacados em negrito.

Exemplo 2.5 (Matriz com os elementos máximos destacados)

$$M = \begin{bmatrix} 5 & 7 & \mathbf{25} & 37 & 7 & 2 \\ 12 & \mathbf{13} & 9 & 28 & 6 & 2 \\ 1 & 8 & 14 & 40 & 9 & \mathbf{3} \\ \mathbf{14} & 5 & 12 & \mathbf{43} & 4 & 3 \\ 13 & 10 & 25 & 43 & \mathbf{10} & 3 \end{bmatrix}$$

Definição 2.6 (Matriz monotônica) *A matriz M é dita monotônica se I_M está em ordem crescente.*

Vale a pena ressaltar que I_M não precisa estar em ordem estritamente crescente, mas apenas crescente, ou seja os valores de I_M podem não ser distintos. De forma equivalente, podemos dizer que uma matriz M é monotônica se $I_M[j_1] \leq I_M[j_2]$, para quaisquer j_1 e j_2 tais que $0 \leq j_1 < j_2 \leq m-1$.

Por exemplo, para a matriz M do exemplo 2.7 o vetor I_M é igual ao vetor $[1,2,2,4,5,5]$ e portanto a matriz é monotônica. Já para a matriz M do

exemplo 2.5 o vetor I_M é igual a $[4, 2, 1, 4, 5, 3]$ e portanto a matriz não é monotônica.

No exemplo 2.7, os elementos máximos de cada coluna de M estão destacados em negrito.

Exemplo 2.7 (Matriz monotônica)

$$M = \begin{bmatrix} \mathbf{14} & 7 & 9 & 37 & 7 & 2 \\ 12 & \mathbf{13} & \mathbf{25} & 28 & 6 & 2 \\ 1 & 8 & 14 & 40 & 9 & 2 \\ 5 & 5 & 12 & \mathbf{43} & 4 & 2 \\ 13 & 10 & 25 & 43 & \mathbf{10} & \mathbf{3} \end{bmatrix}$$

É interessante reparar que para que uma matriz M seja monotônica, não é necessário que os elementos de diferentes colunas sejam comparáveis entre si. Nem tampouco é necessário haver qualquer relação entre os elementos de diferentes colunas.

O exemplo 2.8 a seguir é mostrado para ilustrar esta independência entre os elementos de diferentes colunas. Neste exemplo utilizamos a ordem lexicográfica para comparar os elementos da coluna 3. Já na coluna 4, as comparações são feitas utilizando o comprimento da palavra.

Assim, no exemplo 2.8 o vetor I_M é igual a $[1, 2, 2, 4, 5, 5]$ e portanto M é uma matriz monotônica.

Novamente, no exemplo 2.8 os valores destacados em negrito são os elementos máximos de cada coluna de M.

Exemplo 2.8 (Matriz monotônica com tipos diferentes)

$$M = \begin{bmatrix} \mathbf{14} & 7 & a & \texttt{casado} & 7 & 2 \\ 12 & \mathbf{13} & \mathbf{e} & \texttt{viúvo} & 6 & 2 \\ 1 & 8 & c & \texttt{noivo} & 9 & 2 \\ 5 & 5 & b & \mathbf{\texttt{solteiro}} & 4 & 2 \\ 13 & 10 & d & \texttt{namorado} & \mathbf{10} & \mathbf{3} \end{bmatrix}$$

Definição 2.9 (Elemento morto) *Uma posição (i, j) de uma matriz M é dita morta se $I_M(j) \neq i$. Por abuso de linguagem, diremos que o elemento $M[i, j]$ é morto.*

Definição 2.10 (Região morta) *Dizemos que as regiões da matriz que só contêm posições mortas são regiões mortas.*

Proposição 2.11 *Seja M uma matriz $n \times m$ monotônica. Se $I_M[j] = i$ então as regiões $[0\,..\,i-1, j\,..\,m-1]$ e $[i+1\,..\,n-1, 0\,..\,j]$ de M são regiões mortas.*

Prova. Suponha que $I_M[j] = i$. Como M é monotônica, temos que $I_M[j'] \leq i$ para qualquer j' tal que $0 \leq j' < j$. Portanto as posições de M cujas linhas têm índices maiores que i e cujas colunas tem índices menores que j são mortas e $[i+1\,..\,n-1, 0\,..\,j-1]$ é uma região morta de M. De forma análoga, as posições cujas linhas têm índices menores que i e cujas colunas tem índices maiores que j são mortas e $[0\,..\,i-1, j+1\,..\,m-1]$ é uma região morta de M. Como as posições da coluna com índice j, exceto a da linha com índice i, são mortas temos que $[0\,..\,i-1, j\,..\,m-1]$ e $[i+1\,..\,n-1, 0\,..\,j]$ são regiões mortas de M. ∎

Problema 2.12 (Máximo das colunas) *Dada uma matriz M, queremos encontrar I_M.*

Se M for uma matriz monotônica o problema 2.12 da obtenção do máximo das colunas de M pode ser resolvido utilizando um algoritmo recursivo com a estratégia de divisão e conquista, como mostrado no Algoritmo 0 [2].

Algoritmo 1 Coloca em $I_M(j)$ o índice da linha do máximo da coluna j

```
maxMonotonica(M, i1, i2, j1, j2, I_M)
 1  ▷ Põe em I_M os índices dos máximos da região [i1..i2, j1..j2]
 2  se (i1 ≤ i2) e (j1 ≤ j2) então
 3        j ← ⌊(j1 + j2)/2⌋
 4        ▷ Coloca em I_M[j] a linha do 1º máximo da coluna j
 5        I_M[j] ← i1
 6        para i de i1 + 1 até i2 faça
 7              se M[i, j] > M[I_M[j], j] então
 8                    I_M[j] ← i
 9        maxMonotonica(M, i1, I_M[j], j1, j − 1, I_M)
10        maxMonotonica(M, I_M[j], i2, j + 1, j2, I_M)
```

Sejam $n = i_2 - i_1 + 1$ e $m = j_2 - j_1 + 1$. O tempo de execução do Algoritmo 0 é dado pela seguinte recorrência:

- $t(0, m) = t(n, 0) = c$, com c constante e

- $t(n,m) \leq n + \max_{1 \leq i \leq n}(t(i, \lfloor (m-1)/2 \rfloor) + t(n-i+1, \lceil (m-1)/2 \rceil))$, para $n > 0$ e $m > 0$.

Resolvendo esta recorrência temos que $t(n,m) = O(n \log m)$.

O Algoritmo 0 aloca memória somente para a pilha da chamada recursiva, ou seja, ele aloca memória $O(\log n)$. Vale a pena ressaltar que além desta alocação de memória, o algoritmo necessita de acesso ao vetor de resposta I_M e à matriz M.

Aggarwal et al. provaram em 1987 [2] que qualquer algoritmo que resolva o problema de encontrar os máximos das colunas em uma matriz monotônica, levando em consideração somente a propriedade da monotonicidade da matriz, necessita no pior caso de tempo $\Omega(n \log m)$.

O Algoritmo 0 está correto pois, para a coluna $j = \lfloor (j_1 + j_2)/2 \rfloor$ todos os seus elementos são verificados na linha 7 e é colocado em $I_M[j]$ o índice i do elemento máximo desta coluna j. Para as colunas $j' \neq j$ as chamadas recursivas das linhas 9 e 10 encontram os seus respectivos elementos máximos e colocam os seus índices em $I_M[j']$. As chamadas recursivas das linhas 9 e 10 não verificam os elementos das regiões $[0\,..\,i{-}1, j\,..\,m{-}1]$ e $[i{+}1\,..\,n{-}1, 0\,..\,j]$ de M, as quais de acordo com a proposição 2.11 são mortas e portanto não precisam ser verificadas.

A figura 2.1 ilustra o funcionamento do Algoritmo 0.

2.3 Matriz totalmente monotônica

Nesta seção consideraremos que a matriz M é uma matriz $n \times m$ comparável nas colunas.

Definição 2.13 (Matriz totalmente monotônica) *Dizemos que M é uma matriz totalmente monotônica se toda submatriz de M é monotônica.*

Lembrando que estamos considerando como submatriz de M a restrição de M aos seus elementos que pertencem a uma subseqüência de linhas e a uma subseqüência de colunas de M. Ressaltando que as linhas, assim como as colunas, não precisam ser consecutivas mas sim ordenadas. Para que toda submatriz de M seja monotônica, basta verificar que toda submatriz 2×2 de M seja monotônica, ou seja, para qualquer i_1, i_2 , j_1 e j_2 tal que $0 \leq i_1 < i_2 \leq n-1$ e $0 \leq j_1 < j_2 \leq m-1$, se $M[i_1, j_1] < M[i_2, j_1]$ então $M[i_1, j_2] < M[i_2, j_2]$.

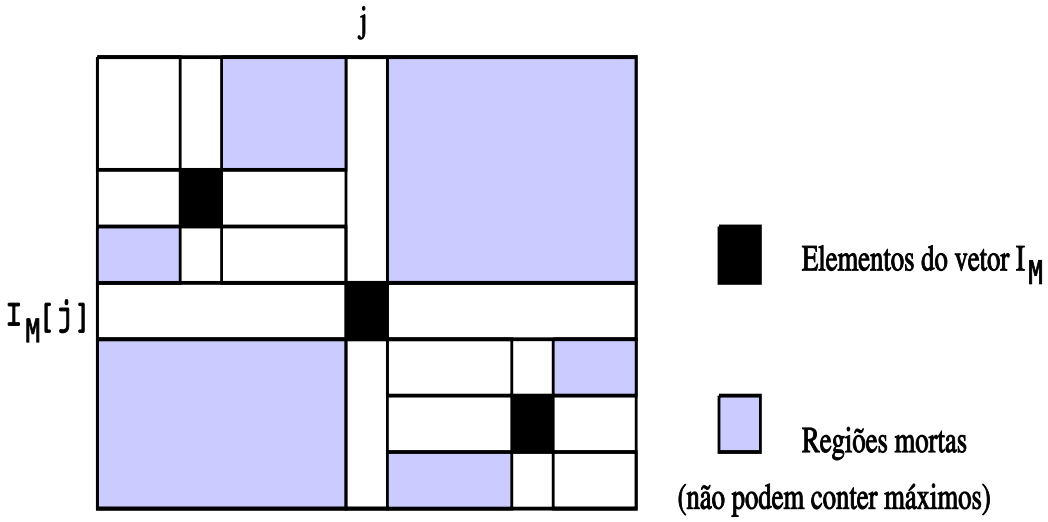

Figura 2.1: Após encontrar cada $I_M[j]$ são identificadas duas regiões mortas destacadas na figura e que não são inspecionadas pelo algoritmo.

O exemplo 2.14 mostra uma matriz totalmente monotônica. Os valores destacados em negrito são os elementos $M[I_M[j], j]$.

Exemplo 2.14 (Matriz totalmente monotônica)

$$M = \begin{bmatrix} \mathbf{15} & 7 & 33 & 11 & 70 & 15 \\ 14 & \mathbf{10} & \mathbf{51} & 12 & 71 & 16 \\ 13 & 8 & 40 & 17 & 72 & 17 \\ 12 & 9 & 41 & \mathbf{20} & 79 & 23 \\ 11 & 6 & 13 & 5 & \mathbf{98} & \mathbf{31} \end{bmatrix}$$

Proposição 2.15 *Seja M uma matriz $n \times m$ totalmente monotônica e sejam i_1 e i_2 tais que $0 \leq i_1 < i_2 < n$. Então:*

a) $M[i_1, j] \geq M[i_2, j]$ implica que, para todo j' tal que $0 \leq j' \leq j$, a posição (i_2, j') é morta em M.

b) $M[i_1, j] < M[i_2, j]$ implica que, para todo j' tal que $j \leq j' < m$, a posição (i_1, j') é morta em M.

Prova. a) *Se* $M[i_1, j] \geq M[i_2, j]$ *então* $\forall\ j'$ *tal que* $0 \leq j' \leq j$ *a posição* (i_2, j') *é morta.*

Se $j' = j$ então $I_M[j'] \neq i_2$, pois por hipótese $M[i_1, j] \geq M[i_2, j]$ e portanto a posição (i_2, j') é morta. Para $0 \leq j' < j$ suponha, por absurdo, que a posição (i_2, j') não é morta e portanto $I_M[j'] = i_2$. Sendo assim, teríamos $M[i_1, j'] < M[i_2, j']$ e a submatriz de M restrita aos elementos das linhas i_1 e i_2 e das colunas j' e j não seria monotônica. Portanto M não seria totalmente monotônica. Logo a suposição que $M[i_2, j']$ não é um elemento morto está incorreta.

b) *Se* $M[i_1, j] < M[i_2, j]$ *então* $\forall\ j'$ *tal que* $j \leq j' < m$ *a posição* $M[i_1, j']$ *é morta.*

Se $j' = j$ então $I_M[j'] \neq i_1$ pois por hipótese $M[i_1, j] < M[i_2, j]$ e portanto a posição (i_1, j') é morta. Para $j < j' < m$ suponha, por absurdo, que a posição (i_1, j') não é morta e portanto $I_M[j'] = i_1$. Sendo assim, teríamos $M[i_1, j'] \geq M[i_2, j']$ e a submatriz de M restrita aos elementos das linhas i_1 e i_2 e das colunas j' e j não seria monotônica. Portanto M não seria totalmente monotônica. Logo a suposição que $M[i_1, j']$ não é um elemento morto está incorreta. ∎

A figura 2.2 mostra as posições mortas para cada caso da Proposição 2.15.

Se todas as posições (i, j) de uma linha i são mortas, então dizemos que a linha i é morta.

Corolário 2.16 *Seja* M *uma matriz* $n \times m$ *totalmente monotônica e sejam* i_1 *e* i_2 *tais que* $0 \leq i_1 < i_2 \leq n - 1$. *Se* $M[i_1, m-1] \geq M[i_2, m-1]$ *então a linha* i_2 *é morta.*

Seja M uma matriz $n \times m$ totalmente monotônica e $n \geq m$. Como $n \geq m$ então existem pelo menos $n - m$ linhas em M cujo índice não é um elemento do vetor I_M de máximos de colunas de M, ou seja, existem pelo menos $n - m$ linhas mortas. O Algoritmo 2 remove da matriz totalmente monotônica M $n - m$ destas linhas mortas. Portanto, o Algoritmo 2 devolve uma submatriz C de M restrita a m linhas de M, tal que todas as linhas cujos índices estão em I_M são linhas de C.

Lema 2.17 *O Algoritmo 2 está correto e executa em tempo* $O(n)$.

Prova.

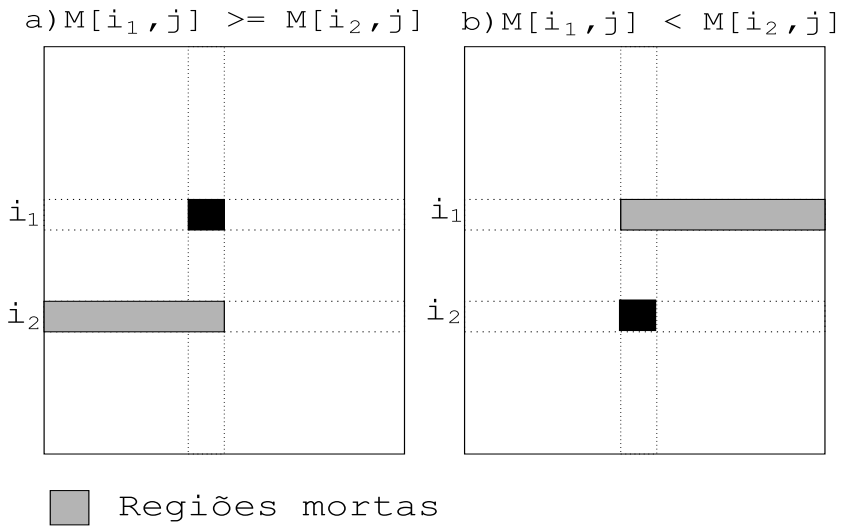

Figura 2.2: a) Se $M[i_1, j] \geq M[i_2, j]$ então a posição (i_2, j') é morta, $0 \leq j' \leq j$. b) Se $M[i_1, j] < M[i_2, j]$ então a posição (i_1, j') é morta, $j \leq j' \leq m - 1$.

Algoritmo 2 Retira $n - m$ linhas da matriz M que não contêm máximos de colunas

```
REDUCAO(M)
 1  C ← M
 2  k ← 0
 3  m ←nº de colunas de C
 4  enquanto nº de linhas de C > m faça
 5          se C[k,k] ≥ C[k+1,k] então
 6                  ▷ C[k+1,j] é um elemento morto ∀ j | 0 ≤ j ≤ k
 7                  se k < m − 1 então
 8                          k ← k + 1
 9                  senão
10                          Remove linha k + 1 de C
11          senão
12                  ▷ C[k,k] < C[k+1,k] e portanto
13                  ▷ C[k,j] é um elemento morto ∀ j | k ≤ j < m
14                  Remove linha k de C
15                  se k > 0 então
16                          k ← k − 1
17  devolva C
```

1) O Algoritmo 2 está correto.

   O laço da linha 4 mantém as seguintes invariantes:

   (a) o conjunto de posições $\Upsilon = \{(i, j) \mid 0 \leq i \leq k \text{ e } 0 \leq j < i\}$ contém somente posições mortas de M e

   (b) qualquer linha cujo índice é um elemento de I_M é uma linha de C.

   A figura 2.3 mostra o conjunto de posições mortas Υ da invariante 1a.

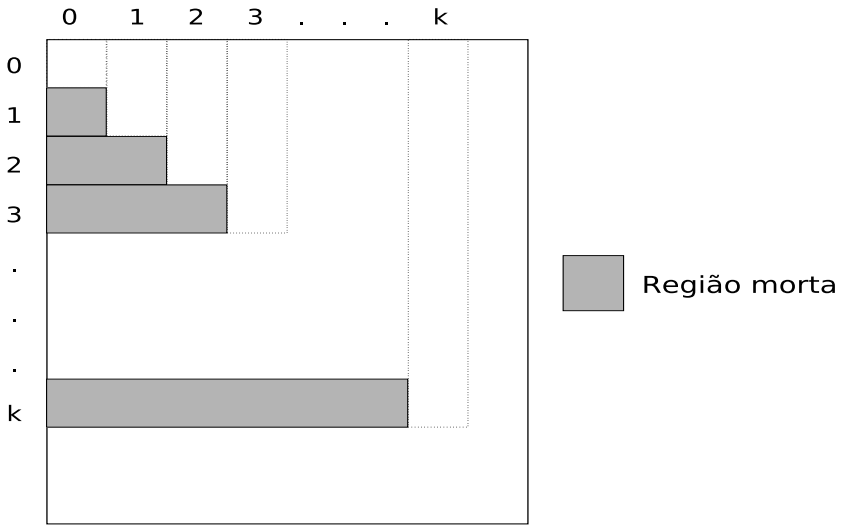

Figura 2.3: Conjunto de posições mortas Υ da invariante 1a do Algoritmo 2.

No início do laço, ou seja na primeira execução da linha 4, a invariante 1a é verdadeira, pois para $k = 0$ o conjunto Υ é vazio. No início do laço a invariante 1b também é verdadeira pois $C = M$.

Só podem ser inseridos elementos no conjunto Υ quando o valor de k é aumentado, ou quando muda a linha indexada por k. Isto ocorre somente nas linhas 8 e 14 do algoritmo. Na linha 8 do algoritmo o valor de k é incrementado e as posições inseridas em Υ são garantidas serem mortas pela afirmação a) da Proposição 2.15. Na execução da linha 14 do algoritmo, a linha de C indexada por k é trocada, porém a nova linha indexada é a linha zero de C ou será alterada, na linha 16 do algoritmo, para a linha $k - 1$, o que em nenhuma das

duas hipóteses insere elementos novos em Υ. Portanto a invariante 1a é mantida verdadeira durante a execução do laço.

A invariante 1b é mantida verdadeira durante a execução do laço pois somente são retiradas linhas mortas de C. A linha $k+1$ retirada de C na linha 10 do algoritmo é garantida ser morta pelo Corolário 2.16. A linha k retirada de C na linha 14 do algoritmo é garantida ser morta pois, pela invariante 1a as posições (k, j) são mortas, para todo j tal que $0 \leq j < k$, e as outras posições da linha k, ou seja as posições (k, j) para todo j tal que $k \leq j < m$, são garantidas serem mortas pela afirmação b) da Proposição 2.15.

O número de linhas de C ou decresce ou fica igual. Ele só não decresce quando k é incrementado, porém a linha 7 do algoritmo garante que k só é incrementado até o limite superior m. Logo, como no início da execução do algoritmo o número de linhas de C é maior ou igual ao número de colunas, então o número de linhas de C atingirá o valor de m e a condição do laço na linha 4 do algoritmo será insatisfeita e o algoritmo será encerrado com a matriz C tendo m linhas. Portanto o algoritmo sempre termina com C tendo m linhas.

Como as únicas alterações realizadas em C são remoções de linhas, temos que a matriz C é uma submatriz de M restrita a m linhas de M.

Além disso, a invariante 1b garante que as linhas de M cujos índices são elementos de I_M são linhas de C.

Portanto o Algoritmo 2 executa corretamente.

2) O Algoritmo 2 executa em tempo $O(n)$.

O tempo de execução do algoritmo é obtido através do número de vezes t que o laço 4 é executado. Esse número t é dado pela seguinte equação:

$$t = a + b + c,$$

onde a, b e c são o número de vezes que as linhas 8, 10 e 14, respectivamente, são executadas[1].

[1] A matriz C pode ser construída com referências às linhas de M e portanto a operação de remoção de uma linha pode ser feita em tempo constante.

O número de linhas removidas é igual a $b + c$. Sabemos também que o número de linhas removidas é igual a $n - m$, portanto

$$b + c = n - m.$$

Seja k_f o valor de k quando o algoritmo termina e k_i o valor de k quando o laço do algoritmo inicia. Temos que $k_f - k_i$ é igual ao número de vezes que k é incrementado menos o número de vezes que k é decrementado. k é incrementado a vezes e decrementado no máximo c vezes. Portanto

$$k_f - k_i \geq a - c.$$

Sabemos também que $k_f < m$ e que $k_i = 0$. Portanto $k_f - k_i \leq m - 1$ e $m - 1 \geq k_f - k_i \geq a - c$. Logo

$$m - 1 \geq a - c.$$

Utilizando estas equações temos que:

$$\begin{aligned}
t &= a + \underbrace{b + c}_{=n-m} \\
&= a + n - m \\
&= a + \underbrace{b - b + c - c}_{=0} + n - m \\
&= \underbrace{a - c}_{\leq m-1} + \underbrace{b + c}_{=n-m} + n - m - b \\
&\leq m - 1 + n - m + n - m - b \\
&\leq 2n - m - 1 \\
&= O(n)
\end{aligned}$$

Portanto o tempo de execução do Algoritmo 2 é $O(n)$. ∎

A matriz C pode ser construída através de referências às linhas de M. Portanto, a memória utilizada pelo Algoritmo 2 é $O(n)$.

Se M for uma matriz totalmente monotônica $n \times m$ o problema 2.12 da obtenção do máximo das colunas de M pode ser resolvido utilizando o

Algoritmo 3 Coloca em $I_M(j)$ o índice da linha do máximo da coluna j

```
maxTotMonotonica(M, I_M, A)
 1  ▷ A é uma submatriz da matriz original M. Na 1a. chamada A = M.
 2  ▷ O algoritmo encontra as linhas dos máximos da matriz A
 3  C ← REDUCAO(A)
 4  n ← nº de linhas de C
 5  se n = 1 então
 6          ▷ Para toda coluna j de C o máximo é a única linha de C
 7          m ← nº de colunas de C
 8          para j de 0 até m − 1 faça
 9                  j* ←índice em M da coluna j de C
10                  i* ←índice em M da linha de C
11                  I_M[j*] ← i*
12  senão
13          ▷ Cria uma matriz C' com as colunas pares de C
14          C' ← colunas pares de C
15          maxTotMonotonica(M, I_M, C')
16          ▷ Coloca em I_M os máximos das colunas ímpares de C
17          maxImpar(M, I_M, C)
```

Algoritmo 4 Coloca em I_M os índices das linhas dos máximos das colunas ímpares de uma matriz monotônica C

```
MAXIMPAR(M, I_M, C)
 1  ▷ C é uma submatriz da matriz original M
 2  ▷ As linhas dos máximos das colunas pares de C já estão em I_M
 3  n ← nº de linhas de C
 4  m ← nº de colunas de C
 5  para j de 1 até m − 1 passo 2 faça
 6          j_1* ←índice em M da coluna j − 1 de C
 7          i_1 ←índice em C da linha I_M[j_1*] de M
 8          se j < m − 1 então
 9                  j_2* ←índice em M da coluna j + 1 de C
10                  i_2 ←índice em C da linha I_M[j_2*] de M
11          senão
12                  i_2 ← n − 1
13          ▷ Obtém linha i_max que tem o máximo em j. (i_1 ≤ i_max ≤ i_2)
14          i_max ← i_1
15          para i de i_1 + 1 até i_2 faça
16                  se C[i, j] > C[i_max, j] então
17                          i_max ← i
18          j* ←índice em M da coluna j de C
19          i* ←índice em M da linha i_max de C
20          I_M[j*] ← i*
```

Algoritmo 3. O Algoritmo 3 foi construído, essencialmente, utilizando a estratégia da divisão e conquista e os Algoritmos 2 e 4.

O Algoritmo 4 é simples e a prova de sua corretude será omitida. Mostramos a seguir somente uma breve descrição da sua execução assim como uma breve análise do tempo da sua execução.

Pela fato de C ser uma matriz monotônica $n \times m$, o índice da linha do máximo de uma coluna ímpar j deve estar entre:

- $I_C[j-1]$ e $I_C[j+1]$ se $j < m-1$ ou entre
- $I_C[j-1]$ e $n-1$ se $j = m-1$.

Essencialmente, o algoritmo usa esta propriedade para encontrar o máximo da cada coluna ímpar da matriz C da seguinte forma: para cada coluna encontra o intervalo de linhas onde deve estar o máximo da coluna e percorre este intervalo, entre as linhas 14 e 17 do algoritmo, para obter o máximo.

Devido a esta restrição no intervalo de busca do máximo de cada coluna ímpar, o algoritmo executa a linha 15 no máximo $n-1$ vezes. Portanto, para $n \geq m$, o Algoritmo 4 é executado em tempo[2] $O(n)$. Vale a pena ressaltar que, tomando-se certos cuidados nas estruturas de dados utilizadas para representar uma submatriz de M, as operações de conversão de linhas e colunas de uma submatriz de M para M, e vice-versa, nos Algoritmos 3 e 4 podem ser executadas em tempo constante[3].

A seguir segue uma breve descrição do funcionamento do Algoritmo 3. Vamos considerar que a matriz M $n \times m$ é a matriz original para a qual queremos resolver o problema 2.12 e que $n \geq m$.

Na linha 3, o algoritmo obtém uma matriz C retirando algumas linhas mortas da matriz A (através do Algoritmo 2). Se o algoritmo estiver sendo executado pela chamada da linha 15 do próprio Algoritmo 3, ou seja, estiver sendo executado dentro da recursão, então a matriz A é $n \times \lceil \frac{n}{2} \rceil$ e a matriz C será $\lceil \frac{n}{2} \rceil \times \lceil \frac{n}{2} \rceil$.

Se a matriz C tem uma única linha, então a linha que tem os máximos das colunas é esta única linha e o algoritmo coloca em I_M, nas posições das

[2]Se $m \geq n$ então o Algoritmo 4 executa em tempo $O(m)$, ou seja, o Algoritmo 4 executa em tempo $O(n+m)$.

[3]Em nossa implementação representamos uma submatriz de M através de uma lista ligada de linhas, sendo que cada linha tem uma referência a linha correspondente da matriz M. As colunas de um submatriz de M foram restritas através de um número k, tal que uma coluna j da submatriz é igual à coluna $j * 2^k$ de M.

colunas de C, o índice desta linha. Vale a pena ressaltar que se o algoritmo estiver sendo executado por uma chamada recursiva, então há somente uma coluna em C e portanto, somente uma posição em I_M onde será colocado o índice da linha de C.

Se a matriz C tem mais do que uma linha, então o Algoritmo 3 obtém os valores dos máximos das colunas pares, fazendo uma chamada recursiva na linha 15 e utilizando a submatriz C' de C restrita às colunas pares. Para encontrar os máximos das colunas ímpares de C o Algoritmo 3 utiliza o Algoritmo 4 na linha 17.

A seguir mostramos a análise do tempo de execução do Algoritmo 3.

Seja m o número de colunas de A e n o número de linhas de A. Como estamos considerando que o número de linhas de M é maior que o número colunas de M e na chamada externa do algoritmo $A = M$, então podemos considerar que $n \geq m$.

Vimos que a linha 3 é executada em tempo $O(n)$. A linha 8 é executada no máximo m vezes. Vimos também que a linha 17 é executada em tempo $O(m)$. Seja $t(n, m)$ o tempo de execução do Algoritmo 3 então a linha 15 é executada em tempo $t(m, m/2)$.

Portanto o tempo de execução do Algoritmo 3 é dado pela seguinte recorrência:

$$t(n, m) \leq c_1 n + c_2 m + t(m, m/2),$$

onde c_1 e c_2 são constantes positivas e $n \geq m$. Resolvendo esta recorrência obtemos que $t(n, m) = O(n)$.

Portanto o tempo de execução do Algoritmo 3 é $O(n)$ e conseguimos resolver o Problema 2.12 em tempo $O(n)$ para matrizes totalmente monotônicas com $n \geq m$.

Segundo Aggarwal et al. [2] se $m > n$ então o Problema 2.12 pode ser resolvido em tempo $\Theta(n(1 + \log(m/n))$. Neste texto utilizaremos matrizes totalmente monotônicas onde $n \geq m$.

Assim como a matriz C pode ser construída com referências às linhas de A, a matriz C' pode ser construída com referências às colunas de C. Seja $e(n, m)$ o espaço em memória utilizado pelo Algoritmo 3. Temos que

$$e(n, m) \leq c_3 n + c_4 m + e(m, m/2),$$

onde c_3 e c_4 são constantes positivas. Resolvendo esta recorrência obtemos que $e(n, m) = O(n)$.

Logo, dados a matriz M e o vetor I_M, a memória consumida pelo Algoritmo 3 é $O(n)$.

2.4 Matriz de Monge

Definição 2.18 (Matriz de monge) *Uma matriz M $n \times m$ é dita uma matriz de monge se*

$$M[i_1, j_1] + M[i_2, j_2] \leq M[i_1, j_2] + M[i_2, j_1]$$

para todo i_1, i_2, j_1, j_2 tais que $0 \leq i_1 < i_2 \leq n-1$ e $0 \leq j_1 < j_2 \leq m-1$.

Algumas vezes a desigualdade é verdadeira na direção inversa. Neste caso temos uma matriz de monge inversa.

Definição 2.19 (Matriz de monge inversa) *Uma matriz M $n \times m$ é dita uma matriz de monge inversa se $M[i_1, j_1]+M[i_2, j_2] \geq M[i_1, j_2]+M[i_2, j_1]$ para todo i_1, i_2, j_1, j_2 tais que $0 \leq i_1 < i_2 \leq n-1$ e $0 \leq j_1 < j_2 \leq m-1$.*

Um caso particular, mas muito comum de matriz de monge, é a matriz de monge triangular superior (inferior). Neste tipo de matriz, para toda posição (i, j), tal que $i > j$ (ou $i < j$ se inferior), o elemento da posição (i, j) é $M[i, j] = +\infty$.[4] De forma análoga, definimos a matriz de monge inversa triangular superior e matriz de monge inversa triangular inferior, usando $M[i, j] = -\infty$ nas posições desprezadas.

Uma propriedade muitas vezes útil para a verificação de que uma determinada matriz é de monge, é que para que uma matriz seja uma matriz de monge basta que a desigualdade seja satisfeita para as linhas e colunas adjacentes, ou seja,

$$M[i, j] + M[i+1, j+1] \leq M[i, j+1] + M[i+1, j]$$

para todo i, j tal que $0 \leq i < n-1$ e $0 \leq j < m-1$. Propriedade análoga vale para matriz de monge inversa.

[4] Alguns autores preferem a seguinte definição para Matriz de monge triangular superior (inferior): uma matriz M $n \times m$ é dita uma matriz de monge triangular superior (ou inferior) se $M[i_1, j_1] + M[i_2, j_2] \leq M[i_1, j_2] + M[i_2, j_1]$ para todo i_1, i_2, j_1, j_2 tais que $0 \leq i_1 < i_2 \leq j_1 < j_2 \leq m-1$ (ou $0 \leq j_1 < j_2 \leq i_1 < i_2 \leq n-1$ se inferior).

Outra propriedade muito importante é que toda matriz de monge (ou monge inversa) é uma matriz totalmente monotônica. Isto é útil, principalmente, nos casos em que se deseja determinar os máximos das colunas de uma matriz de monge (ou monge inversa).

Sejam M e N duas matrizes $n \times m$ de monge (ou monge inversa). Sejam U um vetor de comprimento n e V um vetor de comprimento m. Seja c uma constante tal que $c \geq 0$. Seja M^T a matriz transposta de M. Podemos afirmar que as seguintes matrizes também são matrizes de monge (ou monge inversa):

- M^T,
- $M + N$,
- cM,
- $A \mid A[i, j] = M[i, j] + U[i] + V[j]$.

2.5 Operações de edição

Nesta seção consideraremos que Σ é um alfabeto qualquer e que $s = s[1 \,..\, n] \in \Sigma^*$ para $n \geq 0$. Nos exemplos consideraremos que $\Sigma = \{A, C, T, G\}$.

Definição 2.20 (Operação de edição) *Uma operação de edição é qualquer função parcial cujo domínio e contra-domínio são Σ^*.*

Em outras palavras, uma função de edição é uma função que quando aplicada em uma seqüência $s \in \Sigma^*$ gera uma seqüência $s' \in \Sigma^*$. Vale a pena ressaltar que as operações de edição sendo funções parciais podem não ser definidas para todos os elementos do seu domínio.

As operações de edição permitem representar o resultado das alterações sofridas nas seqüências biológicas diante de diferentes tipos de mutações.

A seguir, vamos relacionar para cada mutação, que desejamos analisar, uma operação de edição correspondente.

Inserção: Dados i e a tais que $0 \leq i$ e $a \in \Sigma$, dizemos que a função parcial $\iota_{i,a} : \Sigma^* \to \Sigma^*$ definida por $\iota_{i,a}(s) = s[1 \,..\, i]\ a\ s[i + 1 \,..\, n]$, para $i \leq n$, é uma *operação de edição de inserção*, ou mais especificamente a *operação de edição de inserção de a após a posição i*. Por exemplo,

$\iota_{5,G}(AACGCCTTCG) = AACGC\mathbf{G}CTTCG$. Diremos que i é o índice desta operação de edição e a é o símbolo envolvido nesta operação de edição. Dizemos que $\iota_{i,a}$ é uma operação de edição do tipo lacuna (ou *gap*). Chamaremos de *classe de operações de edição de inserção* o conjunto com todas as possíveis operações de edição de inserção. Vamos considerar que uma operação de edição de inserção corresponde a uma mutação de inserção.

Remoção: Dados i e a tais que $1 \leq i$ e $a \in \Sigma^*$, dizemos que a função parcial $\rho_{i,a} : \Sigma^* \to \Sigma^*$ definida por $\rho_{i,a}(s) = s[1\,..\,i-1]s[i+1\,..\,n]$, para $i \leq n$ e $s[i] = a$, é uma *operação de edição de remoção*, ou mais especificamente a *operação de edição de remoção da letra a na posição i*. Por exemplo, $\rho_{10,G}(AACGCCTTC\mathbf{G}) = AACGCCTTC$. Diremos que i é o índice desta operação de edição e a é o símbolo envolvido nesta operação de edição. Dizemos que $\rho_{i,a}$ é uma operação de edição do tipo lacuna (ou *gap*). Chamaremos de *classe de operações de edição de remoção* o conjunto com todas as possíveis operações de edição de remoção. Vamos considerar que uma operação de edição de remoção corresponde a uma mutação de remoção.

Substituição: Dados i, a e b tais que $1 \leq i$, $a \in \Sigma$ e $b \in \Sigma$, dizemos que a função parcial $\sigma_{i,a,b} : \Sigma^* \to \Sigma^*$ definida por $\sigma_{i,a,b}(s) = s[1\,..\,i-1]as[i+1\,..\,n]$, para $i \leq n$ e $s[i] = b$, é uma *operação de edição de substituição*, ou mais especificamente a *operação de edição de substituição da letra b na posição i por a*. Por exemplo, $\sigma_{1,G,A}(\mathbf{A}ACGCCTTCG) = \mathbf{G}ACGCCTTCG$. Diremos que i é o índice desta operação de edição e a e b são os símbolos envolvidos nesta operação de edição. Se $a = b$ então dizemos que $\sigma_{i,a,b}$ é uma operação de edição do tipo casamento (ou *match*). Se $a \neq b$ então dizemos que $\sigma_{i,a,b}$ é uma operação de edição do tipo erro (ou *mismatch*). Chamaremos de *classe de operações de edição de substituição* o conjunto com todas as possíveis operações de edição de substituição. Vamos considerar que uma operação de edição de substituição corresponde a uma mutação de substituição.

Inversão: Dados i e i' tais que $1 \leq i \leq i' + 1$, dizemos que a função parcial $\pi_{i,i'} : \Sigma^* \to \Sigma^*$ definida por $\pi_{i,i'}(s) = s[1\,..\,i-1]\overline{s[i\,..\,i']}s[i'+1\,..\,n]$, para $i' \leq n$, é uma *operação de edição de inversão*, ou mais especificamente, a *operação de edição de inversão do trecho de i a i'*. Por exemplo, $\pi_{4,7}(AAC\mathbf{GCCT}TCG) = AAC\mathbf{AGGC}TCG$. Repare que

$\overline{GCCT} = AGGC$, pois consideramos que a operação de inversão mapeia um trecho no seu reverso complementar. Diremos que i e i' são os índices desta operação de edição. Diremos que os símbolos nas posições de i a i', ou seja, nas posições do intervalo $[i, i']$, são os símbolos envolvidos nesta operação de edição e são substituídos pelo seu complementar reverso. Chamaremos de *classe de operações de edição de inversão* o conjunto com todas as possíveis operações de edição de inversão. Vamos considerar que uma operação de edição de inversão corresponde a uma mutação de inversão.

Duplicação: Dados i_1, i_2 e i_3 tais que $1 \leq i_1 \leq i_2$ e $0 \leq i_3$ dizemos que a função parcial $\delta_{i_1,i_2,i_3} : \Sigma^* \to \Sigma^*$ definida por $\delta_{i_1,i_2,i_3}(s) = s[1\,..\,i_3]s[i_1\,..\,i_2]s[i_3+1\,..\,n] = s'$, $i_2 \leq n$ e $i_3 \leq n$, é uma *operação de edição de duplicação*, ou mais especificamente a *operação de edição de duplicação do trecho de i_1 a i_2 após a posição i_3*. Diremos que, para esta operação de edição de duplicação a seqüência $s[i_1\,..\,i_2]$ é a *seqüência original* e a seqüência $s'[i_3{+}1\,..\,i_3{+}i_2{-}i_1{+}1]$ é a *cópia* ou *repetição*. Por exemplo, $\delta_{4,7,9}(AAC\mathbf{GCCT}TCG) = AACGCCTTC\mathbf{GCCT}G$. Diremos que i_1, i_2 e i_3 são os índices desta operação de edição. Chamaremos de *classe de operações de edição de duplicação* o conjunto com todas as possíveis operações de edição de duplicação. Vamos considerar que uma operação de edição de duplicação corresponde a uma mutação de duplicação.

Excisão: Dados i_1, i_2 e i_3 tais que $1 \leq i_1 \leq i_2$ e $0 \leq i_3$ dizemos que a função parcial $\varepsilon_{i_1,i_2,i_3} : \Sigma^* \to \Sigma^*$ definida por $\varepsilon_{i_1,i_2,i_3}(s) = s[1\,..\,i_1 - 1]s[i_2+1\,..\,n] = s'$ tal que $s'[i_3\,..\,i_3+i_2-i_1] = s[i_1\,..\,i_2]$, para $i_2 \leq n$ e $i_3 \leq n-(i_2-i_1+1)$, é uma *operação de edição de excisão*, ou mais especificamente a *operação de edição de excisão do trecho de i_1 a i_2 existente na posição i_3*. Diremos que, para esta operação de edição de excisão a seqüência $s[i_1\,..\,i_2]$ é a *cópia* ou *repetição* e a seqüência $s'[i_3\,..\,i_3+i_2-i_1]$ é a *seqüência original*. Por exemplo, $\delta_{10,13,4}(AACGCCTTC\mathbf{GCCT}G) = AAC\mathbf{GCCT}TCG$. Observe que no exemplo $\delta_{6,9,4}(AACGC\mathbf{GCCT}CTTCG) = AAC\mathbf{GCCT}TCG$ não existe um fragmento em s, exceto a repetição, que seja igual à seqüência original. Diremos que i_1, i_2 e i_3 são os índices desta operação de edição. Chamaremos de *classe de operações de edição de excisão* o conjunto com todas as possíveis operações de edição de excisão. Vamos

considerar que uma operação de edição de excisão corresponde a uma mutação de excisão.

Chamamos as operações de edição definidas acima de *operações de edição elementares*.

Seja γ uma operação de edição elementar. Se γ é uma operação de edição de inserção ou de remoção ou de substituição então γ é uma *operação de edição pontual*, senão γ é uma *operação de edição de rearranjo*.

Seja $\Psi = (\gamma_1, \gamma_2, \ldots, \gamma_k)$ tal que $k \geq 0$ e γ_i é uma operação de edição, para todo i onde $1 \leq i \leq k$. Diremos que Ψ é uma *seqüência de operações de edição* de comprimento k. Se $k = 0$ então Ψ é a seqüência de operações de edição vazia.

Assim, dado uma seqüência de operações de edição $\Psi = (\gamma_1, \gamma_2, \ldots, \gamma_k)$, dizemos que a função parcial $\phi_\Psi : \Sigma^* \to \Sigma^*$ definida por

$$\phi_\Psi(s) = \gamma_k(\gamma_{k-1}(\ldots\gamma_2(\gamma_1(s))\ldots)) = (\gamma_k \circ \gamma_{k-1} \circ \ldots \circ \gamma_2 \circ \gamma_1)(s)$$

é a *operação de edição composta de* Ψ.

Vale a pena ressaltar que, como uma operação de edição qualquer γ_i é uma função parcial, ϕ_Ψ é também uma função parcial.

Observação 2.21 *Seja Ψ uma seqüência de operações de edição elementares. Se $\phi_\Psi(s) = t$ então $\phi_\Psi(sx) = tx$, onde x é uma palavra qualquer.*

Seja γ uma operação de edição e $\gamma^{-1} : \Sigma^* \to \Sigma^*$ uma operação de edição tal que $\gamma^{-1}(\gamma(s)) = s$ para qualquer s onde $\gamma(s)$ é definida e $\gamma(\gamma^{-1}(s)) = s$ para qualquer s onde $\gamma^{-1}(s)$ é definida. Diremos que γ^{-1} é a *operação de edição inversa de* γ e que γ é uma *operação de edição inversível*. Podemos dizer que a operação de edição γ^{-1} desfaz a transformação de uma operação de edição γ e vice versa.

Vale a pena ressaltar que nem todas as operações de edição são inversíveis. Por exemplo, a operação de edição γ_i definida a seguir não é inversível: dado i tal que $1 \leq i$, definimos γ_i como a função parcial $\gamma_i : \Sigma^* \to \Sigma^*$ tal que $\gamma_i(s) = s[1\,..\,i-1]s[i+1\,..\,n]$, para $i \leq n$.

Observação 2.22 *Se uma operação de edição γ é inversível então a operação de edição γ^{-1} também é inversível e $(\gamma^{-1})^{-1} = \gamma$.*

Observação 2.23 *Sejam as operações de edição $\gamma_1, \gamma_2, \ldots, \gamma_k$ tal que, para algum i, $\gamma_i^{-1} = \gamma_{i+1}$, onde $1 \leq i < k$, e as seqüências de operações de edição $\Psi = (\gamma_1, \ldots, \gamma_i, \gamma_{i+1}, \ldots, \gamma_k)$ e $\Psi' = (\gamma_1, \ldots, \gamma_{i-1}, \gamma_{i+2}, \ldots, \gamma_k)$. Temos que $\phi_\Psi(s) = \phi_{\Psi'}(s)$.*

Proposição 2.24 *As operações de edição elementares são inversíveis e suas inversas são também operações de edição elementares e são listadas a seguir:*

$$\begin{aligned}
&\iota_{i,a}: && (\iota_{i,a})^{-1} = \rho_{i+1,a}\\
&\rho_{i,a}: && (\rho_{i,a})^{-1} = \iota_{i-1,a}\\
&\sigma_{i,a,b}: && (\sigma_{i,a,b})^{-1} = \sigma_{i,b,a}\\
&\pi_{i,i'}: && (\pi_{i,i'})^{-1} = \pi_{i,i'}\\
&\delta_{i,i_1,i_2}: && (\delta_{i,i_1,i_2})^{-1} = \varepsilon_{i_2+1,i_2+1+i_1-i,i}\\
&\varepsilon_{i,i_1,i_2}: && (\varepsilon_{i,i_1,i_2})^{-1} = \delta_{i_2,i_2+i_1-i,i-1}.
\end{aligned}$$

Prova. Observe que:

1. $\rho_{i+1,a}(\iota_{i,a}(s)) = \rho_{i+1,a}(s[1\,..\,i]\ a\ s[i+1\,..\,n]) = s[1\,..\,i]s[i+1\,..\,n] = s$ e
2. $\iota_{i,a}(\rho_{i+1,a}(s)) = \iota_{i,a}(s[1\,..\,i]s[i+2\,..\,n]) = s[1\,..\,i]\ a\ s[i+2\,..\,n] = s.$

Portanto a inversa de $\iota_{i,a}$ é $\rho_{i+1,a}$. De modo similar pode-se provar que as inversas das outras operações de edição elementares são as listadas no enunciado da proposição. ∎

Seja $\Psi = (\gamma_1, \gamma_2, \ldots, \gamma_k)$ tal que γ_i é uma operação de edição inversível, para todo i tal que $1 \leq i \leq k$. Temos que a operação de edição composta de Ψ é inversível e sua inversa é dada por

$$(\phi_\Psi)^{-1} = (\gamma_1)^{-1} \circ (\gamma_2)^{-1} \circ \ldots \circ (\gamma_{k-1})^{-1} \circ (\gamma_k)^{-1}.$$

Neste caso, dizemos que $\Psi^{-1} = ((\gamma_k)^{-1}, (\gamma_{k-1})^{-1}, \ldots, (\gamma_1)^{-1})$ é a seqüência inversa de Ψ e $(\phi_\Psi)^{-1} = \phi_{\Psi^{-1}}$.

2.6 Grafo de edição

Sejam s e t duas seqüências de comprimentos n e m respectivamente.

Seja $e = (u, v)$ uma aresta que liga o vértice u ao vértice v de um grafo qualquer. Dizemos que $\text{start}(e) = u$ e $\text{end}(e) = v$.

Definição 2.25 (Grafo de edição de *s* e *t*) *O grafo de edição de s e t é o grafo orientado com pesos nas arestas $G = (V, E, \omega)$, onde:*

1) $V = \{(i, j) | 0 \leq i \leq n, 0 \leq j \leq m\}$.

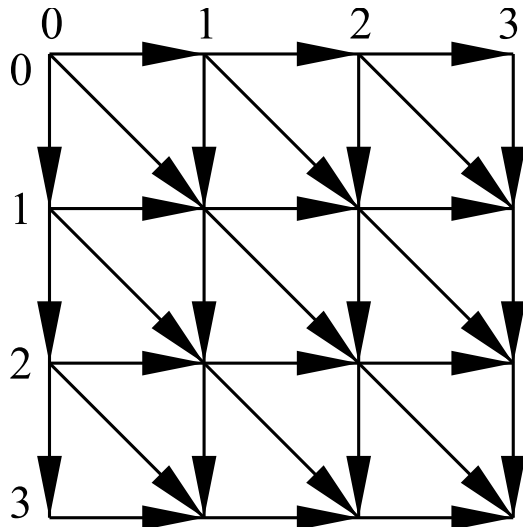

Figura 2.4: Exemplo de grafo de edição

2) $E = E_H \cup E_D \cup E_V$*, tal que:*

- $E_H = \{((i, j-1), (i, j)) | 0 \leq i \leq n, 0 < j \leq m\}$ *é o conjunto das arestas horizontais de* G*;*
- $E_D = \{((i-1, j-1), (i, j)) | 0 < i \leq n, 0 < j \leq m\}$ *é o conjunto das arestas diagonais de* G*;*
- $E_V = \{((i-1, j), (i, j)) | 0 < i \leq n, 0 \leq j \leq m\}$ *é o conjunto das arestas verticais de* G*.*

3) A função $\omega : E \longrightarrow \mathbb{R} \cup \{-\infty\}$ *associa a cada aresta* $e \in E$ *o seu peso* $\omega(e)$*.*

Seja G um grafo de edição de s e t.

As arestas $\epsilon_H^{(i,j)} = ((i, j-1), (i, j))$, $\epsilon_D^{(i,j)} = ((i-1, j-1), (i, j))$ e $\epsilon_V^{(i,j)} = ((i-1, j), (i, j))$ de G são as arestas *horizontal*, *diagonal* e *vertical*, respectivamente, *de* G *que chegam em* (i, j).

A figura 2.4 ilustra a forma de um grafo de edição com $n = 3$ e $m = 3$. Repare que neste exemplo, para não sobrecarregar demais a figura, não mostramos os valores de $\omega(e)$ para qualquer aresta e do grafo.

A cada aresta ϵ de um grafo de edição G de s e t vamos associar uma operação de edição pontual da seguinte forma:

- $\epsilon = \epsilon_D^{(i,j)}$ está associada a operação de edição $\sigma_{j,t[j],s[i]}$,
- $\epsilon = \epsilon_H^{(i,j)}$ está associada a operação de edição $\iota_{j-1,t[j]}$,
- $\epsilon = \epsilon_V^{(i,j)}$ está associada a operação de edição $\rho_{j+1,s[i]}$.

Com esta associação, podemos dizer que um caminho P de $(0,0)$ a (n,m) em G está associado a uma seqüência Ψ de operações de edição pontuais, tal que $\phi_\Psi(s) = t$. Ou seja, qualquer caminho de $(0,0)$ a (n,m) em G corresponde a uma seqüência de operações de edição pontuais que transforma s em t.

Na verdade, podemos afirmar que qualquer caminho de $(i_1.j_1)$ a (i_2,j_2) em G corresponde a uma seqüência de operações de edição que transforma $s[i_1+1\,..\,i_2]$ em $t[j_1+1\,..\,j_2]$, alterando-se apenas os índices da operação de edição associada a cada aresta ϵ da seguinte forma:

- $\epsilon = \epsilon_D^{(i,j)}$ está associada a operação de edição $\sigma_{j-j_1,t[j],s[i]}$,
- $\epsilon = \epsilon_H^{(i,j)}$ está associada a operação de edição $\iota_{j-1-j_1,t[j]}$,
- $\epsilon = \epsilon_V^{(i,j)}$ está associada a operação de edição $\rho_{j+1-j_1,s[i]}$.

Definição 2.26 (Grafo de edição estendido) *O grafo de edição estendido de s e t é o grafo orientado com pesos nas arestas $G = (V,E,\omega)$, onde:*

1) $V = \{(i,j)|0 \leq i \leq n, 0 \leq j \leq m\}$.

2) $E = E_H \cup E_D \cup E_V \cup E_X$, *tal que:*

- $E_H = \{((i,j-1),(i,j))|0 \leq i \leq n, 0 < j \leq m\}$ *é o conjunto das arestas horizontais de G;*
- $E_D = \{((i-1,j-1),(i,j))|0 < i \leq n, 0 < j \leq m\}$ *é o conjunto das arestas diagonais de G;*
- $E_V = \{((i-1,j),(i,j))|0 < i \leq n, 0 \leq j \leq m\}$ *é o conjunto das arestas verticais de G;*
- $E_X = \bigcup_{i=0}^{n} \bigcup_{j=0}^{m} E_X^{i,j}$ *é o conjunto das arestas estendidas de G, tal que:*

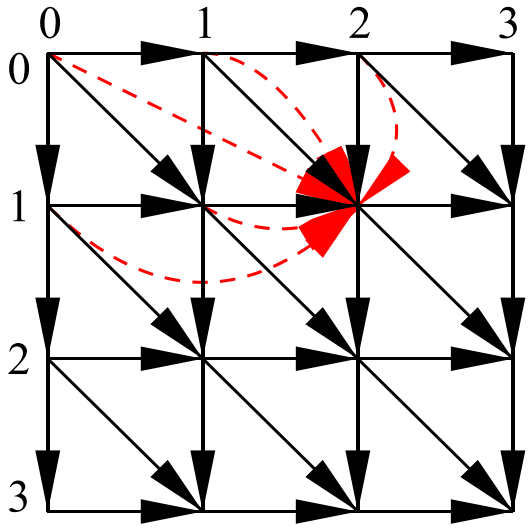

Figura 2.5: Exemplo de grafo de edição estendido

- $E_X^{i,j} = \{((i',j'),(i,j)) \mid 0 \leq i' \leq i \leq n, 0 \leq j' \leq j \leq m \text{ e } (i',j') \neq (i,j)\}$ *é o conjunto das arestas estendidas que chegam no vértice* (i,j) *de* G *e*
- $E_X \cap \{E_H \cup E_D \cup E_V\} = \emptyset$.

3) A função $\omega : E \longrightarrow \mathbb{R} \cup \{-\infty\}$ *associa a cada aresta* $\epsilon \in E$ *o seu peso* $\omega(\epsilon)$.

Seja G um grafo de edição estendido de s e t.

As arestas $\epsilon_H^{(i,j)} = ((i,j-1),(i,j))$, $\epsilon_D^{(i,j)} = ((i-1,j-1),(i,j))$ e $\epsilon_V^{(i,j)} = ((i-1,j),(i,j))$ de G, tais que $\epsilon_H^{(i,j)} \in E_H$, $\epsilon_D^{(i,j)} \in E_D$ e $\epsilon_V^{(i,j)} \in E_V$, são as arestas *horizontal*, *diagonal* e *vertical*, respectivamente, *de* G *que chegam em* (i,j) e serão chamadas de arestas simples.

As arestas $\epsilon_{(i',j')}^{(i,j)} = ((i',j'),(i,j))$ de G tais que $\epsilon_{(i',j')}^{(i,j)} \in E_X^{i,j}$ são as *arestas estendidas de* G *que chegam em* (i,j).

A figura 2.5 ilustra a forma de um grafo de edição estendido com $n = 3$ e $m = 3$, onde somente as arestas estendidas que chegam em $(1,2)$ são mostradas. Repare que neste exemplo, novamente para não sobrecarregar demais a figura, não mostramos as outras arestas estendidas além dos valores de $\omega(e)$ para qualquer aresta e do grafo.

Repare que podem existir até duas arestas que saem de (i',j') e chegam

em (i, j). Por exemplo, na figura 2.5 pode-se observar que $\epsilon_D^{(1,2)}$ e $\epsilon_{(0,1)}^{(1,2)}$ são duas arestas que saem de $(0, 1)$ e chegam em $(1, 2)$.

Seja $\widehat{G}$ um grafo de edição estendido de s e t. Seja G o grafo obtido de $\widehat{G}$ removendo-se as arestas estendidas. Temos que G é um grafo de edição de s e t.

Seja G um grafo de edição de s e t ou um grafo de edição estendido de s e t.

Sejam $u = (i, j)$ e $v = (i', j')$ dois vértices de G. Iremos considerar que um caminho P de u a v é ótimo se ele tiver peso máximo entre todos os caminhos de u a v. Dizemos que $\omega(u, v)$ é o peso de um caminho ótimo de u a v. Se não houver um caminho de u a v então $\omega(u, v) = -\infty$.

Subentendendo-se uma disposição matricial natural dos vértices de G, dizemos que G tem $n + 1$ linhas e $m + 1$ colunas. Mais ainda, dizemos que o vértice $v = (i, j)$ está na linha $\text{row}(v) = i$ e na coluna $\text{col}(v) = j$.

Todos os algoritmos que apresentaremos neste texto, que têm como objetivo a obtenção de um alinhamento ótimo, estão baseados em grafos de edição e grafos de edição estendidos. Em particular, os caminhos em um grafo de edição ou em um grafo de edição estendido serão mapeados para um alinhamento.

Capítulo 3

Alinhamento

3.1 Alinhamento usual

Quando duas seqüências tem alto grau de semelhança é esperado que elas se diferenciem apenas em pequenos trechos. Normalmente as diferenças nestes trechos são conseqüências da ocorrência de alguns eventos biológicos. Em geral, para a visualização destes eventos ocorridos, é feito um procedimento de alinhamento entre as duas seqüências. Procedimentos de alinhamento típicos tentam identificar que partes não mudam e, nas partes que mudam, quais os eventos biológicos que ocasionaram as mudanças.

O alinhamento de seqüências biológicas é uma das principais técnicas utilizadas por biólogos para comparar seqüências.

Neste capítulo vamos considerar que Σ é um alfabeto com um número constante de símbolos. Vamos considerar também que $- \notin \Sigma$ e $\Sigma' = \Sigma \cup \{-\}$. Chamaremos o símbolo $-$ de espaço (ou *gap*).

Definição 3.1 (Alinhamento de s e t) *Sejam $s = \Sigma^*$ e $t = \Sigma^*$ duas seqüências de comprimentos n e m, respectivamente. Um alinhamento de s e t é uma matriz $A_{2\times r}$, tal que:*

- $r \geq m$, $r \geq n$, $r \leq m + n$,
- *para todo j tal que $0 \leq j \leq r-1$, se $A[0,j] = A[1,j]$ então $A[0,j] \neq -$ e*
- *existem duas subseqüências $S = (i_1, i_2, \ldots, i_n)$ e $T = (j_1, j_2, \ldots, j_m)$ dos índices das colunas de A, tais que:*

- $s = A[0, i_1]A[0, i_2] \ldots A[0, i_n]$,
- $t = A[1, j_1]A[1, j_2] \ldots A[1, j_m]$,
- $A[0, i] = -$ *para toda coluna* i *de* A*, tal que* $i \notin S$*, e*
- $A[1, j] = -$ *para todo coluna* j *de* A*, tal que* $j \notin T$.

Em outras palavras, um alinhamento de s e t é uma matriz A com duas linhas de comprimento r tal que, a primeira linha de A é a seqüência s alterada com possíveis espaços inseridos entre os seus símbolos ou nas suas extremidades, e a segunda linha de A é a seqüência t alterada com possíveis espaços inseridos entre os seus símbolos ou nas suas extremidades. Além disso, para que A seja um alinhamento, assumimos que toda coluna de A tem pelo menos um símbolo em Σ. Dizemos que o comprimento do alinhamento A é r.

O exemplo 3.2 mostra um alinhamento das seqüências
$s = AGCGTATCCAGT$ e $t = AGTATCACGGAT$.

Exemplo 3.2 (Alinhamento de duas seqüências)

$$A = \begin{bmatrix} A & G & C & G & T & - & A & T & C & C & A & G & - & T \\ A & G & T & A & T & C & A & - & C & G & - & G & A & T \end{bmatrix}$$

Neste capítulo consideraremos que s e t são duas seqüências em Σ^* e que A é um alinhamento de s e t de comprimentos n e m, respectivamente.

De acordo com os espaços contidos em uma coluna k de A, esta coluna k pode ser classificada nos seguintes tipos:

***Substituição*:** se existem i e j tais que $s[i]$ e $t[j]$ estão na coluna k então esta é uma coluna de substituição. Neste caso, dizemos que $s[i]$ e $t[j]$ estão alinhados em A, ou mais especificamente, $s[i]$ e $t[j]$ estão alinhados na coluna k de A. Se $s[i] = t[j]$ então dizemos que há um *casamento* (ou *match*) na coluna k de A. Se $s[i] \neq t[j]$ então dizemos que há um *erro* (ou *mismatch*) na coluna k de A.

***Inserção*:** se $A[0, k] = -$ e existe j tal que $t[j]$ está na coluna k então esta é uma coluna de inserção. Neste caso, dizemos que $t[j]$ está alinhado com um espaço em A, ou mais especificamente, $t[j]$ está alinhado com um espaço na coluna k de A. Dizemos que há uma *lacuna* (ou *gap*) na coluna k de A.

***Remoção*:** se $A[1,k] = -$ e existe i tal que $s[i]$ está na coluna k então esta é uma coluna de remoção. Neste caso, dizemos que $s[i]$ está alinhado com um espaço em A, ou mais especificamente, $s[i]$ está alinhado com um espaço na coluna k de A. Dizemos que há uma *lacuna* (ou *gap*) na coluna k de A.

Diremos que Λ é o conjunto de todos os possíveis alinhamentos de quaisquer duas seqüências. Dado s, diremos que Λ_s é o conjunto de todos os possíveis alinhamentos de s e qualquer seqüência ou de qualquer seqüência e s. Dados s e t, diremos que $\Lambda_{s,t}$ é o conjunto de todos os possíveis alinhamentos de s e t.

Diremos que $A[i, j\,..\,j']$ é a palavra $A[i,j]A[i,j+1]\ldots A[i,j']$. Se $j' < j$ então $A[i, j\,..\,j']$ é a palavra vazia. Por exemplo, no Exemplo 3.2 a palavra $A[0, 3\,..\,7]$ é igual a $GT{-}AT$.

Seja a função parcial $\mathrm{col} : \Sigma^* \times \mathbb{N} \times \Lambda \to \mathbb{N}$ definida por $\mathrm{col}(s,i,A) = k$ tal que i é um índice de s, $A \in \Lambda_s$ e k é o índice da coluna de A tal que $A[l,k] = s[i]$ e existem $k - i + 1$ espaços na palavra $A[l, 0\,..\,k]$, onde $l = 0$ se A é um alinhamento de s e t ou $l = 1$ se A é um alinhamento de t e s. Podemos dizer que se $\mathrm{col}(s,i,A) = k$ então $s[i]$ está alinhado na coluna k de A. Como o número de espaços numa palavra é maior ou igual a zero, temos que $k - i + 1 \geq 0$, ou seja, $k \geq i - 1$. Por exemplo, no Exemplo 3.2, $\mathrm{col}(s, 5, A) = 4$.

Repare que, como $s[i]$ representa um símbolo, podemos ter outra posição $i' \neq i$ tal que $s[i'] = s[i]$ e portanto $\mathrm{col}(s,i',A) = k' \neq k$. Logo o símbolo $s[i] = s[i']$ está alinhado na coluna k' de A também. Por exemplo, no Exemplo 3.2, $\mathrm{col}(t, 10, A) = 11$ e $t[10] = G$ está alinhado nas coluna 1, 9 e 11 de A.

Seja a função parcial $\mathrm{pref} : \Sigma^* \times \mathbb{N} \times \Lambda \to \Sigma^*$ definida por $\mathrm{pref}(s,k,A) = s[1\,..\,i]$ tal que k é um índice de A, $A \in \Lambda_s$ e $s[1\,..\,i]$ é o prefixo de s tal que $\mathrm{col}(s,i',A) \leq k$ para todo i' tal que $i' \leq i$ e $\mathrm{col}(s,i'',A) > k$ para todo i'' tal que $i'' > i$. Ou seja, $\mathrm{pref}(s,k,A)$ é o mais longo prefixo de s cujos elementos estão alinhados até a coluna k de A. Por exemplo, no Exemplo 3.2 $\mathrm{pref}(t, 11, A) = t[1\,..\,10] = AGTATCACGG$ e $\mathrm{pref}(s, 5, A) = s[1\,..\,5] = AGCGT$.

Observação 3.3 *Se col$(s,i,A) = k$ então temos que pref$(s,k,A) = s[1\,..\,i]$ e pref$(s,k-1,A) = s[1\,..\,i-1]$. Repare que se pref$(s,k,A) = s[1\,..\,i]$ então não necessariamente col$(s,i,A) = k$, porém se pref$(s,k,A) = s[1\,..\,i]$ e pref$(s,k-$*

$1, A) = s[1 \ldots i-1]$ *então* $col(s, i, A) = k$. *Se* $col(s, i, A) = k$ *e* $s[i]$ *é alinhado com* $-$ *na coluna* k *de* A *então* $pref(t, k, A) = pref(t, k-1, A)$.

Existem diferentes sistemas de pontuação para alinhamentos, ou maneiras de atribuir pontuação a um alinhamento. Um sistema de pontuação muito utilizado é aquele que, dada uma função $\varphi : \{\Sigma'\} \times \{\Sigma'\} \to \mathbb{R}$, determina que a pontuação de cada coluna k de A é igual a $\varphi(A[0, k], A[1, k])$ e a pontuação do alinhamento A é igual à somatória da pontuação de cada coluna de A. Chamaremos este sistema de pontuação de sistema de pontuação com custo de *gap* linear. Existe um outro sistema de pontuação também muito utilizado que determina que a pontuação de uma coluna é igual a $\varphi(A[0, k], A[1, k]) + \rho$, onde ρ é diferente de zero se $A[0, k] = -$ e $A[0, k-1] \neq -$, ou se $A[1, k] = -$ e $A[1, k-1] \neq -$. Normalmente ρ é chamado de penalidade para abertura de lacuna, ou *gap open penalty*. Chamaremos este sistema de pontuação de sistema de pontuação com *gap* afim.

A questão de como definir similaridade ainda é controversa. Não existe uma única, precisa ou universalmente aceita noção de similaridade, assim como um sistema que define ou mede o grau, ou valor, de similaridade. Por exemplo, duas proteínas podem ser similares (ou seja, com alto grau de similaridade) com relação à sua estrutura, ou com relação à sua função ou com relação à sua seqüência. Em geral, é a seqüência que determina a estrutura e a estrutura determina a função. Assim, quando analisamos e atribuímos um grau de similaridade às seqüências, esperamos que este grau reflita, da mesma forma que nas seqüências, a similaridade nas estruturas e nas funções destas seqüências.

Em geral, um alinhamento de seqüências é utilizado para visualizar os fragmentos iguais e os fragmentos diferentes destas seqüências. No entanto, algumas vezes o alinhamento de seqüências é utilizado para medir apenas o grau de similaridade das seqüências, sem interessar onde estão e quais são as diferenças e as igualdades destas seqüências, ou seja, existem situações onde não há interesse em visualizar o alinhamento, mas sim em saber o quanto as seqüências são similares.

Assumiremos que o sistema de pontuação a ser utilizado em um alinhamento considera que a pontuação do alinhamento é diretamente proporcional às similaridades mostradas por este alinhamento, ou seja, um alinhamento procura medir o grau de similaridade das seqüências. Portanto, assumiremos que um *alinhamento ótimo de* s *e* t é aquele que possui pontuação máxima dentre todos os possíveis alinhamentos de s e t. Desta forma, quanto maior

a pontuação de um alinhamento ótimo de s e t, maior é a similaridade destas seqüências. Vale a pena ressaltar que dois alinhamentos diferentes podem ter a mesma pontuação e assim podem existir mais de um alinhamento ótimo para um mesmo par de seqüências e um mesmo sistema de pontuação.

Muitas vezes, quando as seqüências têm grau de similaridade, ou pontuação de um alinhamento ótimo, acima de um certo limite mínimo estipulado, vamos dizer que as seqüências são similares.

3.2 Alinhamentos e operações de edição

Seja γ^* o conjunto de todas as operações de edição. Para cada coluna k de um alinhamento qualquer A de s e t vamos associar uma operação de edição pontual através da função parcial $\text{op} : \Lambda \times \mathbb{N} \to \gamma^*$ definida por:

- $\text{op}(A, k) = \sigma_{j,t[j],s[i]}$ se a coluna k de A é do tipo substituição, onde i e j são tais que $\text{col}(s, i, A) = k$ e $\text{col}(t, j, A) = k$;

- $\text{op}(A, k) = \iota_{j-1,t[j]}$ se a coluna k de A é do tipo inserção, onde j é tal que $\text{col}(t, j, A) = k$;

- $\text{op}(A, k) = \rho_{j+1,s[i]}$ se a coluna k de A é do tipo remoção, onde i e j são tais que $\text{col}(s, i, A) = k$, $j \geq 0$ e $\text{pref}(t, k, A) = t[1 \,..\, j]$.

Como um alinhamento é uma seqüência de colunas e cada coluna está associada a uma operação de edição pontual, podemos associar um alinhamento a uma seqüência de operações de edição pontuais através da função parcial op.

Proposição 3.4 *Seja A um alinhamento de s e t de comprimento r. Temos que, para toda coluna l de A, $\phi_{\Psi_l}(\text{pref}(s, l, A)) = \text{pref}(t, l, A)$, onde $\Psi_l = (\gamma_0, \gamma_1, \ldots, \gamma_l)$ e $\gamma_k = \text{op}(A, k)$ para toda coluna k de A tal que $0 \leq k \leq l$.*

Prova. A prova é por indução em l.

1. Base da indução: $\phi_{\Psi_l}(\text{pref}(s, l, A)) = \text{pref}(t, l, A)$, para $l = 0$.

   De acordo com o tipo da coluna $l = 0$ temos que:

(a) Se l é uma coluna do tipo substituição, então $A[0,0] = s[1]$, $A[1,0] = t[1]$ e $\gamma_l = \sigma_{1,t[1],s[1]}$. Como, $\text{pref}(s,0,A) = s[1]$ e $\text{pref}(t,0,A) = t[1]$ então

$$\begin{aligned}\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(s[1]) \\ &= \sigma_{1,t[1],s[1]}(s[1]) \\ &= t[1] \\ &= \text{pref}(t,l,A).\end{aligned}$$

(b) Se l é uma coluna do tipo inserção, então $A[1,0] = t[1]$ e $A[0,0] = -$. Portanto, $\text{col}(t,1,A) = 0$, $\gamma_l = \iota_{0,t[1]}$, $\text{pref}(t,l,A) = t[1]$ e $\text{pref}(s,l,A) = s[1\,..\,0]$.

$$\begin{aligned}\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(s[1\,..\,0]) \\ &= \iota_{0,t[1]}(s[1\,..\,0]) \\ &= t[1] \\ &= \text{pref}(t,l,A).\end{aligned}$$

(c) Se l é uma coluna do tipo remoção, então $A[1,0] = -$ e $A[0,0] = s[1]$. Portanto, $\text{col}(s,1,A) = 0$, $\text{pref}(t,l,A) = t[1\,..\,0]$, $\gamma_l = \rho_{1,s[1]}$ e $\text{pref}(s,l,A) = s[1]$.

$$\begin{aligned}\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(s[i]) \\ &= \rho_{1,s[1]}(s[1]) \\ &= t[1\,..\,0] \\ &= \text{pref}(t,l,A).\end{aligned}$$

Portanto, para $l = 0$ temos que $\phi_{\Psi_l}(\text{pref}(s,l,A)) = \text{pref}(t,l,A)$.

2. Passo da indução: Dado que $\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A)) = \text{pref}(t,l-1,A)$ então $\phi_{\Psi_l}(\text{pref}(s,l,A)) = \text{pref}(t,l,A)$, para qualquer l tal que $0 \leq l \leq r-1$.

   De acordo com o tipo da coluna l temos que:

   (a) Se l é uma coluna do tipo substituição, então $\gamma_l = \sigma_{j,t[j],s[i]}$, onde i e j são índices tais que $\text{col}(s,i,A) = l$ e $\text{col}(t,j,A) = l$. Utilizando

as observações 2.21 e 3.3 temos que:

$$\begin{aligned}
\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(\text{pref}(s,l-1,A)s[i]) \\
&= \sigma_{j,t[j],s[i]}(\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A)s[i])) \\
&= \sigma_{j,t[j],s[i]}(\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A))s[i]) \\
&= \sigma_{j,t[j],s[i]}(\text{pref}(t,l-1,A)s[i]) \\
&= \text{pref}(t,l-1,A)t[j] \\
&= \text{pref}(t,l,A).
\end{aligned}$$

(b) Se l é uma coluna do tipo inserção, então $\gamma_l = \iota_{j-1,t[j]}$, onde j é tal que $\text{col}(t,j,A) = l$. Utilizando as observações 2.21 e 3.3 temos que:

$$\begin{aligned}
\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(\text{pref}(s,l-1,A)) \\
&= \iota_{j-1,t[j]}(\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A))) \\
&= \iota_{j-1,t[j]}(\text{pref}(t,l-1,A)) \\
&= \iota_{j-1,t[j]}(t[1\,..\,j-1]) \\
&= t[1\,..\,j] \\
&= \text{pref}(t,l,A).
\end{aligned}$$

(c) Se l é uma coluna do tipo remoção, então $\gamma_l = \rho_{j+1,s[i]}$, onde j é tal que $\text{pref}(t,l,A) = t[1\,..\,j]$ e $j \geq 0$, e i é tal que $\text{col}(s,i,A) = l$. Utilizando as observações 2.21 e 3.3 temos que:

$$\begin{aligned}
\phi_{\Psi_l}(\text{pref}(s,l,A)) &= \phi_{\Psi_l}(\text{pref}(s,l-1,A)s[i]) \\
&= \rho_{j+1,s[i]}(\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A)s[i])) \\
&= \rho_{j+1,s[i]}(\phi_{\Psi_{l-1}}(\text{pref}(s,l-1,A))s[i]) \\
&= \rho_{j+1,s[i]}(\text{pref}(t,l-1,A)s[i]) \\
&= \rho_{j+1,s[i]}(t[1\,..\,j]s[i]) \\
&= t[1\,..\,j] \\
&= \text{pref}(t,l,A). \quad \blacksquare
\end{aligned}$$

Corolário 3.5 *Seja A um alinhamento de s e t de comprimento r. Seja $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_{r-1})$ onde, para cada coluna k de A, γ_k é a operação de edição associada a coluna k através da função parcial op. Temos que $\phi_\Psi(s) = t$.*

Repare que, dado uma seqüência de operações de edição pontuais $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_i, \ldots, \gamma_j, \ldots, \gamma_{r-1})$ tal que $\phi_\Psi(s) = t$, como no Corolário 3.5, podemos obter $\Psi' = (\gamma_0, \gamma_1, \ldots, \gamma'_j, \gamma'_{i+1}, \ldots, \gamma'_{j-1}, \gamma'_i, \ldots, \gamma_{r-1})$, para $r \geq 2$ e quaisquer i e j tais que, $0 \leq i < j \leq r-1$, de tal forma que $\phi_\Psi(s) = \phi_{\Psi'}(s) = t$ e as operações de edição $\gamma'_i, \gamma'_{i+1}, \ldots, \gamma'_{j-1}$ e γ'_j pertencem, respectivamente, à mesma classe das operações de edição $\gamma_i, \gamma_{i+1}, \ldots, \gamma_{j-1}$ e γ_j, mas são diferentes destas em seus índices. Ou seja, as operações de edição da seqüência Ψ podem ter a ordem alterada juntamente com seus índices para obter uma outra seqüência Ψ' tal que $\phi_\Psi(s) = \phi_{\Psi'}(s) = t$.

Em biologia é comum ao fazer uma análise evolutiva de duas seqüências, supor a existência de uma seqüência ancestral destas duas seqüênciase que após um processo de replicação gera duas seqüências iguais, as quais sofrem mutações com o decorrer do tempo e geram as duas seqüências que estão sendo analisadas. Portanto, é interessante, e muitas vezes desejado pelos biólogos, fazer uma análise evolutiva de duas seqüências para tentar descobrir esta seqüência ancestral e conhecer as mutações que ocorreram com o decorrer do tempo.

Apesar de cada coluna de um alinhamento estar associada a uma operação de edição e cada operação de edição estar relacionada a uma mutação, não podemos dizer que um alinhamento por si só mostra a seqüência de mutações que ocorreram em uma seqüência ancestral que geraram as duas seqüências sendo alinhadas, pois como já vimos, dado um alinhamento de duas seqüências, a ordem das operações de edição associadas a este alinhamento pode facilmente ser alterada. Portanto descobrir esta ordem dos eventos requer mais informações que as utilizadas num alinhamento. Além disso, um alinhamento não indica a seqüência ancestral, e portanto ele não mostra precisamente qual foi a mutação que ocorreu, mesmo restringindo as possibilidades de mutações às mutações relacionadas às operações de edição pontuais. Assim, um alinhamento de duas seqüências não tem como objetivo, mostrar que os eventos biológicos ocorridos são precisamente os eventos mapeados pelas operações de edição associadas a cada coluna deste alinhamento. Por exemplo, uma coluna do tipo inserção, de um alinhamento de s e t, indica que pode ter ocorrido um evento de inserção em s ou um evento de remoção em t, pois o alinhamento não mostra qual é a seqüência ancestral de s e t. As operações de edição associadas as colunas indicam apenas que podemos alterar s com estas operações de edição para obter t. Quando associamos as operações de edição às colunas de um alinhamento de s e t, estabelecemos arbitrariamente que as operações de edição são sempre sobre s, porém sem a

intenção de indicar que s é a seqüência ancestral. Além disto, estabelecemos arbitrariamente que a ordem das operações de edição são da esquerda para a direita nas colunas de um alinhamento.

Do ponto de vista evolutivo, um alinhamento de duas seqüências por si só não garante a ordem em que ocorreram as mutações, nem tampouco mostra qual é a seqüência ancestral. Porém, é possível utilizar um alinhamento de duas seqüências para se inferir hipóteses para uma análise evolutiva.

3.3 Alinhamentos e grafos de edição

Proposição 3.6 *Seja G um grafo de edição de $s = s[1\,..\,n]$ e $t = t[1\,..\,m]$. Seja E o conjunto de arestas de G. Sejam $\Lambda_{s,t}$ o conjunto de todos os possíveis alinhamentos de s e t e $\Pi_{(0,0),(n,m)}$ o conjunto de todos os possíveis caminhos de $(0,0)$ a (n,m) em G. Seja a função parcial edge $: \Lambda_{s,t} \times \mathbb{N} \to E$ definida por:*

- *edge$(A,\ k) = \epsilon_D^{(i,j)}$ se k é uma coluna de A e é do tipo substituição,*
- *edge$(A,\ k) = \epsilon_H^{(i,j)}$ se k é uma coluna de A e é do tipo inserção,*
- *edge$(A,\ k) = \epsilon_V^{(i,j)}$ se k é uma coluna de A e é do tipo remoção,*

onde i e j são tais que $0 \leq i \leq n$, $0 \leq j \leq m$, pref$(s, k, A) = s[1\,..\,i]$ e pref$(t, k, A) = t[1\,..\,j]$. Temos que:

1. *A seqüência $P = (\text{edge}(A,\ 0), \text{edge}(A,\ 1), \ldots, \text{edge}(A,\ r-1))$ é um caminho de $(0,0)$ a (n,m) em G, onde A é um alinhamento de s e t de comprimento r.*
2. *A função path $: \Lambda_{s,t} \to \Pi_{(0,0),(n,m)}$ definida por, path$(A) = (\text{edge}(A,\ 0), \text{edge}(A,\ 1), \ldots, \text{edge}(A,\ r-1)) = P$, onde A é um alinhamento de comprimento r, é bijetora.*

Prova. 1. P é um caminho de $(0,0)$ a (n,m) em G.

Se $n = 0$ e $m = 0$ então P é a seqüência vazia e como G só tem o vértice $(0,0)$ então P é um caminho de $(0,0)$ a (n,m) em G.

Vamos assumir que $n \neq 0$ ou $m \neq 0$.

Para que P seja um caminho de $(0,0)$ a (n,m) em G, é necessário provarmos que start(edge$(A,\ 0)) = (0,0)$, end(edge$(A,\ r-1)) = (n,m)$

e que para todo k, $1 \leq k \leq r-1$, end(edge(A, $k-1$)) = start(edge(A, k)).

(a) start(edge(A, 0)) = $(0,0)$

Se a coluna 0 de A é do tipo substituição então pref$(s, 0, A) = s[1]$ e pref$(t, 0, A) = t[1]$ e portanto edge(A, 0) = $\epsilon_D^{(1,1)}$ e start(edge(A, 0)) = $(0,0)$.

Se a coluna 0 de A é do tipo inserção então pref$(s, 0, A) = s[1\,..\,0]$ e pref$(t, 0, A) = t[1]$ e portanto edge(A, 0) = $\epsilon_H^{(0,1)}$ e start(edge(A, 0)) = $(0,0)$.

Se a coluna 0 de A é do tipo remoção então pref$(s, 0, A) = s[1]$ e pref$(t, 0, A) = t[1\,..\,0]$ e portanto edge$(A, 0) = \epsilon_V^{(1,0)}$ e start(edge(A, 0)) = $(0,0)$.

(b) end(edge(A, $r-1$)) = (n, m)

Para a coluna $r-1$ de A temos que pref$(s, r-1, A) = s[1\,..\,n]$ e pref$(t, r-1, A) = t[1\,..\,m]$, pois A é um alinhamento de s e t de comprimento r. A aresta edge(A, $r-1$) pode ser $\epsilon_D^{(n,m)}$, $\epsilon_H^{(n,m)}$ ou $\epsilon_V^{(n,m)}$, dependendo do tipo da coluna $r-1$ de A. Portanto, para qualquer que seja o tipo da coluna $r-1$ de A, end(edge(A, $r-1$)) = (n, m).

(c) end(edge(A, $k-1$)) = start(edge(A, k)), para todo k tal que $1 \leq k \leq r-1$.

Seja end(edge(A, $k-1$)) = (i, j). Independente do tipo da coluna $k-1$, como end(edge(A, $k-1$)) = (i, j) então, pela definição de edge, pref$(s, k-1, A) = s[1\,..\,i]$ e pref$(t, k-1, A) = t[1\,..\,j]$.

Se a coluna k de A é do tipo substituição então pref$(s, k, A) = s[1\,..\,i+1]$ e pref$(t, k, A) = t[1\,..\,j+1]$ e portanto edge(A, k) = $\epsilon_D^{(i+1,j+1)}$ e start(edge(A, k)) = (i, j) = end(edge(A, $k-1$)).

Se a coluna k de A é do tipo inserção então pref$(s, k, A) = s[1\,..\,i]$ e pref$(t, k, A) = t[1\,..\,j+1]$ e portanto edge(A, k) = $\epsilon_H^{(i,j+1)}$ e start(edge(A, k)) = (i, j) = end(edge(A, $k-1$)).

Se a coluna k de A é do tipo remoção então pref$(s, k, A) = s[1\,..\,i+1]$ e pref$(t, k, A) = t[1\,..\,j]$ e portanto edge(A, k) = $\epsilon_V^{(i+1,j)}$ e start(edge(A, k)) = (i, j) = end(edge(A, $k-1$)).

Portanto P é um passeio de $(0,0)$ a (n,m) em G. Como um grafo de edição é um grafo acíclico, não há vértices repetidos e a primeira afirmação da proposição está demonstrada.

2. path é uma função bijetora.

   Para que path seja bijetora é necessário que path seja uma função definida para todos os elementos do seu domínio e que seu contra-domínio seja igual ao seu domínio, ou seja, para todo alinhamento A existe um único P tal que $\text{path}(A) = P$ assim como para todo caminho P existe um único A tal que $\text{path}(A) = P$.

   Se $n = 0$ e $m = 0$ então A é o alinhamento sem colunas e $\text{path}(A) = P$ é um caminho sem arestas. Temos também que $\Lambda_{s,t} = A$ e $\Pi_{(0,0),(n,m)} = P$. Portanto path é uma função bijetora.

   Vamos assumir que $n \neq 0$ ou $m \neq 0$.

   (a) Para todo alinhamento A existe um único P tal que $\text{path}(A) = P$.
   Como edge é uma função parcial definida para toda coluna de qualquer alinhamento A e como edge é unívoca[1], então path é também uma função definida para qualquer A e existe um único P tal que $\text{path}(A) = P$.

   (b) Para todo caminho P existe um único A tal que $\text{path}(A) = P$.
   Para provar esta afirmação precisamos provar que não existem dois alinhamentos A e A' distintos em $\Lambda_{s,t}$ tais que $\text{path}(A) = \text{path}(A') = P$ e que para todo caminho P em $\Pi_{(0,0),(n,m)}$ existe um alinhamento A tal que $\text{path}(A) = P$.

      i. Não existem dois alinhamentos A e A' distintos tais que $\text{path}(A) = \text{path}(A') = P$.
      Suponha por absurdo, que existam os alinhamento A e A' distintos tais que $\text{path}(A) = \text{path}(A') = P$. Seja k o menor índice tal que a coluna k de A é diferente da coluna k de A'. Como $\text{pref}(s, k-1, A) = \text{pref}(s, k-1, A')$ e $\text{pref}(t, k-1, A) = \text{pref}(t, k-1, A')$ então as colunas k de A e A' são de tipos diferentes. Logo $\text{edge}(A,\ k) \neq \text{edge}(A',\ k)$ e portanto $\text{path}(A) \neq \text{path}(A')$, o que é uma contradição com a hipótese que $\text{path}(A) = \text{path}(A') = P$.

[1]Uma função f é unívoca se quando $f(x) = y$ e $f(x) = z$ então $y = z$, para todo x.

ii. Para todo caminho P em $\Pi_{(0,0),(n,m)}$ existe um alinhamento A tal que $\text{path}(A) = P$.
Seja $P = (\epsilon_0, \epsilon_1, \ldots, \epsilon_{r-1})$ um caminho de $(0,0)$ a (n,m) em G. Seja a função $f : E \to \Sigma' \times \Sigma'$ definida por:
- $f(\epsilon_D^{(i,j)}) = (s[i], t[j])$,
- $f(\epsilon_H^{(i,j)}) = (-, t[j])$,
- $f(\epsilon_V^{(i,j)}) = (s[i], -)$.

Seja a matriz A $(2 \times r)$ definida da seguinte forma: $(A[0,k], A[1,k]) = f(\epsilon_k)$, para todo k tal que $0 \leq k \leq r-1$. Se A é um alinhamento então $\text{path}(A) = P$, pois f associa uma aresta de um grafo de edição a uma coluna de um alinhamento, analogamente como edge associa uma coluna de um alinhamento a uma aresta de um grafo de edição.
Para que A seja um alinhamento é necessário que:

A. $r \geq m$ e $r \geq n$.
Seja $a_k = \text{row}(\text{end}(\epsilon_k))$ o índice da linha de $\text{end}(\epsilon_k)$ e $b_k = \text{col}(\text{end}(\epsilon_k))$ o índice da coluna de $\text{end}(\epsilon_k)$, para qualquer aresta ϵ_k do caminho P. Sabemos que, para todo k tal que $1 \leq k \leq r-1$, $a_k - a_{k-1} = 1$ somente se ϵ_k é uma aresta diagonal ou vertical, e $b_k - b_{k-1} = 1$ somente se ϵ_k é uma aresta diagonal ou horizontal. Como $a_{r-1} - a_0 = n$ então há exatamente n arestas diagonais ou verticais em P e como $b_{r-1} - b_0 = m$ então há exatamente m arestas diagonais ou horizontais em P. Portanto o número r de arestas no caminho P é tal que $r \geq n$ e $r \geq m$.

B. Para toda coluna j da matriz A se $A[0,j] = A[1,j]$ então $A[0,j] \neq -$.
Isto é verdade pois $(-,-)$ não é elemento da imagem de f e a matriz A é construída de tal forma que toda coluna de A é um elemento da imagem de f.

C. Existem duas subseqüências $S = (i_1, i_2, \ldots, i_n)$ e $T = (j_1, j_2, \ldots, j_m)$ dos índices das colunas de A, tais que:
- $s = A[0,i_1]A[0,i_2]\ldots A[0,i_n]$,
- $t = A[1,j_1]A[1,j_2]\ldots A[1,j_m]$,
- $A[0,i] = -$ para toda coluna i de A, tal que $i \notin S$, e
- $A[1,j] = -$ para todo coluna j de A, tal que $j \notin T$.

Vamos demonstrar a seguir que existem seqüências S e T que satisfazem estas restrições.
Como o número de arestas diagonais e verticais de P é igual a n e o número de arestas diagonais e horizontais de P é igual a m, definiremos as seqüências S e T da seguinte forma: sejam $S = (i_1, i_2, \ldots, i_n)$ e $T = (j_1, j_2, \ldots, j_m)$ tais que, para cada coluna k de A, se ϵ_k é uma aresta diagonal ou vertical então $k \in S$ e se ϵ_k é uma aresta diagonal ou horizontal então $k \in T$.
Como P é um caminho de $(0,0)$ a (n,m) em G então $\text{row}(\text{end}(\epsilon_{i_1})) = 1$, $\text{row}(\text{end}(\epsilon_{i_n})) = n$ e $\text{row}(\text{end}(\epsilon_{i_l})) - \text{row}(\text{end}(\epsilon_{i_{l-1}})) = 1$, para todo l tal que $2 \leq l \leq n$, ou seja, $(\text{row}(\text{end}(\epsilon_{i_1})), \text{row}(\text{end}(\epsilon_{i_2})), \ldots, \text{row}(\text{end}(\epsilon_{i_n}))) = (1, 2, \ldots, n)$.
Como ϵ_{i_l} é uma aresta diagonal ou vertical então $A[0, i_l] = s[\text{row}(\text{end}(\epsilon_{i_l}))]$, para todo l tal que $1 \leq l \leq n$. Portanto $A[0,i_1]A[0,i_2]\ldots A[0,i_n] = s[\text{row}(\text{end}(\epsilon_{i_1}))]\ s[\text{row}(\text{end}(\epsilon_{i_2}))]$ $\ldots\ s[\text{row}(\text{end}(\epsilon_{i_n}))] = s[1]s[2]\ldots s[n] = s$. De modo análogo temos que $A[1,j_1]\ A[1,j_2]\ \ldots\ A[1,j_n] = t[\text{col}(\text{end}(\epsilon_{j_1}))]$ $t[\text{col}(\text{end}(\epsilon_{j_2}))]\ \ldots\ t[\text{col}(\text{end}(\epsilon_{j_n}))] = t[1]t[2]\ldots t[n] = t$.
Seja i uma coluna de A, tal que $i \notin S$. Logo ϵ_i é uma aresta horizontal e portanto $A[0,i] = -$.
Seja j uma coluna de A, tal que $j \notin T$. Logo ϵ_j é uma aresta vertical e portanto $A[i,j] = -$.
Portanto S e T, como definidos no início da demonstração, satisfazem as restrições do item 2(b)iiC.

Logo A é um alinhamento tal que $\text{path}(A) = P$. Assim, existe um alinhamento A para qualquer P tal que $\text{path}(A) = P$.

Assim, para todo caminho $P \in \Pi_{(0,0),(n,m)}$ existe um único alinhamento A tal que $\text{path}(A) = P$.

Portanto path é uma função bijetora e a segunda afirmação da proposição está demonstrada. ∎

Portanto, podemos associar cada alinhamento A de s e t a um caminho P de $(0,0)$ a (n,m) no grafo de edição de s e t e vice-versa.

A Figura 3.1 destaca um caminho de $(0,0)$ a $(3,3)$ num grafo de edição de 4 linhas e 4 colunas. O caminho representa o seguinte alinhamento de $s = s[1\,..\,3]$ e $t = t[1\,..\,3]$:

$$A = \begin{bmatrix} - & s[1] & s[2] & s[3] \\ t[1] & - & t[2] & t[3] \end{bmatrix}$$

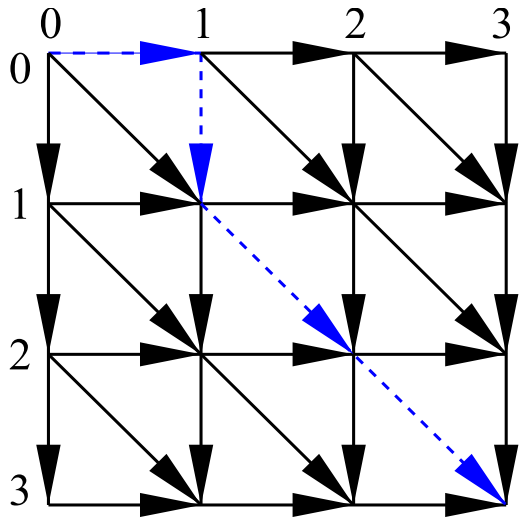

Figura 3.1: Arestas tracejadas indicam um caminho de $(0,0)$ a $(3,3)$ num grafo de edição de 4 linhas e 4 colunas.

Se o sistema de pontuação de um alinhamento de s e t é tal que, não considera como estão alinhadas as colunas anteriores ou posteriores para atribuir a pontuação de cada coluna do alinhamento, então podemos atribuir um peso a cada aresta de um grafo de edição $G = (V, E, \omega)$ de s e t de tal forma que, o peso de qualquer caminho P de $(0,0)$ a (n,m) em G será o peso do alinhamento correspondente a P. Por exemplo, isto é válido quando utilizamos o sistema de pontuação com custo de *gap* linear e atribuímos um peso a cada aresta da seguinte forma:

- se a aresta é $\epsilon_D^{(i,j)}$ então $\omega(\epsilon_D^{(i,j)}) = \varphi(s[i], t[j])$;
- se a aresta é $\epsilon_H^{(i,j)}$ então $\omega(\epsilon_H^{(i,j)}) = \varphi(-, t[j])$;
- se a aresta é $\epsilon_V^{(i,j)}$ então $\omega(\epsilon_V^{(i,j)}) = \varphi(s[i], -)$.

Podemos também associar o peso de uma aresta a pontuação de uma operação de edição da seguinte forma: o peso da aresta $e_V^{i,j}$ corresponde à

pontuação associada à operação de edição de remoção da letra $s[i]$ após o alinhamento de $s[1\,..\,i-1]$ e $t[1\,..\,j]$; o peso da aresta $e_H^{i,j}$ corresponde à pontuação associada à operação de edição de inserção da letra $t[j]$ após o alinhamento de $s[1\,..\,i]$ e $t[1\,..\,j-1]$ e; o peso da aresta $e_D^{i,j}$ corresponde à pontuação associada à operação de edição de substituição da letra $s[i]$ pela letra $t[j]$ após o alinhamento de $s[1\,..\,i-1]$ e $t[1\,..\,j-1]$. Esta associação permite a utilização do sistema de pontuação com custo de *gap* linear, além de outras possibilidades, e permite que o peso de qualquer caminho P de $(0,0)$ a (n,m) num grafo de edição seja o peso do alinhamento correspondente a P.

Portanto, para determinados sistemas de pontuação, podemos afirmar que encontrar um alinhamento ótimo de s e t é o mesmo que encontrar um caminho ótimo de $(0,0)$ a (n,m) no grafo de edição de s e t.

Vale a pena ressaltar que essa associação de pontuação de um alinhamento ao peso do caminho correspondente no grafo de edição, não é válida para o sistema de pontuação que utiliza penalidade para abertura de gaps (*gap open penalty*) descrito anteriormente, ou seja, o sistema de pontuação com *gap* afim. No entanto, freqüentemente neste texto, vamos assumir a utilização do sistema de pontuação com custo de *gap* linear e portanto poderemos nos referir a um alinhamento de s e t como um caminho no grafo de edição de s e t e vice-versa.

3.4 Alinhamento global, semiglobal e local

Nesta seção vamos considerar que o sistema de pontuação utilizado é o sistema de pontuação com custo de *gap* linear e que as seqüências s e t utilizadas nos alinhamentos são de comprimento n e m, respectivamente.

Muitas vezes um alinhamento de s e t é chamado de alinhamento global de s e t.

O conhecido artigo de Needleman e Wunsch [48] é geralmente considerado a primeira contribuição importante em procedimentos computacionais para a comparação de seqüências. O algoritmo proposto por eles neste artigo, apesar de não ter sido feita uma análise de tempo no artigo, é considerado cúbico ($O(n^2m)$ ou $O(nm^2)$) e, apesar da recorrência que vamos ver a seguir possibilitar um algoritmo com tempo de execução quadrático ($O(nm)$), normalmente o nome "Algoritmo de Needleman e Wunsch"é utilizado no algoritmo de programação dinâmica para alinhamentos globais [57].

Seja M a matriz $n+1 \times m+1$ tal que $M[i,j]$ contém a pontuação de um alinhamento global ótimo de $s[1\,..\,i]$ e $t[1\,..\,j]$. Uma maneira de construir a matriz M é através da seguinte recorrência:

$$\begin{array}{lll} M[0,0] & = 0 & \\ M[i,0] & = M[i-1,0] + \varphi(s[i],-) & ,0 < i \leq n \\ M[0,j] & = M[0,j-1] + \varphi(-,t[j]) & ,0 < j \leq m \\ M[i,j] & = \max \left(\begin{array}{l} M[i-1,j] + \varphi(s[i],-), \\ M[i,j-1] + \varphi(-,t[j]), \\ M[i-1,j-1] + \varphi(s[i],t[j]) \end{array} \right) & , \begin{array}{l} 0 < i \leq n \text{ e} \\ 0 < j \leq m \end{array} \end{array}$$

Portanto $M[n,m]$ é a pontuação de um alinhamento ótimo de s e t.

A obtenção do alinhamento ótimo de $s[1\,..\,n]$ e $t[1\,..\,m]$ é feita traçando-se o caminho reverso no grafo de edição $G = (V,E,\omega)$ de s e t, ou seja, analisando-se todas as arestas que chegam em (n,m) e obtendo-se a aresta $((i_1,j_1),(n,m))$ tal que $M[n,m] = M[i_1,j_1] + \omega(((i_1,j_1),(n,m)))$, depois analisa-se todas as arestas que chegam em (i_1,j_1) e obtém-se a aresta $((i_2,j_2),(i_1,j_1))$ tal que $M[i_1,j_1] = M[i_2,j_2] + \omega(((i_2,j_2),(i_1,j_1)))$, e assim por diante até chegar-se em $(0,0)$. Portanto, dado que o número de arestas que chegam em qualquer vértice é constante, podemos obter o alinhamento ótimo, com um pós-processamento, em tempo $O(n+m)$. Repare que deste modo precisamos manter a matriz M para obter o alinhamento ótimo e portanto o espaço requerido é $O(nm)$. Existe um método desenvolvido por Hirschberg [37] que utiliza espaço linear para obter o LCS de duas seqüências e que é comumente adaptado para obter o alinhamento ótimo sem a necessidade de espaço quadrático.

É comum que fragmentos de duas seqüências de DNA tenham alto grau de similaridade, enquanto que outros fragmentos destas seqüências tenham grau de similaridade baixo. Por isto, muitas vezes é interessante descobrir quais são os fragmentos que possuem alto grau de similaridade entre duas seqüências. Isto leva a necessidade da obtenção de um alinhamento local ótimo de s e t.

Um alinhamento local de s e t é um alinhamento global de qualquer fragmento de s e qualquer fragmento de t. Um alinhamento local ótimo é um alinhamento local com pontuação máxima dentre todos os possíveis alinhamentos locais de s e t.

Como já dissemos, um alinhamento global de s e t é um caminho de $(0,0)$ a (n,m) no grafo de edição G de s e t. Já um alinhamento local de s e t é

um caminho de qualquer vértice (i_1, j_1) a qualquer vértice (i_2, j_2) de G. O problema de encontrar um alinhamento local ótimo de s e t é o problema de encontrar o caminho de maior peso dentre todos os possíveis caminhos no grafo de edição de s e t.

Repare que, se todas as arestas do grafo de edição de s e t têm peso não negativo então o caminho de $(0, 0)$ a (n, m) é um alinhamento local ótimo. Portanto, neste caso, encontrar um alinhamento local ótimo é o mesmo que encontrar um alinhamento global ótimo. Para que haja uma diferença ente os dois tipos de alinhamentos, vamos assumir que algumas arestas do grafo de edição têm peso negativo. Freqüentemente, é considerado que as arestas associadas a *gaps* e *mismatches* têm peso negativo.

Smith e Waterman [58] desenvolveram um algoritmo que obtém um alinhamento local ótimo que executa em tempo $O(nm)$ e utiliza a seguinte recorrência:

$$
\begin{aligned}
M[0,0] &= 0 \\
M[i,0] &= \max \left(\begin{array}{l} 0, \\ M[i-1,0] + \varphi(s[i], -) \end{array} \right) && , 0 < i \leq n \\
M[0,j] &= \max \left(\begin{array}{l} 0, \\ M[0,j-1] + \varphi(-, t[j]) \end{array} \right) && , 0 < i \leq n \\
M[i,j] &= \max \left(\begin{array}{l} 0, \\ M[i-1,j] + \varphi(s[i], -), \\ M[i,j-1] + \varphi(-, t[j]), \\ M[i-1,j-1] + \varphi(s[i], t[j]) \end{array} \right) && , \begin{array}{l} 0 < i \leq n \text{ e} \\ 0 < j \leq m \end{array}
\end{aligned}
$$

Neste caso, o valor de $M[i, j]$ é a maior pontuação de um alinhamento ótimo de qualquer sufixo de $s[1 \,..\, i]$ e qualquer sufixo de $t[1 \,..\, j]$. A pontuação de um alinhamento local ótimo é o maior valor $M[i, j]$ entre todas as posições (i, j) de M.

Repare que a única diferença para a recorrência do algoritmo que obtém um alinhamento global ótimo é que foi incluída a possibilidade de todo elemento $M[i, j]$ ser no mínimo zero.

Seja G o grafo de edição de s e t. Ao invés de um alinhamento ótimo de s e t (alinhamento global ótimo) ou um alinhamento ótimo de qualquer fragmento de s e qualquer fragmento de t (alinhamento local ótimo), um alinhamento semiglobal ótimo é um alinhamento com maior pontuação dentre os seguintes alinhamentos ótimos:

1. Um alinhamento ótimo de s e qualquer fator de t, ou seja, um caminho ótimo entre $(0, j)$ e (n, j'), $0 \leq j \leq j' \leq m$. Neste caso, dizemos que qualquer prefixo de t e qualquer sufixo de t podem ser retirados para obter um alinhamento semiglobal ótimo.

2. Um alinhamento ótimo entre t e qualquer fator de s, ou seja, um caminho ótimo de $(i, 0)$ e (i', m), $0 \leq i \leq i' \leq n$. Neste caso, dizemos que qualquer prefixo de s e qualquer sufixo de s podem ser retirados para obter um alinhamento semiglobal ótimo.

3. Um alinhamento ótimo de qualquer prefixo de s e qualquer sufixo de t, ou seja, um caminho ótimo entre $(0, j)$ e (i, m), $0 \leq j \leq m$ e $0 \leq i \leq n$. Neste caso, dizemos que qualquer prefixo de t e qualquer sufixo de s podem ser retirados para obter um alinhamento semiglobal ótimo.

4. Um alinhamento ótimo de qualquer sufixo de s e qualquer prefixo de t, ou seja, um caminho ótimo entre $(i, 0)$ e (n, j), $0 \leq j \leq m$ e $0 \leq i \leq n$. Neste caso, dizemos que qualquer prefixo de s e qualquer sufixo de t podem ser retirados para obter um alinhamento semiglobal ótimo.

Nos alinhamentos semiglobais sufixos e/ou prefixos das seqüências podem ser retirados para obter um alinhamento semiglobal ótimo. Podemos também restringir as opções para a escolha de um alinhamento semiglobal ótimo, a somente um ou alguns alinhamentos dentre os 4 descritos acima e portanto, definir diferentes tipos de alinhamentos semiglobais ótimos. Por exemplo, podemos definir o alinhamento semiglobal ótimo de s e t como o alinhamento com maior pontuação entre o alinhamento ótimo de s e qualquer fator de t e o alinhamento ótimo de t e qualquer fator de s. Neste exemplo, o alinhamento semiglobal ótimo é o caminho com maior peso dentre todos os caminhos que inicia em qualquer vértice da primeira linha e termina em qualquer vértice da última linha, ou inicia em qualquer vértice da primeira coluna e termina em qualquer vértice da última coluna do grafo de edição de s e t.

De acordo com a possibilidade de retirada dos prefixos, a matriz M pode

ser construída com a seguinte recorrência:

$$
\begin{array}{lll}
M[0,0] & = 0 & \\
M[i,0] & = \max \left(\begin{array}{l} forgive_{s,t}(s,i), \\ M[i-1,0] + \varphi(s[i],-) \end{array} \right) & , 0 < i \leq n \\
M[0,j] & = \max \left(\begin{array}{l} forgive_{s,t}(t,j), \\ M[0,j-1] + \varphi(-,t[j]) \end{array} \right) & , 0 < i \leq n \\
M[i,j] & = \max \left(\begin{array}{l} M[i-1,j] + \varphi(s[i],-), \\ M[i,j-1] + \varphi(-,t[j]), \\ M[i-1,j-1] + \varphi(s[i],t[j]) \end{array} \right) & \begin{array}{l} 0 < i \leq n \text{ e} \\ 0 < j \leq m, \end{array}
\end{array}
$$

onde a função $forgive_{s,t} : \{s,t\} \times \mathbb{N} \to \{0, -\infty\}$ é tal que $forgive_{s,t}(w,i) = 0$ se o prefixo $w[1\,..\,i]$ pode ser retirado de w para obter um alinhamento semiglobal ótimo, e $forgive_{s,t}(s,i) = -\infty$ caso contrário.

O elemento $M[i,j]$, com a pontuação do alinhamento semiglobal ótimo, será o maior elemento na região Υ de posições de M, onde Υ é tal que:

- $(i,m) \in \Upsilon$ se o sufixo $s[i+1\,..\,n]$ pode ser retirado.
- $(n,j) \in \Upsilon$ se o sufixo $t[j+1\,..\,m]$ pode ser retirado.

Resumindo, dado o grafo de edição G de s e t, um alinhamento global ótimo de s e t é um caminho ótimo de $(0,0)$ a (n,m) em G; um alinhamento local ótimo é um caminho ótimo dentre todos os possíveis caminhos em G e; um alinhamento semiglobal ótimo é um caminho ótimo que sai da primeira linha ou da primeira coluna de G e chega na última linha ou na última coluna de G.

Repare que todo alinhamento semiglobal ótimo pode ser um alinhamento local ótimo, e que todo alinhamento global ótimo pode ser um alinhamento semiglobal ótimo.

Capítulo 4

Alinhamento com inversões

4.1 Introdução

Neste capítulo, consideraremos que s e t são duas seqüências de comprimentos n e m, respectivamente.

Alguns eventos biológicos típicos que são considerados normalmente nos procedimentos de alinhamentos atuais de seqüências de DNA são: substituição, remoção e inserção de nucleotídeos. Um outro evento biológico que ocorre, mas que normalmente não é considerado nos alinhamentos usuais, é a inversão, a qual iremos considerar sob algumas restrições nos algoritmos de alinhamentos deste capítulo.

Definição 4.1 (Alinhamento com inversões de s e t) *Um alinhamento com inversões de s e t é uma matriz A de 3 linhas e r colunas, tal que:*

- $\max(n, m) \leq r \leq m + n$;
- *para toda coluna k de A temos que $A[0, k] \in [1, n] \cup \{\_\}$, $A[1, k] \in [1, m] \cup \{\_\}$, $A[2, k] \in \{+, -\}$ e se $A[0, k] = A[1, k]$ então $A[0, k] \neq \_$;*
- *existe uma subseqüência $T = (k_1, k_2, \ldots, k_m)$ da seqüência de índices das colunas de A, tal que $(A[1, k_1], A[1, k_2], \ldots, A[1, k_m]) = (1, 2, \ldots, m)$ e $A[1, k] = \_$, para todo k, tal que $k \notin T$;*
- *existe uma e somente uma coluna k de A tal que $A[0, k] = i$, para cada i tal que, $1 \leq i \leq n$.*

O exemplo 4.2 mostra um alinhamento com inversões de duas seqüências de comprimento 10.

Exemplo 4.2 (Alinhamento com inversões)

$$A = \begin{bmatrix} 1 & \_ & 2 & 6 & 5 & \_ & 9 & 3 & 7 & 8 & 4 & 10 \\ 1 & 2 & 3 & \_ & 4 & 5 & 6 & 7 & 8 & 9 & \_ & 10 \\ + & + & + & - & - & - & - & - & + & + & + & + \end{bmatrix}$$

Se $A[0,k] = \_$ então associamos à coluna k uma operação de edição de inserção. Se $A[1,k] = \_$ então associamos à coluna k uma operação de edição de remoção. Se $A[0,k] \neq \_$ e $A[1,k] \neq \_$ então associamos à coluna k uma operação de edição de substituição.

Definição 4.3 (Transformação com inversões de s e t) *Dadas duas seqüências s e t, dizemos que uma transformação com inversões de s e t é qualquer seqüência de operações de edição $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ tal que $\phi_\Psi(s) = t$ e γ_k é uma operação de edição de inserção, remoção, substituição ou inversão, para todo k tal que $0 \leq k \leq r$.*

Em geral os sistemas de pontuação para alinhamentos consideram que a pontuação do alinhamento é a soma das pontuações das operações de edição da transformação associada, através de algum critério, ao alinhamento.

Porém, dado um alinhamento com inversões de s e t, obter uma transformação com inversões de s e t pode não ser uma tarefa simples, pois as operações de edição de inversão não estão explícitas no alinhamento. Como veremos, para alinhamentos com inversões não sobrepostas a identificação das operações de edição de inversão é mais fácil.

Contudo, dada uma transformação com inversões Ψ de s e t, facilmente associamos um alinhamento com inversões de s e t à Ψ. Portanto, muitas vezes é de interesse obter a transformação com inversões de s e t e a partir desta, obter o alinhamento com inversões associado e sua pontuação.

Dadas duas seqüências s e t e pontuações fixas para cada operação de edição associada a uma mutação, o problema de *obter uma transformação ótima com inversões de s e t* é um problema de otimização que procura uma transformação com inversões de s e t tal que a soma das pontuações das operações de edição desta transformação é a máxima possível. Algumas vezes, pode ser desejável apenas a obtenção da soma das pontuações das operações de edição da transformação ótima com inversões de s e t, ou seja, a pontuação da transformação ótima com inversões de s e t.

Vale a pena ressaltar que existem também outras possibilidades de eventos relacionados a inversão, como por exemplo:

- a *reversão-por*-2, que reverte a ordem de *dois símbolos consecutivos*;
- a *reversão*, que reverte a ordem de *qualquer segmento* de símbolos ao invés de um segmento de comprimento 2;

Em 1975, Wagner [64] estudou o problema de obter um alinhamento ótimo com reversões-por-2 e provou que ele admite uma solução polinomial se o custo de uma reversão-por-2 é nulo. Por outro lado, ele também provou que a obtenção de uma solução ótima é *NP-difícil*, se cada operação tem um custo positivo constante.

Como pode ser visto em [15], o problema de decisão associado ao problema de obter uma transformação ótima com inversões para um alfabeto ilimitado é NP-difícil

Dada a dificuldade de se resolver o problema de obter um alinhamento ótimo com inversões de forma eficiente, três estratégias principais têm sido consideradas:

1. inversões sem sobreposições;
2. ordenação de permutações sem sinal por reversões e;
3. ordenação de permutações com sinal por reversões.

A segunda estratégia (ordenação de permutações sem sinal por reversões), aplica-se bem a *seqüências de genes* e na comparação de genomas de mitocôndrias. Ela não se aplica a seqüências de nucleotídeos nem a seqüências de aminoácidos porque *repetições* de símbolos *não são* permitidas. Além disso, *nenhuma inserção* e *nenhuma remoção* são consideradas e a única operação permitida é a reversão, em que a *reversão* é definida para transformar uma seqüência tal como $1, 2, 3, 4, 5$ em $1,\ 4, 3, 2,\ 5$. O problema, também chamado de *ordenação de permutações sem sinal por reversões*, calcula a distância de edição de duas permutações considerando a operação de reversão. Neste caso, os dados são duas permutações de $1, 2, 3, \ldots, n$, onde n é o número de genes. Kececioglu e Sankoff [40] propuseram um algoritmo de 2-aproximação em 1995 e Christie [16] propôs um algoritmo de aproximação de razão 3/2 em 1998. De fato, Caprara [14] provou em 1999 que esse problema na verdade é NP-difícil.

A terceira estratégia é o problema chamado *ordenação de permutações com sinal por reversões*. Este é o mesmo problema de ordenação de permutações sem sinal por reversões até o ponto em que os sinais também são atribuídos a um gene e uma reversão também troca seu sinal. Por exemplo, uma reversão poderia transformar $1, 2, 3, 4, 5$ em $1, -4, -3, -2, 5$. Este sinal é normalmente associado à direção do gene (a qual filamento de DNA ele pertence). Hannenhalli e Pevzner [35] propuseram o primeiro algoritmo polinomial para o problema em 1995 e iniciaram uma seqüência de artigos baseados nessa estratégia. O algoritmo de Hannenhalli e Pevzner era $O(n^4)$ e foi melhorado para $O(n^2)$ por Kaplan, Shamir e Tarjan [38, 39] em 1997. Em 2004 Tannier e Sagot [59] propuseram um algoritmo subquadrático para este problema. Em 2001, Bader, Moret, e Yan [8] propuseram um algoritmo que calcula a distância de edição em $O(n)$ (a seqüência de reversões ainda requer $O(n^2)$). Estes métodos têm sido aplicados a estudos de reconstrução filogenética.

Em 2000, El-Mabrouk [24, 25] estudou a inclusão de duas operações: inserções e remoções de segmentos genéticos. Ela obteve resultados parciais e propôs uma solução polinomial exata para um caso e uma heurística polinomial com um testador polinomial para otimalidade no outro caso. Símbolos repetidos ainda não foram permitidos. Em 2002, El-Mabrouk [26] também obteve alguns resultados parciais ao considerar reversões e duplicações.

A terceira estratégia é principalmente usada no estudo de inversões de seqüências de genes. Novos resultados comparativos feitos por Sherer et al. [28] nas seqüências de DNA do homem e do chimpanzé, mostram também a importância do estudo da inversão ao nível da seqüência de DNA, onde esses métodos não podem ser aplicados. Por exemplo, Sherer et al. reportaram 83 seqüências reversas que estão contidas dentro de genes.

Em 1992, Schöniger e Waterman [55] introduziram uma hipótese simplificadora: todas as regiões envolvendo inversões não se sobrepõem, ou seja, a primeira estratégia. Esta simplificação é realista para comparação de seqüências de DNA relativamente próximas. Isto levou ao problema do *alinhamento com inversões não sobrepostas*, que será definido mais adiante. Eles apresentaram uma solução que leva tempo $O(n^6)$ para esse problema. Também introduziram uma heurística que reduz a complexidade para algo entre $O(n^2)$ e $O(n^4)$, dependendo dos dados de entrada. Essa heurística usa o algoritmo desenvolvido por Waterman e Eggert [65] que informa os k melhores alinhamentos locais não mutuamente intersectantes.

Em 2003, trabalhos independentes [20, 31] apresentaram algoritmos exa-

tos que resolvem o problema do alinhamento com inversões não sobrepostas em tempo $O(n^4)$ e memória $O(n^2)$.

Em 2005, do Lago et al.[21] propuseram um outro algoritmo que é uma implementação dinâmica esparsa que reduz o uso de recursos se $o(n^2)$ atribuições são dadas. Isto é freqüentemente esperado se a cardinalidade do alfabeto for grande, como por exemplo quando as letras são fragmentos de DNA de comprimento fixo.

Propomos dois algoritmos exatos que resolvem o problema do alinhamento com inversões não sobrepostas em tempo $O(n^3 \log n)$ [4] e $O(n^3)$ [62]. O algoritmo $O(n^3 \log n)$ permite maior flexibilidade no sistema de pontuação, mas os sistemas de pontuação usuais, com pesos inteiros e constantes, se usados no algoritmo $O(n^3)$ resultam numa complexidade cúbica no tempo de execução.

4.2 Alinhamento com inversões não sobrepostas

Nesta seção, algumas vezes aplicaremos a operação de edição de inversão sobre o alfabeto $\mathbb{N} \cup \{\_\}$. Consideraremos que se a é um símbolo tal que, $a \in \mathbb{N} \cup \{\_\}$, então $\overline{a} = a$.

Definição 4.4 (Alinhamento com inversões não sobrepostas de s e t) *Um alinhamento com inversões não sobrepostas de s e t é uma matriz A de 3 linhas e r colunas, tal que:*

- $\max(n, m) \leq r \leq m + n$;
- *para toda coluna k de A temos que $A[0, k] \in [1, n] \cup \{\_\}$, $A[1, k] \in [1, m] \cup \{\_\}$, $A[2, k] \in \{+, -, *\}$ e se $A[0, k] = A[1, k]$ então $A[0, k] \neq \_$;*
- *existe uma subseqüência $T = (k_1, k_2, \ldots, k_m)$ da seqüência de índices das colunas de A, tal que $(A[1, k_1], A[1, k_2], \ldots, A[1, k_m]) = (1, 2, \ldots, m)$ e $A[1, k] = \_$, para todo k, tal que $k \notin T$;*
- *existe uma seqüência de operações de edição de inversão Ψ tal que:*
  - *para cada operação de edição de inversão $\gamma = \pi_{i_1, i_2}$ em Ψ temos que $A[2, i_1] = *$ e $A[2, i] = -$, para todo i tal que $i_1 < i \leq i_2$, e*

  - *para cada coluna k de A, tal que $A[2,k] \in \{-,*\}$, existe uma e somente uma operação de edição $\gamma = \pi_{i_1,i_2}$ em Ψ tal que $i_1 \leq k \leq i_2$;*

- *existe uma subseqüência $S = (k_1, k_2, \ldots, k_n)$ da seqüência de índices de $s' = \phi_\Psi(A[0, 0\,..\,r-1])$ tal que $(s'[k_1], s'[k_2], \ldots, s'[k_n]) = (1, 2, \ldots, n)$ e $s'[k] = \_$, para todo k, tal que $k \notin S$.*

Diremos que o comprimento de um alinhamento com inversões não sobrepostas é a quantidade de colunas deste alinhamento.

O exemplo 4.5 mostra um alinhamento com inversões não sobrepostas de comprimento 12 de duas seqüências de comprimento 10 cada uma.

Exemplo 4.5 (Alinhamento com inversões não sobrepostas)

$$A = \begin{bmatrix} 1 & \_ & 2 & 6 & 5 & 4 & 3 & \_ & 9 & 8 & 7 & 10 \\ 1 & 2 & 3 & \_ & 4 & 5 & 6 & 7 & 8 & 9 & \_ & 10 \\ + & + & + & * & - & - & - & - & * & - & - & + \end{bmatrix}$$

Dado um alinhamento A com inversões não sobrepostas de s e t, diremos que as colunas que estão no intervalo $[k_1, k_2]$ são um trecho de inversão de A se $A[2,k_1] = *$, $A[2,k_2] = -$ e não existe uma coluna $k_2 + 1$ tal que $A[2, k_2 + 1] = -$.

Seja $\widehat{\Lambda}$ o conjunto de todos os possíveis alinhamentos com inversões não sobrepostas de duas seqüências.

Definição 4.6 (Função parcial simb) *Seja a função parcial $simb : \widehat{\Lambda} \times \{0,1\} \times \mathbb{N} \to \Sigma \cup \{\_\}$ definida por:*

- *$simb(A, 0, k) = s[A[0,k]]$ se $A[0,k] \neq \_$, $A[2,k] = +$ e k é o índice de uma coluna de A, onde A é um alinhamento com inversões não sobrepostas de s e t;*

- *$simb(A, 1, k) = t[A[1,k]]$ se $A[1,k] \neq \_$ e k é o índice de uma coluna de A, onde A é um alinhamento com inversões não sobrepostas de s e t;*

- *$simb(A, 0, k) = \overline{s[A[0,k]]}$ se $A[0,k] \neq \_$, $A[2,k] \in \{-,*\}$ e k é o índice de uma coluna de A, onde A é um alinhamento com inversões não sobrepostas de s e t;*

- *$simb(A, 0, k) = \_$ se $A[0,k] = \_$ e k é o índice de uma coluna de A;*

- *simb*$(A, 1, k) =$ _ *se* $A[1, k] =$ _ *e* k *é o índice de uma coluna de* A.

Seja Γ_P o conjunto com todas as possíveis operações de edição pontuais. Seja Γ_P^* o conjunto com todas as possíveis seqüências de operações de edição pontuais. Seja $\widehat{\Gamma}$ o conjunto com todas as possíveis operações de edição de inversão.

O operador $\cdot$ será utilizado para indicar a concatenação de seqüências de operações de edição. Por exemplo, $(\iota_{j-1,t[j]}) \cdot (\pi_{j,j+1}, \sigma_{j,t[j],\overline{s[i]}}, \rho_{j+1,\overline{s[i+1]}}) = (\iota_{j-1,t[j]}, \pi_{j,j+1}, \sigma_{j,t[j],\overline{s[i]}}, \rho_{j+1,\overline{s[i+1]}})$

Seja $\Gamma_1 = \{(\gamma_1)\} \cup \{(\gamma_2) \cdot \gamma_3\}$, onde $\gamma_1 \in \Gamma_P$, $\gamma_2 \in \widehat{\Gamma}$ e $\gamma_3 \in \Gamma_P^*$, ou seja, γ_1 é uma operação de edição pontual, γ_2 é uma operação de edição de inversão e γ_3 é uma seqüência de operações de edição pontuais. Se Ψ é um elemento de Γ_1 então Ψ ou é uma seqüência composta por uma operação de edição pontual, ou é uma seqüência composta por uma operação de edição de inversão concatenada com uma seqüência de operações de edição pontuais. Por exemplo, $(\iota_{j-1,t[j]}) \in \Gamma_1$ e $(\pi_{j,j+1}, \sigma_{j,t[j],\overline{s[i]}}, \rho_{j+1,\overline{s[i+1]}}) = (\pi_{j,j+1}) \cdot (\sigma_{j,t[j],\overline{s[i]}}, \rho_{j+1,\overline{s[i+1]}}) \in \Gamma_1$.

Definição 4.7 (Função parcial maxi) *Seja a função parcial maxi* : $\widehat{\Lambda} \times \{0,1\} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *definida por maxi*$(A, x, i, i') = k$*, onde* k *é o valor do maior elemento diferente de* _ *dentre os elementos de* $A[x, i\,..\,i']$*, ou é zero se não existirem elementos diferentes de* _ *em* $A[x, i\,..\,i']$*, para* $x \in \{0,1\}$.

Definição 4.8 (Função parcial mini) *Seja a função parcial mini* : $\widehat{\Lambda} \times \{0,1\} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *definida por mini*$(A, x, i, i') = k$*, onde* k *é o valor do menor elemento diferente de* _ *dentre os elementos de* $A[x, i\,..\,i']$*, ou é zero se não existirem elementos diferentes de* _ *em* $A[x, i\,..\,i']$*, para* $x \in \{0,1\}$.

Repare que zero não é um índice das seqüências s e t, pois o índice do primeiro elemento de qualquer uma destas seqüências é 1 (um).

Definição 4.9 (Função parcial end) *Seja a função parcial end* : $\widehat{\Lambda} \times \mathbb{N} \to \mathbb{N}$ *definida por end*$(A, k) = k'$ *se* $A[2, k] = *$*, onde* k' *é uma coluna de* A *tal que, para todo* l *tal que* $k + 1 \leq l \leq k'$*, temos que* $A[2, l] = -$ *e não existe uma coluna* $k' + 1$ *em* A *tal que* $A[2, k' + 1] = -$.

Por exemplo, se $A[2, 0..m] = (+, *, -, -, -, +)$ então end(A,k) só é definida para $k = 1$ e end$(A, 1) = 4$. Em outras palavras, end(A, k) é a coluna de A onde termina um trecho de inversão iniciado na coluna k de A.

A seguir vamos definir uma função parcial que associa uma coluna de um alinhamento com inversões não sobrepostas, a uma seqüência de operações de edição contendo uma ou duas operações de edição. Esta função parcial será utilizada para estabelecermos um sistema de pontuação para um alinhamento com inversões não sobrepostas.

Definição 4.10 (Função parcial opi) *Seja a função parcial* $opi : \widehat{\Lambda} \times \mathbb{N} \rightarrow \Gamma_1$ *tal que* $opi(A, k) = \Psi$ *se* k *é o índice de uma coluna de* A*, onde* A *é um alinhamento com inversões não sobrepostas de* s *e* t *e* Ψ *é uma seqüência de operações de edição tal que:*

- $\Psi = (\iota_{j-1,t[j]})$ *se* $A[0,k] = \_$ *e* $A[2,k] \neq *$*, onde* $j = A[1,k]$*,*
- $\Psi = (\sigma_{j,t[j],a})$ *se* $A[0,k] \neq \_$*,* $A[1,k] \neq \_$ *e* $A[2,k] \neq *$*, onde* $a = simb(A,0,k)$ *e* $j = A[1,k]$*,*
- $\Psi = (\rho_{j+1,a})$ *se* $A[1,k] = \_$ *e* $A[2,k] \neq *$*, onde* $a = simb(A,0,k)$ *e* $j = maxi(A,1,0,k-1)$*,*
- $\Psi = (\pi_{j,j+i'-i}, \iota_{j-1,t[j]})$ *se* $A[0,k] = \_$ *e* $A[2,k] = *$*, onde* $j = A[1,k]$*,* $i = mini(A,0,k,k')$*,* $i' = maxi(A,0,k,k')$ *e* $k' = end(A,k)$*,*
- $\Psi = (\pi_{j,j+i'-i}, \sigma_{j,t[j],a})$ *se* $A[0,k] \neq \_$*,* $A[1,k] \neq \_$ *e* $A[2,k] = *$*, onde* $a = simb(A,0,k)$*,* $j = A[1,k]$*,* $i = mini(A,0,k,k')$*,* $i' = maxi(A,0,k,k')$ *e* $k' = end(A,k)$*,*
- $\Psi = (\pi_{j,j+i'-i}, \rho_{j+1,a})$ *se* $A[1,k] = \_$ *e* $A[2,k] = *$*, onde* $a = simb(A,0,k)$*,* $j = maxi(A,1,0,k-1)$*,* $i = mini(A,0,k,k')$*,* $i' = maxi(A,0,k,k')$ *e* $k' = end(A,k)$*.*

Repare que a cada coluna associamos uma seqüência de operações de edição com uma ou duas operações de edição, sendo que a seqüência é composta por duas operações de edição, somente quando a coluna representar o início de um trecho invertido ($A[2,k] = *$). Neste caso, a primeira operação de edição da seqüência de operações de edição associada a coluna é sempre uma inversão.

Pode-se verificar que, dado um alinhamento com inversões não sobrepostas A de s e t de comprimento r, temos que $\phi_\Psi(s) = t$, onde $\Psi = \text{opi}(A,0) \cdot \text{opi}(A,1) \cdot \ldots \cdot \text{opi}(A,r-1)$.

Sejam Ψ uma seqüência de operações de edição pontuais ou de inversão e s' uma seqüência sobre a qual ϕ_Ψ é definida tal que, a cada símbolo de

s' está associada a sua posição em s', e se um símbolo a que está associado a posição i de s' é substituído por um símbolo b numa operação de edição de substituição ou pelo seu complementar $\overline{a}$ numa operação de edição de inversão, então associamos a este símbolo b ou $\overline{a}$ a mesma posição i de a. Dizemos que uma posição de s é envolvida em uma operação de edição de inversão γ de Ψ, se existe uma operação de edição de inversão γ em Ψ tal que, ao aplicarmos ϕ_Ψ sobre s' então γ envolve um símbolo associado a i. Dizemos que Ψ não tem inversões sobrepostas se ao aplicarmos ϕ_Ψ sobre s', qualquer posição i de s' é envolvida por no máximo uma única operação de edição de inversão γ em Ψ.

Definição 4.11 (Transformação com inversões não sobrepostas de s e t) *Dadas duas seqüências s e t, dizemos que uma transformação com inversões não sobrepostas de s e t é uma seqüência de operações de edição sem inversões sobrepostas $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ tal que $\phi_\Psi(s) = t$ e γ_k é uma operação de edição de inserção, remoção, substituição ou inversão, para todo k tal que $0 \leq k \leq r$.*

Dado um alinhamento com inversões não sobrepostas A de s e t, obter uma transformação com inversões não sobrepostas de s e t, é uma tarefa simples, pois tanto as operações de edição pontuais quanto as operações de edição de inversão necessárias para esta transformação estão explícitas no alinhamento. Em particular, a seqüência de operações de edição $\Psi^A_{\text{opi}} = \text{opi}(A, 0) \cdot \text{opi}(A, 1) \cdot \ldots \cdot \text{opi}(A, r-1)$, é uma transformação com inversões não sobrepostas de s e t, onde r é o comprimento de A. Temos também que, dada uma transformação com inversões não sobrepostas Ψ de s e t, podemos associar um alinhamento com inversões não sobrepostas de s e t à Ψ.

Dado um alinhamento com inversões não sobrepostas A de s e t de comprimento r, diremos que a seqüência $\Psi^A_{\text{opi}} = \text{opi}(A, 0) \cdot \text{opi}(A, 1) \cdot \ldots \cdot \text{opi}(A, r-1)$ é a *transformação com inversões não sobrepostas de s e t associada a A pela função parcial opi.*

Seja Γ o conjunto com todas as operações de edição e Γ^* o conjunto com todas as seqüências de operações de edição.

Definição 4.12 (Função ω_{op}) *Seja a função $\omega_{op} : \Gamma \cup \Gamma^* \to \mathbb{R}$ tal que:*

- *$\omega_{op}(\gamma)$ = pontuação da operação de edição γ, se γ é uma operação de edição.*
- *$\omega_{op}(\Psi) = \sum_{\gamma \in \Psi} \omega_{op}(\gamma)$, se Ψ é uma seqüência de operações de edição.*

Ou seja, dado uma operação de edição γ diremos que $\omega_{op}(\gamma)$ é a pontuação da operação de edição γ e, dado uma seqüência de operações de edição Ψ, diremos que $\omega_{op}(\Psi)$ é a pontuação da seqüência de operações de edição Ψ.

Definição 4.13 (Pontuação de alinhamento com inversões não sobrepostas) *Consideraremos que a* pontuação de um alinhamento com inversões não sobrepostas A de s e t *é igual a pontuação da transformação com inversões não sobrepostas de s e t associada a A pela função parcial opi.*

Ou seja, vamos considerar que a pontuação de um alinhamento com inversões não sobrepostas A de s e t é a soma das pontuações das operações de edição associadas a A através da função parcial opi.

Denotaremos por ω^A a pontuação de um alinhamento com inversões não sobrepostas A de s e t. Logo, consideraremos que $\omega^A = \omega_{op}(\Psi^A_{\text{opi}})$.

Um alinhamento ótimo A^* com inversões não sobrepostas de s e t é um alinhamento que tem pontuação máxima dentre todos os possíveis alinhamentos com inversões não sobrepostas de s e t. Diremos que $\omega^{A^*} = \omega_{op}(\Psi^{A^*}_{\text{opi}})$ é a pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t.

Seja Ψ uma transformação com inversões não sobrepostas de s e t. Pode-se verificar que podemos alterar a ordem de quaisquer operações de edição que estão em Ψ e, de acordo com a nova ordem das operações de edição, fazer algumas alterações nos índices e nos símbolos das operações de edição de Ψ para obter uma nova seqüência de operações de edição Ψ' tal que, Ψ' também é uma transformação com inversões não sobrepostas de s e t. Ou seja, as operações de edição de Ψ podem ser colocadas em qualquer ordem, alterando-se apenas os índices e símbolos das operações de edição, para obter uma outra transformação com inversões não sobrepostas de s e t.

Suponha um sistema de pontuação tal que, $-g < 0$ é a pontuação das operações de edição de inserção e remoção (*gaps*), $-r < 0$ é a pontuação das operações de edição de substituição de símbolos diferentes (*mismatches*), zero é a pontuação da operação de edição de substituição de um símbolo por ele mesmo (*match*) e $\omega_{inv} < 0$ é a pontuação de uma operação de edição de inversão. Seja Ψ uma transformação com inversões não sobrepostas de s e t de pontuação máxima, ou seja, não existe outra transformação Ψ_1 com inversões não sobrepostas de s e t tal que $\omega_{op}(\Psi_1) > \omega_{op}(\Psi)$. Utilizando esse sistema de pontuação, temos que $\omega_{op}(\Psi^{A^*}_{\text{opi}}) = \omega_{op}(\Psi)$, ou seja, a transformação com inversões não sobrepostas de s e t $\Psi^{A^*}_{\text{opi}}$ é uma transformação de pontuação

máxima dentre todas as possíveis transformações com inversões não sobrepostas de s e t.

Definição 4.14 (Função parcial $\mathbf{add_{j'}}$) *Seja a função parcial $add_{j'} : \Gamma \cup \Gamma^* \to \Gamma$, onde $j' \in \mathbb{Z}$, definida por:*

- $add_{j'}(\sigma_{j,a,b}) = \sigma_{j'+j,a,b}$, *se* $j' + j \geq 1$,
- $add_{j'}(\iota_{j-1,a}) = \iota_{j'+j-1,a}$, *se* $j' + j - 1 \geq 1$,
- $add_{j'}(\rho_{j+1,a}) = \rho_{j'+j+1,a}$, *se* $j' + j + 1 \geq 1$,
- $add_{j'}(\pi_{i,i'}) = \pi_{j'+i,j'+i'}$, *se* $j' + i \geq 1$,
- $add_{j'}(\delta_{i_1,i_2,i_3}) = \delta_{j'+i_1,j'+i_2,j'+i_3}$, *se* $j' + i_1 \geq 1$ *e* $j' + i_3 \geq 0$,
- $add_{j'}(\varepsilon_{i_1,i_2,i_3}) = \varepsilon_{j'+i_1,j'+i_2,j'+i_3}$, *se* $j' + i_1 \geq 1$ *e* $j' + i_3 \geq 0$,
- $add_{j'}(\Psi) = \Psi'$, *onde* $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ *é uma seqüência de operações de edição e* $\Psi' = (add_{j'}(\gamma_0), add_{j'}(\gamma_1), \ldots, add_{j'}(\gamma_r))$, *se* $add_{j'}$ *é definida para cada operação de edição de* Ψ.

Ou seja, a função parcial $\mathrm{add}_{j'}$ acrescenta o valor j' aos índices de uma ou mais operações de edição.

A seguir vamos associar a cada aresta de um grafo de edição estendido G de s e t, uma operação de edição de tal forma que um caminho de $(0,0)$ a (n,m) em G está associado a uma transformação Ψ^A_{opi} com inversões não sobrepostas de s e t. Estabelecemos um sistema de pontuação para as arestas de G de tal forma que, a pontuação de um caminho ótimo de $(0,0)$ a (n,m) em G é a pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t, ou seja, $\omega((0,0),(n,m)) = \omega^{A^*}$.

Definição 4.15 (Função $\widehat{\mathbf{op}}$) *Seja $G = (V, E, \omega)$ um grafo de edição estendido de s e t. Dado um critério para escolher um alinhamento ótimo dentre todos os alinhamentos ótimos de quaisquer duas seqüências, podemos associar a cada aresta de G uma operação de edição através da função $\widehat{op} : E \to \Gamma_1$ definida por:*

- $\widehat{op}(\epsilon_D^{(i,j)}) = (\sigma_{j,t[j],s[i]})$,
- $\widehat{op}(\epsilon_H^{(i,j)}) = (\iota_{j-1,t[j]})$,

- $\widehat{op}(\epsilon_V^{(i,j)}) = (\rho_{j+1,s[i]})$ *e*

- $\widehat{op}(\epsilon_{(i',j')}^{(i,j)}) = (\pi_{j'+1,j'+i-i'}) \cdot (add_{j'}(op(A,0)),\ add_{j'}(op(A,1)),\ \ldots,\ add_{j'}(op(A,r-1)))$, *onde* r *é o comprimento do alinhamento ótimo* A *de* $\overline{s[i'+1\,..\,i]}$ *e* $t[j'+1\,..\,j]$ *escolhido dentre os alinhamentos ótimos de* $\overline{s[i'+1\,..\,i]}$ *e* $t[j'+1\,..\,j]$ *sob o critério dado.*

Repare que a função $\widehat{\text{op}}$ não é injetora. Contudo, se as arestas ϵ e ϵ' são tais que $\epsilon \neq \epsilon'$ e $\widehat{\text{op}}(\epsilon) = \widehat{\text{op}}(\epsilon')$, então ϵ e ϵ' são arestas simples da mesma classe (diagonal, horizontal ou vertical) e chegam em vértices de uma mesma coluna de G. Por exemplo, para as arestas $\epsilon = \epsilon_H^{(i,j)}$ e $\epsilon' = \epsilon_H^{(i',j)}$ temos que $\widehat{\text{op}}(\epsilon) = \widehat{\text{op}}(\epsilon') = (\iota_{j-1,t[j]})$.

A seqüência $(\text{add}_{j'}(\text{op}(A,0)), \text{add}_{j'}(\text{op}(A,1)), \ldots, \text{add}_{j'}(\text{op}(A,r-1)))$, descrita acima, é a seqüência de operações de edição associadas a um alinhamento ótimo de $\overline{s[i+1\,..\,i']}$ e $t[j+1\,..\,j']$ com apenas seus índices alterados pela função parcial $\text{add}_{j'}$.

Dizemos que $\widehat{\Pi}_{(0,0),(n,m)}$ é o conjunto de todos os possíveis caminhos de $(0,0)$ a (n,m) num grafo de edição estendido de s e t.

Seja Γ_1^* o conjunto com todas as possíveis concatenações de seqüências de elementos que pertencem ao conjunto Γ_1. Por exemplo, se $\Psi_1 \in \Gamma_1$, $\Psi_2 \in \Gamma_1$ e $\Psi_3 \in \Gamma_1$ então $\Psi_1 \cdot \Psi_2 \cdot \Psi_3 \in \Gamma_1^*$.

Definição 4.16 (Função transf) *Seja a função* $transf : \widehat{\Pi}_{(0,0),(n,m)} \rightarrow \Gamma_1^*$ *definida por* $transf(P) = \widehat{op}(\epsilon_0) \cdot \widehat{op}(\epsilon_1) \cdot \ldots \cdot \widehat{op}(\epsilon_r)$*, onde* $P = (\epsilon_0, \epsilon_1, \ldots, \epsilon_r)$ *é um caminho de* $(0,0)$ *a* (n,m) *em* G*.*

Pode-se verificar que transf(P), onde P é um caminho de $(0,0)$ a (n,m) em um grafo de edição estendido de s e t, é uma transformação com inversões não sobrepostas de s e t.

Seja o grafo de edição estendido $G = (V, E, \overline{\omega})$ de s e t, onde a função $\overline{\omega} : E \rightarrow \mathbb{R}$ é definida por:

- $\overline{\omega}(\epsilon_D^{(i,j)}) = \omega_{op}(\widehat{\text{op}}(\epsilon_D^{(i,j)}))$,

- $\overline{\omega}(\epsilon_H^{(i,j)}) = \omega_{op}(\widehat{\text{op}}(\epsilon_H^{(i,j)}))$,

- $\overline{\omega}(\epsilon_V^{(i,j)}) = \omega_{op}(\widehat{\text{op}}(\epsilon_V^{(i,j)}))$ e

- $\overline{\omega}(\epsilon_{(i',j')}^{(i,j)}) = \omega_{op}(\widehat{\text{op}}(\epsilon_{(i',j')}^{(i,j)}))$.

Utilizando uma função equivalente a $\overline{\omega}$ para atribuir pesos às arestas de um grafo de edição estendido G qualquer, temos que, a pontuação de um caminho ótimo P de $(0,0)$ a (n,m) em G é igual à pontuação da transformação com inversões não sobrepostas transf(P) de s e t. Além disto temos que, a pontuação de um caminho ótimo de $(0,0)$ a (n,m) em G é igual à pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t. Ou seja,

$$\overline{\omega}(P) = \omega_{op}(\text{transf}(P)) = \omega^{A^*} = \omega_{op}(\Psi^{A^*}_{\text{opi}}),$$

onde P é um caminho ótimo de $(0,0)$ a (n,m) em G.

Em suma, se consideramos que a pontuação de um alinhamento com inversões não sobrepostas A de s e t é a pontuação da transformação com inversões não sobrepostas de s e t associada a A pela função parcial opi, e a função de peso das arestas de um grafo de edição estendido de s e t é uma função equivalente a $\overline{\omega}$, então o peso de um caminho ótimo de $(0,0)$ a (n,m) em G é igual à pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t.

Portanto, podemos encontrar a pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t, encontrando o peso de um caminho ótimo de $(0,0)$ a (n,m) num grafo de edição estendido de s e t. A partir de um caminho ótimo, podemos também encontrar um alinhamento ótimo.

No restante deste capítulo vamos desenvolver algoritmos para encontrar um caminho ótimo de $(0,0)$ a (n,m) num grafo de edição estendido G de s e t ao invés de encontrar explicitamente o alinhamento ótimo com inversões não sobrepostas de s e t. Muitas vezes estaremos interessados em obter apenas a pontuação de um caminho ótimo de $(0,0)$ a (n,m) em G. Em geral, utilizamos matrizes de programação dinâmica e algumas outras estruturas de dados para obter as pontuações dos caminhos ótimos. Estas estruturas de dados permitem, a partir delas, construir um caminho ótimo de $(0,0)$ a (n,m) em G em tempo $O(n^2)$ e espaço $O(n^2)$. Os algoritmos que vamos descrever têm tempo de execução $O(n^3 \log n)$ e $O(n^3)$ e precisam de espaço $O(n^2)$ para obter o peso de um caminho ótimo de $(0,0)$ a (n,m) num grafo de edição estendido G de s e t, ou seja, a pontuação de um alinhamento ótimo com inversões não sobrepostas de s e t. Portanto, um caminho ótimo de $(0,0)$ a (n,m) em G, ou seja, um alinhamento ótimo com inversões não sobrepostas de s e t, pode ser obtido em tempo $O(n^3 \log n)$ e $O(n^3)$ e utilizando espaço $O(n^2)$.

4.3 Caminhos ótimos em um grafo de edição

Seja $G = (V, E, \omega)$ o grafo de edição de s e t. Dados i e i', tais que $0 \leq i' \leq i \leq n$, queremos calcular o valor de $\omega((i', j'), (i, j))$, ou seja, o peso de um caminho ótimo de (i', j') a (i, j) em G, para todo j e j' tais que $0 \leq j' \leq j \leq m$.

Dados i e i', tais que $0 \leq i' \leq i \leq n$, diremos que $i - i' = n'$. Um algoritmo simples que calcula os valores de $\omega((i', j'), (i, j))$, para todo j e j' tais que $0 \leq j' \leq j \leq m$, é o seguinte: para cada j' o algoritmo calcula para todo j, tal que $j' \leq j \leq m$, o valor de $\omega((i', j'), (i, j))$. Com programação dinâmica simples é possível calcular, para um dado j', o valor de todos os $\omega((i', j'), (i, j))$ em tempo $O(mn')$. Portanto, para calcular $\omega((i', j'), (i, j))$ para todos os j' este algoritmo simples leva tempo $O(m^2 n')$.

Aggarwal e Park [3] e Apostolico et al. [6] desenvolveram um algoritmo recursivo que, dados um grafo de edição, i e i', tais que $0 \leq i' \leq i \leq n$, calcula todos os valores de $\omega((i', j'), (i, j))$ em tempo $O(mn' \log m)$.

Jeanette Schmidt [54] desenvolveu um algoritmo que, dados um grafo de edição, i e i', tais que $0 \leq i' \leq i \leq n$, constrói uma estrutura de árvores que armazenam todos os valores de $\omega((i', j'), (i, j))$. A construção desta estrutura de árvores leva tempo $O(mn' \log m)$ e o tempo de acesso ao valor de $\omega((i', j'), (i, j))$ é $O(\log m)$. A vantagem do algoritmo proposto por Jeanette Schmidt, em relação aos outros algoritmos mencionados anteriormente, é o fato dele ser incremental. Para construir a estrutura de árvores com os valores de $\omega((i', j'), (i, j))$ o algoritmo utiliza a estrutura de árvores com os valores de $\omega((i', j'), (i - 1, j))$. Dada a estrutura de árvores com os valores de $\omega((i', j'), (i - 1, j))$ o algoritmo leva tempo $O(m \log m)$ para construir a estrutura de árvores com os valores de $\omega((i', j'), (i, j))$.

De agora em diante nesta seção, veremos detalhes bastante técnicos que se encontram essencialmente em [54] e que descrevem como construir essa estrutura de árvores.

Lema 4.17 *Seja $G = (V, E, \omega)$ um grafo de edição de s e t. Sejam i e i' tais que $0 \leq i' \leq i \leq n$. Seja $W_G^{i',i}$ a matriz $(m+1) \times (m+1)$ tal que $W_G^{i',i}[j', j] = \omega((i', j'), (i, j))$, para todo j e j' tais que $0 \leq j' \leq j \leq m$. A matriz $W_G^{i',i}$ é uma matriz de monge inversa triangular superior.*

Prova. Sejam (i', j_1), (i', j_2), (i, j_3) e (i, j_4) vértices de G, tais que $0 \leq j_1 < j_2 \leq j_3 < j_4 \leq m$. Como é ilustrado na figura 4.1, existe um vértice v

onde os caminhos ótimos de (i', j_2) a (i, j_3) e de (i', j_1) a (i, j_4) se cruzam. Sejam $a = \omega((i', j_1), (i, j_3))$, $b = \omega((i', j_2), (i, j_4))$, $c = \omega((i', j_1), v)$, $d = \omega(v, (i, j_4))$, $e = \omega((i', j_2), v)$ e $f = \omega(v, (i, j_3))$. Suponha, por absurdo, que $W_G^{i',i}[j_1, j_3] + W_G^{i',i}[j_2, j_4] < W_G^{i',i}[j_1, j_4] + W_G^{i',i}[j_2, j_3]$. Logo teríamos:

$$\begin{aligned}
(c + d) + (e + f) &> a + b \\
c + d + e + f - a - b &> 0 \\
(c + f - a) + (d + e - b) &> 0 \\
c + f - a > 0 \text{ ou } (d + e - b) &> 0 \\
c + f > a \text{ ou } d + e &> b
\end{aligned}$$

Porém se $c + f > a$ (ou $d + e > b$) teríamos um caminho de (i', j_1) a (i, j_3) (ou de (i', j_2) a (i, j_4)) com peso $c + f > a$ (ou $d + e > b$). Isto é impossível pois a e b são caminhos ótimos de (i', j_1) a (i, j_3) e de (i', j_2) a (i, j_4) respectivamente.

Portanto $W_G^{i',i}[j_1, j_3] + W_G^{i',i}[j_2, j_4] \geq W_G^{i',i}[j_1, j_4] + W_G^{i',i}[j_2, j_3]$ para $0 \leq j_1 < j_2 \leq j_3 < j_4 \leq m$.

Além disso $\omega((i', j'), (i, j)) = -\infty$ se $0 \leq j < j' \leq m$.

Portanto $W_G^{i',i}$ é uma matriz de monge inversa triangular superior. ∎

Chamamos a matriz $W_G^{i',i}$ de matriz de pesos de caminhos ótimos em G entre as linhas i' e i.

Como conseqüência do Lema 4.17 pode-se calcular os máximos das colunas de $W_G^{i',i}$ em tempo linear, utilizando o Algoritmo 3.

Observação 4.18 *O valor de $\omega((i', j'), (i, j))$ pode ser obtido com a seguinte recorrência:*

$$\omega((i', j'), (i, j)) = \begin{cases}
-\infty & \text{,se } j' > j \\
0 & \text{,se } i' = i \text{ e } j' = j \\
\omega((i', j'), (i, j-1)) + \omega(\epsilon_H^{(i,j)}) & \text{,se } i' = i \text{ e } j' < j \\
\omega((i', j'), (i-1, j)) + \omega(\epsilon_V^{(i,j)}) & \text{,se } i' < i \text{ e } j' = j \\
\max \begin{pmatrix} \omega((i', j'), (i, j-1)) + \omega(\epsilon_H^{(i,j)}) \\ \omega((i', j'), (i-1, j-1)) + \omega(\epsilon_D^{(i,j)}) \\ \omega((i', j'), (i-1, j)) + \omega(\epsilon_V^{(i,j)}) \end{pmatrix} & \text{,se } i' < i \text{ e } j' < j
\end{cases}$$

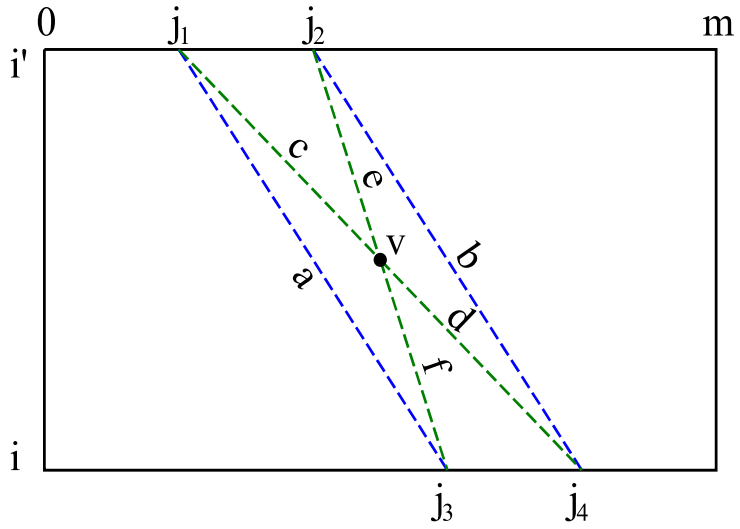

Figura 4.1: Ilustração da prova que $W_G^{i',i}$ é uma matriz de monge inversa

Definição 4.19 (Funções $\mathbf{hDif_G}$, $\mathbf{vDif_G}$ e $\mathbf{dDif_G}$) *Dado um grafo de edição $G = (V, E, \omega)$ de s e t, definimos as funções $hDif_G : V \times V \to \mathbb{R} \cup \{-\infty\}$, $vDif_G : V \times V \to \mathbb{R} \cup \{-\infty\}$ e $dDif_G : V \times V \to \mathbb{R} \cup \{-\infty\}$ da seguinte forma:*

- $hDif_G((i', j'), (i, j)) = \omega((i', j'), (i, j)) - \omega((i', j'), (i, j-1))$, *se* $j' < j$ *e* $i' \leq i$
- $vDif_G((i', j'), (i, j)) = \omega((i', j'), (i-1, j)) - \omega((i', j'), (i, j))$, *se* $j' \leq j$ *e* $i' < i$
- $dDif_G((i', j'), (i, j)) = \omega((i', j'), (i-1, j)) - \omega((i', j'), (i, j-1))$, *se* $j' < j$ *e* $i' < i$
- $hDif_G((i', j'), (i, j)) = -\infty$, *se* $j' \geq j$ *ou* $i' > i$
- $vDif_G((i', j'), (i, j)) = -\infty$, *se* $j' > j$ *ou* $i' \geq i$
- $dDif_G((i', j'), (i, j)) = -\infty$, *se* $j' \geq j$ *ou* $i' \geq i$

A Figura 4.2 ilustra os caminhos ótimos que são utilizados para obter as diferenças $\text{hDif}_G((i', j'), (i, j))$, $\text{vDif}_G((i', j'), (i, j))$ e $\text{dDif}_G((i', j'), (i, j))$ quando $j' < j$ e $i' < i$.

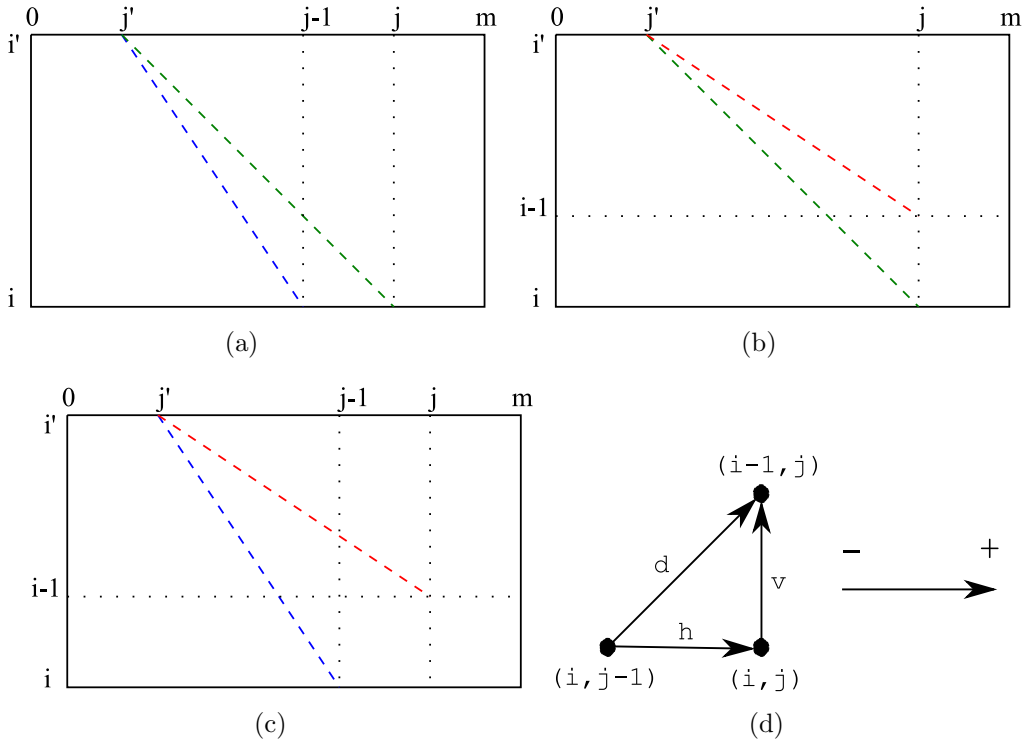

Figura 4.2: Ilustração dos caminhos ótimos utilizados nas funções hDif_G, vDif_G e dDif_G
4.2(a) $\text{hDif}_G((i',j'),(i,j)) = \omega((i',j'),(i,j)) - \omega((i',j'),(i,j-1))$,
4.2(b) $\text{vDif}_G((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) - \omega((i',j'),(i,j))$,
4.2(c) $\text{dDif}_G((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) - \omega((i',j'),(i,j-1))$ e
4.2(d) Esquema para representar as diferenças. O peso do caminho até o vértice da origem da seta é subtraído do peso do caminho até o vértice apontado pela seta.

A seguir vamos definir vetores que armazenam os valores das funções hDif_G, vDif_G e dDif_G para i' e (i,j) fixos.

Definição 4.20 (Vetores $\mathbf{hDif_G^{i',i,j}}$, $\mathbf{vDif_G^{i',i,j}}$ e $\mathbf{dDif_G^{i',i,j}}$) *Sejam G um grafo de edição de s e t, (i,j) um vértice de G e i' um índice de s.*

- *Se $i' \leq i$ então existe o vetor $hDif_G{}^{i',i,j}$ de comprimento j definido por $hDif_G{}^{i',i,j}[j'] = hDif_G((i',j'),(i,j))$, para todo j' tal que $0 \leq j' < j$.*
- *Se $i' < i$ então existe o vetor $vDif_G{}^{i',i,j}$ de comprimento $j+1$ definido por $vDif_G{}^{i',i,j}[j'] = vDif_G((i',j'),(i,j))$, para todo j' tal que $0 \leq j' \leq j$.*
- *Se $i' < i$ então existe o vetor $dDif_G{}^{i',i,j}$ de comprimento j definido por $dDif_G{}^{i',i,j}[j'] = dDif_G((i',j'),(i,j))$, para todo j' tal que $0 \leq j' < j$.*

Proposição 4.21 *Os vetores $hDif_G{}^{i',i,j}$, $vDif_G{}^{i',i,j}$ e $dDif_G{}^{i',i,j}$ são não decrescentes.*

Prova. Sejam G um grafo de edição de s e t, (i,j) um vértice de G e i' o índice de s tal que $i' \leq i$. Como a matriz $W_G^{i',i}$, definida no Lema 4.17, é uma matriz de monge inversa, temos que

$$\begin{aligned}
W_G^{i',i}[j'-1,j-1] + W_G^{i',i}[j',j] &\geq W_G^{i',i}[j',j-1] + W_G^{i',i}[j'-1,j] \\
W_G^{i',i}[j',j] - W_G^{i',i}[j',j-1] &\geq W_G^{i',i}[j'-1,j] - W_G^{i',i}[j'-1,j-1] \\
\omega((i',j'),(i,j)) - \omega((i',j'),(i,j-1)) &\geq \omega((i',j'-1),(i,j)) - \\
&\qquad \omega((i',j'-1),(i,j-1)) \\
\mathrm{hDif}_G((i',j'),(i,j)) &\geq \mathrm{hDif}_G((i',j'-1),(i,j)) \\
\mathrm{hDif}_G{}^{i',i,j}[j'] &\geq \mathrm{hDif}_G{}^{i',i,j}[j'-1]
\end{aligned}$$

Portanto $\mathrm{hDif}_G{}^{i',i,j}$ é não decrescente.

De forma similar é possível verificar que $\mathrm{vDif}_G{}^{i',i,j}$ é não decrescente. Como $\mathrm{dDif}_G{}^{i',i,j}[j'] = \mathrm{hDif}_G{}^{i',i,j}[j'] + \mathrm{dDif}_G{}^{i',i,j}[j']$ então $\mathrm{dDif}_G{}^{i',i,j}$ também é não decrescente. ∎

Proposição 4.22 *Dados um grafo de edição G de s e t e dois vértices, (i',j') e (i,j), de G tais que $i' < i$ e $j' < j$, então podemos dizer que*

1. *$\omega((i',j'),(i,j)) = \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)})$ se e somente se*

(a) $vDif_G((i',j'),(i,j-1)) \leq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)})$ *e*

(b) $dDif_G((i',j'),(i,j)) \leq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$

2. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$ *se e somente se*

   (a) $hDif_G((i',j'),(i-1,j)) \leq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$ *e*

   (b) $vDif_G((i',j'),(i,j-1)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)})$

3. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$ *se e somente se*

   (a) $hDif_G((i',j'),(i-1,j)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$ *e*

   (b) $dDif_G((i',j'),(i,j)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$

Prova. Utilizando as definições das funções hDif_G, dDif_G e vDif_G e a Observação 4.18 temos que:

1. $\omega((i',j'),(i,j)) = \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)})$ se e somente se
   $\omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \geq \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$ e
   $\omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \geq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$.

   Contudo, temos que

   $$\omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \geq \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \iff$$
   $$\omega((i',j'),(i,j-1)) - \omega((i',j'),(i-1,j-1)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_H^{(i,j)}) \iff$$
   $$-\mathrm{vDif}_G((i',j'),(i,j-1)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_H^{(i,j)}) \iff$$
   $$\mathrm{vDif}_G((i',j'),(i,j-1)) \leq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)})$$

   e

   $$\omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \geq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) \iff$$
   $$\omega((i',j'),(i,j-1)) - \omega((i',j'),(i-1,j)) \geq \omega(\epsilon_V^{(i,j)}) - \omega(\epsilon_H^{(i,j)}) \iff$$
   $$-\mathrm{dDif}_G((i',j'),(i,j)) \geq \omega(\epsilon_V^{(i,j)}) - \omega(\epsilon_H^{(i,j)}) \iff$$
   $$\mathrm{dDif}_G((i',j'),(i,j)) \leq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)}).$$

2. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$ se e somente se $\omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \geq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$ e $\omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \geq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)})$.

   Contudo, temos que

$$\begin{aligned}
&\omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \geq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) \iff \\
&\omega((i',j'),(i-1,j-1)) - \omega((i',j'),(i-1,j)) \geq \omega(\epsilon_V^{(i,j)}) - \omega(\epsilon_D^{(i,j)}) \iff \\
&-\mathrm{hDif}_G((i',j'),(i-1,j)) \geq \omega(\epsilon_V^{(i,j)}) - \omega(\epsilon_D^{(i,j)}) \iff \\
&\mathrm{hDif}_G((i',j'),(i-1,j)) \leq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})
\end{aligned}$$

   e

$$\begin{aligned}
&\omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \geq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \iff \\
&\omega((i',j'),(i-1,j-1)) - \omega((i',j'),(i,j-1)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)}) \iff \\
&\mathrm{vDif}_G((i',j'),(i,j-1)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)}).
\end{aligned}$$

3. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$ se e somente se $\omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) \geq \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$ e $\omega((i',j'),(i-1,j)) + q\omega(\epsilon_V^{(i,j)}) \geq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)})$.

   Contudo, temos que

$$\begin{aligned}
&\omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) \geq \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \iff \\
&\omega((i',j'),(i-1,j)) - \omega((i',j'),(i-1,j-1)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)}) \iff \\
&\mathrm{hDif}_G((i',j'),(i-1,j)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})
\end{aligned}$$

   e

$$\begin{aligned}
&\omega((i',j'),(i-1,j)) + q\omega(\epsilon_V^{(i,j)}) \geq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \iff \\
&\omega((i',j'),(i-1,j)) - \omega((i',j'),(i,j-1)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)}) \iff \\
&\mathrm{dDif}_G((i',j'),(i,j)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)}). \quad \blacksquare
\end{aligned}$$

A Proposição 4.22 dá condições para provar o Lema 4.23, que por sua vez é fundamental na construção eficiente das estruturas de árvores descritas por Jeanette Schmidt em [54].

Lema 4.23 *Dados um grafo de edição G de s e t, um vértice (i,j) de G e a linha i' de G tal que $i' \leq i$, então existem j_1 e j_2 tais que $0 \leq j_1 \leq j_2 \leq j$ e*

$$\omega((i',j'),(i,j)) = \begin{cases} \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) & \forall\ j' \mid 0 \leq j' < j_1 \\ \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) & \forall\ j' \mid j_1 \leq j' < j_2 \\ \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) & \forall\ j' \mid j_2 \leq j' \leq j \end{cases}$$

Prova. Sejam um grafo de edição G de s e t, um vértice (i,j) de G e a linha i' de G tal que $i' \leq i$. Estabelecidos os valores de j_1 e j_2, tais que $0 \leq j_1 \leq j_2 \leq j$, vamos provar que:

1. $\omega((i',j'),(i,j)) = \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)})$ para todo j' tal que $0 \leq j' < j_1$.
2. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$ para todo j' tal que $j_1 \leq j' < j_2$.
3. $\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$ para todo j' tal que $j_2 \leq j' \leq j$.

Seja j_1 o menor j' tal que

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}).$$

Repare que, pelo menos para $j' = j$ a desigualdade acima é satisfeita, pois $\omega((i',j),(i,j-1)) = -\infty$.

Portanto, para todo j', tal que $0 \leq j' < j_1$,

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}),$$

e está provado que a afirmação 1 está correta.

Seja j_2 o menor j' tal que $j_1 \leq j_2 \leq j$ e

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}).$$

Pela Proposição 4.22 temos que:

a) $\text{hDif}_G((i',j_2),(i-1,j)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$ e

b) $\text{dDif}_G((i',j_2),(i,j)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$.

Pela Proposição 4.21 os vetores $\text{hDif}_G{}^{i',i-1,j}$ e $\text{dDif}_G{}^{i',i,j}$ são não decrescentes. Logo, para todo j' tal que $j_2 \leq j' < j$ as desigualdades se mantém, ou seja:

a) $\text{hDif}_G((i',j'),(i-1,j)) \geq \omega(\epsilon_D^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$ e

b) $\text{dDif}_G((i',j'),(i,j)) \geq \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$.

Portanto, pela Proposição 4.22, para todo j' tal que $j_2 \leq j' < j$ temos que

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}).$$

Para o caso onde $j' = j$ temos que, pela Observação 4.18,

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}).$$

Portanto, para todo j' tal que $j_2 \leq j' \leq j$ temos que

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}),$$

e está provado que a afirmação 3 está correta.

Se $j_1 = j_2$ então a afirmação 2 está correta. Vamos supor que $j_1 < j_2$. Portanto, de acordo com a escolha de j_2, para todo j' tal que $0 \leq j' < j_2$, temos que

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}).$$

De acordo com a escolha de j_1 e pela Proposição 4.22 temos que:

a) $\text{vDif}_G((i',j_1),(i,j-1)) > \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_D^{(i,j)})$ ou

b) $\text{dDif}_G((i',j_1),(i,j)) > \omega(\epsilon_H^{(i,j)}) - \omega(\epsilon_V^{(i,j)})$

Como os dois vetores $\text{vDif}_G{}^{i',i,j-1}$ e $\text{dDif}_G{}^{i',i,j}$ são não decrescentes então, independente de qual das duas desigualdades seja verdadeira, temos que para todo j', tal que $j_1 \leq j' < j$,

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}).$$

Portanto, para todo j' tal que $j_1 \leq j' < j_2$ temos que

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}) \text{ e}$$

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}).$$

Logo, pela recorrência da Observação 4.18, para todo j' tal que $j_1 \leq j' < j_2$, temos que

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}).$$

Portanto, está provado que a afirmação 2 está correta. ∎

A idéia do Lema 4.23 é que a existência de j_1 e j_2 permite agrupar os índices j', de $\omega((i',j'),(i,j))$, em 3 grupos de acordo com a utilização das arestas horizontal, vertical e diagonal pelos caminhos ótimos que saem da linha i' e chegam em (i,j). Repare que a única aresta que obrigatoriamente é utilizada, quando $j' = j$, é a aresta vertical. A Figura 4.3 ilustra este agrupamento dos índices j'.

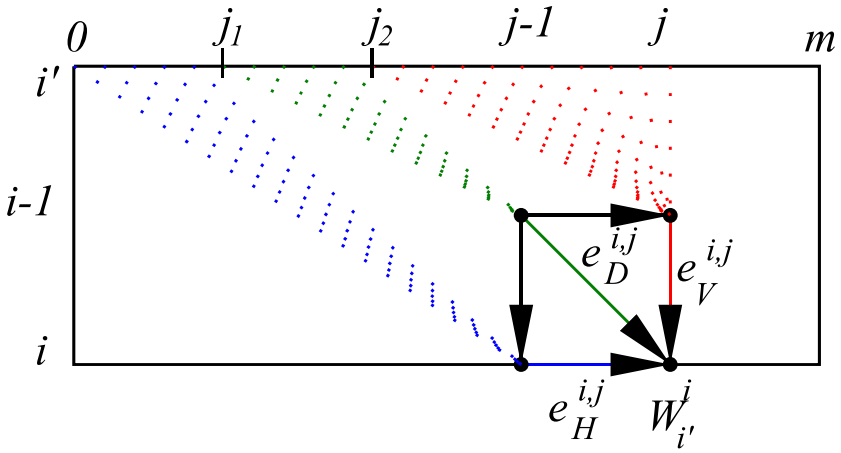

Figura 4.3: Os índices j_1 e j_2 agrupam os caminhos ótimos que chegam em (i,j) de acordo com a utilização das arestas $e_H^{i,j}$, $e_D^{i,j}$ e $e_V^{i,j}$

Na demonstração do Lema 4.23, foi escolhida uma coluna j_1^{max} de G cujo índice é o máximo possível de ser o j_1, ou seja, foi escolhido um j_1 tal que j_1 é o menor j' tal que $0 \leq j' \leq j$ e

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}).$$

Contudo, pode-se demonstrar que j_1 é tal que $j_1 \in [j_1^{min}, j_1^{max}]$, onde j_1^{min} é o menor j' tal que $0 \leq j' \leq j$ e

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}) \text{ ou}$$

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)}).$$

Do mesmo modo, após a escolha de j_1, podemos escolher qualquer j_2 tal que $j_2 \in [j_2^{min}, j_2^{max}]$, onde j_2^{min} (este foi o j_2 escolhido na demonstração do Lema 4.23) é o menor j' tal que $j_1 \leq j' \leq j$ e

$$\omega((i',j'),(i,j)) = \omega((i',j'),(i-1,j)) + \omega(\epsilon_V^{(i,j)})$$

e j_2^{max} é o menor j' tal que $j_1 \leq j' \leq j$,

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)}),$$

$$\omega((i',j'),(i,j)) \neq \omega((i',j'),(i,j-1)) + \omega(\epsilon_H^{(i,j)}) \text{ e}$$

$\forall j'' \mid j'' \in [j_1, j_2[$ temos $\omega((i',j''),(i,j)) = \omega((i',j''),(i-1,j-1)) + \omega(\epsilon_D^{(i,j)})$.

Repare que este modo para obter j_2^{min} e j_2^{max}, depende do valor escolhido de j_1. No entanto, é possível obter j_2^{min} e j_2^{max} sem depender do valor escolhido de j_1. Porém, para isto, a obtenção de j_1^{min} e j_1^{max} dependerá do valor escolhido de j_2.

Em suma, os índices j_1 e j_2 podem não ser únicos e existem intervalos $[j_1^{min}, j_1^{max}]$ e $[j_2^{min}, j_2^{max}]$ onde os índices j_1 e j_2, respectivamente, podem ser escolhidos.

Na linha 2 do Algoritmo 5, veremos como usar a Proposição 4.22 para calcular os valores j_1 e j_2 descritos no Lema 4.23.

4.4 Árvores para armazenar $\omega((i',j'),(i,j))$

Nesta seção, vamos descrever uma estrutura de árvores que armazena os valores de $\omega((i',j'),(i,j))$ de um grafo de edição G de s e t. Dados um índice i' de s e um vértice (i,j) de G tal que $i' \leq i$, cada árvore armazena os valores de $\omega((i',j'),(i,j))$ para todos os j' tal que $0 \leq j' \leq j$. Cada árvore, apesar de manter $j+1$ valores de $\omega((i',j'),(i,j))$, pode ser construída em tempo e espaço logarítmicos.

Definição 4.24 (Árvores para armazenar $\omega((i',j'),(i,j))$) *Sejam G um grafo de edição de s e t, (i,j) um vértice de G e i' o índice de uma linha de G tal que $i' \leq i$. Seja a árvore binária $B_G^{i',i,j}$ definida da seguinte forma:*

- *$B_G^{i',i,j}$ possui pesos nas arestas.*

- $B_G^{i',i,j}$ *tem* $j+1$ *folhas, tal que todas têm profundidade* $\lceil \log_2(j+1) \rceil$ *e são rotuladas seqüencialmente de* 0 *a* j*.*
- *A cada vértice* v *de* $B_G^{i',i,j}$ *associamos os seguintes atributos:*
  - esq_v*: o filho à esquerda de* v*,*
  - dir_v*: o filho à direita de* v*,*
  - h_v*: o comprimento do caminho de* v *até uma folha,*
  - p_v*: o peso do caminho da raiz até* v *e*
  - pe_v*: o peso do caminho de* v *até a folha mais a direita da subárvore esquerda de* v*, se esta subárvore for completa; e* $-\infty$ *se esta subárvore for incompleta.*
- *Para cada* j' *tal que* $0 \leq j' \leq j$*, o peso do caminho da raiz até a folha* j' *em* $B_G^{i',i,j}$ *é igual ao peso* $\omega((i',j'),(i,j))$ *em* G*.*

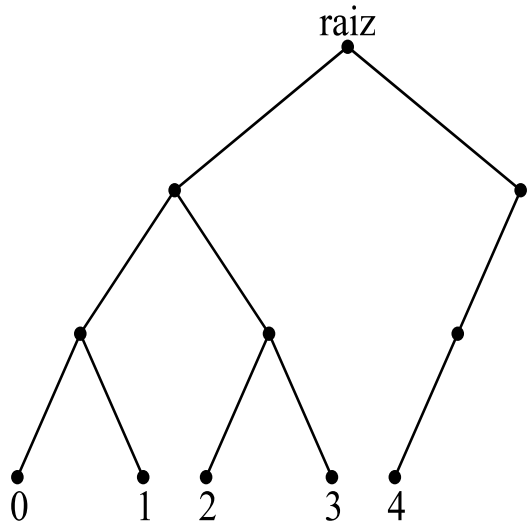

Figura 4.4: Exemplo de árvore binária $B_G^{i',i,4}$. Neste exemplo, não são mostrados os pesos das arestas e os atributos esq_v, dir_v, h_v, p_v e pe_v de cada vértice.

A Figura 4.4 ilustra a topologia de uma árvore $B_G^{i',i,j}$. Fixado um j, a topologia das árvores $B_G^{i',i,j}$ são todas iguais, para todos i e i' tais que $0 \leq i' \leq i \leq n$. A única diferença entre as árvores são os pesos nas arestas.

Como visto no Lema 4.23, existe pelo menos um j_1 tal que, para todo j' tal que $j' \in [0, j_1[$,

$$\omega((i', j'), (i, j)) = \omega((i', j'), (i, j-1)) + \omega(\epsilon_H^{(i,j)}).$$

Portanto, o valor de $\omega((i', j'), (i, j))$, para j' tal que $j' \in [0, j_1[$, é o peso do caminho da raiz até a folha j' na árvore $B_G^{i',i,j-1}$ mais $\omega(\epsilon_H^{(i,j)})$.

Assim, utilizando a subárvore de $B_G^{i',i,j-1}$ restrita aos caminhos da raiz até as folhas de 0 até $j_1 - 1$, podemos obter a subárvore de $B_G^{i',i,j}$ restrita aos caminhos da raiz até as folhas de 0 até $j_1 - 1$. Algumas poucas alterações nessa subárvore de $B_G^{i',i,j-1}$ bastam para contemplar o valor adicional $\omega(\epsilon_H^{(i,j)})$ e obter a subárvore análoga de $B_G^{i',i,j}$. Estas alterações podem ser feitas, adicionando-se este valor $\omega(\epsilon_H^{(i,j)})$ a toda aresta que parta de um vértice no caminho da raiz a j_1, e termine num vértice que não está neste caminho e é ancestral de uma folha j', onde j' é tal que $j' \in [0, j_1[$.

De forma equivalente é possível construir as subárvores de $B_G^{i',i,j}$ que contêm os caminhos da raiz até as folhas de j_1 até $j_2 - 1$ e de j_2 até m, com as subárvores análogas de $B_G^{i',i-1,j-1}$ e $B_G^{i',i-1,j}$, respectivamente.

A Figura 4.5 ilustra como fica uma árvore $B_G^{i',i,j}$ construída como descrito acima.

Assim, uma árvore $B_G^{i',i,j}$ pode ser construída, essencialmente, com subárvores das árvores $B_G^{i',i,j-1}$, $B_G^{i',i-1,j-1}$ e $B_G^{i',i-1,j}$.

O Algoritmo 5 constrói uma árvore binária $B_G^{i',i,j}$ a partir das árvores $B_G^{i',i,j-1}$, $B_G^{i',i-1,j-1}$ e $B_G^{i',i-1,j}$.

A partir daqui descreveremos como o Algoritmo 5 funciona.

Na linha 2 os valores de j_1 e j_2 são encontrados como descrito a seguir. Caminhe simultaneamente pelas árvores $B_G^{i',i,j-1}$, $B_G^{i',i-1,j-1}$ e $B_G^{i',i-1,j}$ e com os valores pe_v de cada árvore, calcule, para a folha j' mais à direita da subárvore esquerda de v, os valores de $\text{hDif}_G((i', j'), (i-1, j))$, $\text{dDif}_G((i', j'), (i, j))$ e $\text{vDif}_G((i', j'), (i, j-1))$. Desta forma, semelhante a uma busca binária, estenda para a direita ou para a esquerda os caminhos para j_1 e j_2 usando a Proposição 4.22.

Na linha 3 a construção de $B_G^{i',i,j}$ é iniciada construindo os caminhos p_1 e p_2 da raiz até j_1 e j_2 respectivamente, atribuindo peso zero para todas as arestas de p_1 e p_2, exceto na última aresta dos caminhos onde são colocados pesos: $\omega((i', j_1), (i-1, j-1)) + \omega(\epsilon_D^{(i,j)})$ para a aresta que chega em j_1 (se $j_1 \neq j_2$) e em seguida $\omega((i', j_2), (i-1, j)) + \omega(\epsilon_V^{(i,j)})$ para a aresta que chega

Algoritmo 5 Algoritmo que constrói uma árvore $B_G^{i',i,j}$

CONSTROIB($B_G^{i',i,j-1}, B_G^{i',i-1,j-1}, B_G^{i',i-1,j}, G, i, j$)

1 ▷ G é o grafo de edição onde estão as arestas $\epsilon_V^{(i,j)}$, $\epsilon_D^{(i,j)}$ e $\epsilon_H^{(i,j)}$
2 Encontra j_1 e j_2
3 Constrói a partir de r os caminhos p_1 e p_2 até j_1 e j_2 respectivamente
4 **para** cada $v \in p_1$ e $v \neq j_1$ **faça**
5 ▷ Coloca subárvores de $B_G^{i',i,j-1}$ à esquerda de p_1
6 ▷ Coloca subárvores de $B_G^{i',i-1,j-1}$ à direita de p_1
7 $v' \leftarrow$vértice de $B_G^{i',i,j-1}$ no caminho até j_1 cujo $h_v = h_{v'}$
8 $v'' \leftarrow$vértice de $B_G^{i',i-1,j-1}$ no caminho até j_1 cujo $h_v = h_{v'}$
9 **se** $esq_v = nil$ **então**
10 Coloca uma aresta esquerda em v cujo peso é
11 $p_{esq_{v'}} + \omega(\epsilon_H^{(i,j)})$ e chega em $esq_{v'}$
12 **senão**
13 Coloca uma aresta direita em v cujo peso é
14 $p_{dir_{v''}} + \omega(\epsilon_D^{(i,j)})$ e chega em $dir_{v''}$
15 Preenche os atributos de v (esq_v,dir_v,h_v,p_v e pe_v,)
16 **para** cada $v \in p_2$ e $v \neq j_2$ **faça**
17 ▷ Coloca subárvores de $B_G^{i',i-1,j-1}$ à esquerda de p_2
18 ▷ Coloca subárvores de $B_G^{i',i-1,j}$ à direita de p_2
19 $v' \leftarrow$vértice de $B_G^{i',i-1,j-1}$ no caminho até j_2 cujo $h_v = h_{v'}$
20 $v'' \leftarrow$vértice de $B_G^{i',i-1,j}$ no caminho até j_2 cujo $h_v = h_{v'}$
21 **se** $esq_v = nil$ **então**
22 Coloca uma aresta esquerda em v cujo peso é
23 $p_{esq_{v'}} + \omega(\epsilon_D^{(i,j)})$ e chega em $esq_{v'}$
24 **senão**
25 Coloca uma aresta direita em v cujo peso é
26 $p_{dir_{v''}} + \omega(\epsilon_V^{(i,j)})$ e chega em $dir_{v''}$
27 Preenche os atributos de v (esq_v,dir_v,h_v,p_v e pe_v,)
28 **devolva** r

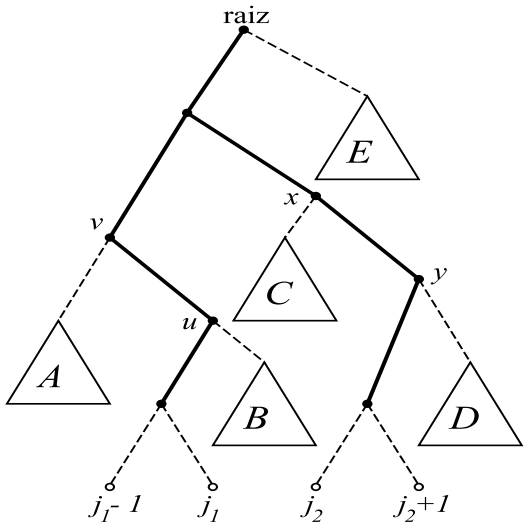

Figura 4.5: Exemplo de árvore binária $B_G^{i',i,j}$ construída a partir das árvores $B_G^{i',i,j-1}$ (subárvore A e folha $j_1 - 1$), $B_G^{i',i-1,j-1}$ (subárvores B e C) e $B_G^{i',i-1,j}$ (subárvores D e E e folha $j_2 + 1$).

em j_2.

A Figura 4.5 ilustra como fica uma árvore $B_G^{i',i,j}$ construída pelo Algoritmo 5 a partir das árvores $B_G^{i',i,j-1}$,$B_G^{i',i-1,j-1}$ e $B_G^{i',i-1,j}$. Seja v' o vértice em $B_G^{i',i,j-1}$ tal que $h_v = h_{v'}$. De modo análogo definimos os vértices u' e x' em $B_G^{i',i-1,j-1}$ e o vértice y' em $B_G^{i',i-1,j}$. A subárvore A é exatamente a mesma subárvore de $B_{i'}^{i,j-1}$ cuja raiz é o vértice $esq_{v'}$. Analogamente, as subárvores B e C são subárvores de $B_{i'}^{i-1,j-1}$ cujas raízes são os vértice $dir_{u'}$ e $esq_{x'}$, respectivamente; e a subárvore D é a subárvore de $B_{i'}^{i-1,j}$ cuja raiz é o vértice $dir_{y'}$. As folhas $j_1 - 1$ e $j_2 + 1$ de $B_{i'}^{i,j}$ são as folhas $j_1 - 1$ de $B_{i'}^{i,j-1}$ e $j_2 + 1$ de $B_{i'}^{i-1,j}$, respectivamente.

Para a construção de uma $B_G^{i',i,j}$ é necessário percorrer os caminhos até j_1 e até j_2 nas 4 árvores e ir acrescentando as subárvores correspondentes. Assim o tempo de execução do Algoritmo 5 é $O(\log m)$. São criados $O(\log m)$ vértices e arestas, assim a memória consumida é $O(\log m)$.

Se já tivermos calculado as m árvores $B_G^{i',i-1,j}$, as m árvores $B_G^{i',i-1,j-1}$e as m árvores $B_G^{i',i,j-1}$, levaremos tempo $O(m \log m)$ para construir as m árvores $B_G^{i',i,j}$. O Algoritmo 6 mostra como é feito a construção da matriz $W_G^{i',i}$ definida no Lema 4.17.

O tempo de acesso a um valor de $W_G^{i',i}[j',j]$ é $O(\log m)$. Porém, para obter os valores dos m elementos de uma coluna de $W_G^{i',i}[j',j]$ é necessário tempo $O(m)$.

Repare que, por uma questão de espaço e legibilidade do pseudo-código, não está contemplado no Algoritmo 5 o caso de construir uma árvore $B_G^{i',i,j}$, onde $i' = i$ ou $j = 0$. O algoritmo pode ser facilmente alterado para tratar estes casos.

Algoritmo 6 Representação da matriz $W_G^{i',i}$ por árvores binárias

CONSTROIW$(W_G^{i',i-1}, G, i', i)$
1 $W_G^{i',i}[0] \leftarrow constroiB(\emptyset, \emptyset, W_G^{i',i-1}[0], G, i, 0)$
2 **para** j **de** 1 **até** m **faça**
3 $\quad W_G^{i',i}[j] \leftarrow constroiB(W_G^{i',i}[j-1], W_G^{i',i-1}[j-1], W_G^{i',i-1}[j], G, i, j)$
4 **devolva** $W_G^{i',i}$

4.5 Algoritmo $O(n^3 \log n)$

Sejam $s = s[1\,..\,n]$ e $t = t[1\,..\,m]$ as seqüências a serem alinhadas.

Sejam $G = (V, E, \omega)$ o grafo de edição estendido de s e t e $\overline{G} = (V, \overline{E}, \overline{\omega})$ o grafo de edição de $\overline{s}$ e t. Repare que em $\overline{G}$, os pesos $\overline{\omega}(\overline{\epsilon}_H^{(i,j)})$, $\overline{\omega}(\overline{\epsilon}_D^{(i,j)})$ e $\overline{\omega}(\overline{\epsilon}_V^{(i,j)})$ correspondem, respectivamente, às pontuações das operações de edição de inserção de $t[j]$, substituição de $\overline{s[n+1-i]}$ por $t[j]$ e remoção de $\overline{s[n+1-i]}$. Um caminho ótimo em $\overline{G}$ de $(n-i, j)$ a $(n-i', j)$ representa um alinhamento ótimo entre $\overline{s[i'+1\,..\,i]}$ e $t[j'+1\,..\,j]$.

Consideraremos que a pontuação para uma operação de edição de inversão é fixa e independe do trecho sendo invertido. Denotaremos esta pontuação por ω_{inv}.

Vamos considerar que o peso de uma aresta estendida $\epsilon_{(i',j')}^{(i,j)}$ em G é igual ao peso de um caminho ótimo em $\overline{G}$ de $(n-i, j)$ a $(n-i', j)$ mais a pontuação de uma operação de edição de inversão, ou seja,

$$\omega(\epsilon_{(i',j')}^{(i,j)}) = \overline{\omega}((n-i, j), (n-i', j)) + \omega_{inv}.$$

A Figura 4.6 mostra a aresta estendida $\epsilon_{(i',j')}^{(i,j)}$ em G e uma representação do

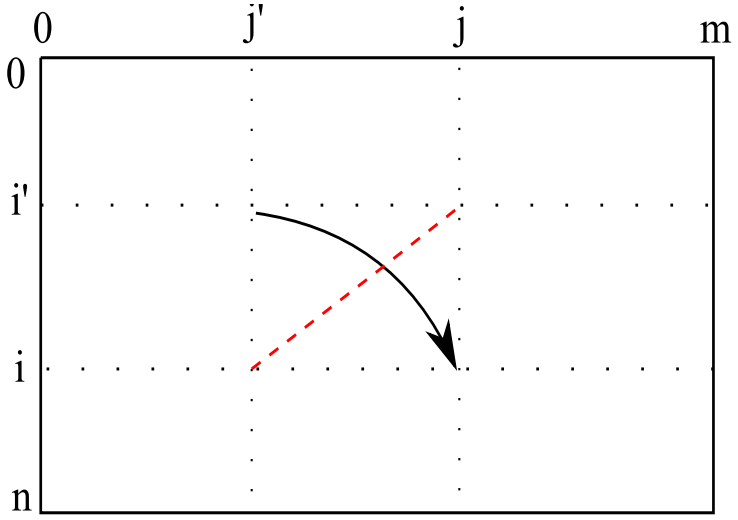

Figura 4.6: A seta é a aresta estendida $\epsilon^{(i,j)}_{(i',j')}$ em G e a linha tracejada representa o alinhamento ótimo entre $\overline{s[i'+1\,..\,i]}$ e $t[j'+1\,..\,j]$, ou seja, um caminho ótimo em $\overline{G}$ de $(n-i,j)$ a $(n-i',j)$.

alinhamento ótimo entre $\overline{s[i'+1\,..\,i]}$ e $t[j'+1\,..\,j]$, ou seja, uma representação de um caminho ótimo em $\overline{G}$ de $(n-i,j)$ a $(n-i',j)$.

Seja a matriz B $(n+1)\times(m+1)$ tal que $B[i,j]$ é a pontuação de um alinhamento ótimo com inversões não sobrepostas entre $s[1\,..\,i]$ e $t[1\,..\,j]$, para quaisquer i e j tais que $0\leq i\leq n$, $0\leq j\leq m$, ou seja, $B[i,j]$ é o peso de um caminho ótimo entre $(0,0)$ e (i,j) em G.

Seja $W^{n-i,n-i'}_{\overline{G}}$ a matriz de pesos de caminhos em $\overline{G}$ entre $n-i$ e $n-i'$ de acordo com a definição do Lema 4.17, ou seja, $W^{n-i,n-i'}_{\overline{G}}[j',j]$ é a pontuação de um alinhamento ótimo entre $\overline{s[i'+1\,..\,i]}$ e $t[j'+1\,..\,j]$. Portanto $\omega(\epsilon^{(i,j)}_{(i',j')}) = W^{n-i,n-i'}_{\overline{G}}[j',j]+\omega_{inv}$.

Podemos dizer que um caminho ótimo entre $(0,0)$ e (i,j) em G, para qualquer (i,j) tal que $(i,j)\neq(0,0)$, termina com uma aresta horizontal, vertical, diagonal ou estendida.

Ou seja, $B[0,0]=0$ e $\forall\ (i,j)\in V \mid (i,j)\neq(0,0)$ temos que

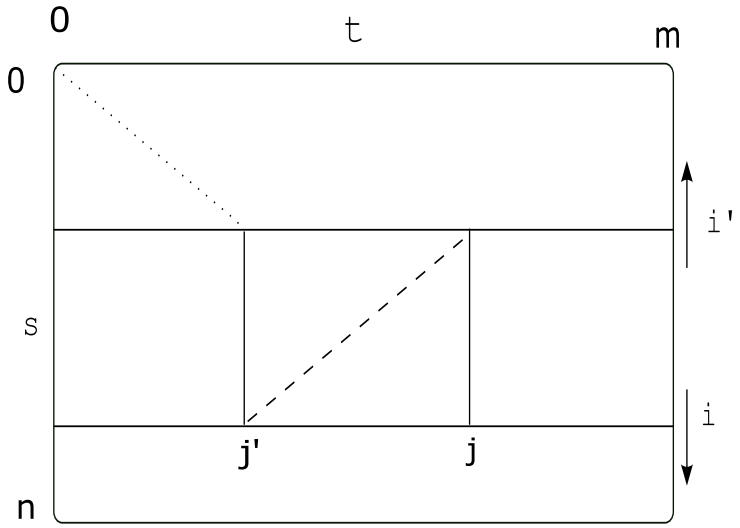

Figura 4.7: Ilustração da execução do Algoritmo 7. A linha pontilhada representa um caminho de $(0,0)$ a (i',j') em G. A linha tracejada representa o alinhamento de $\overline{s[i'+1\,..\,i]} \times t[j'+1\,..\,j]$

.

$$B[i,j] = \max \begin{cases} B[i-1,j] + \omega(\epsilon_V^{(i,j)}), & se \quad \exists \quad \epsilon_V^{(i,j)} \in E, \\ B[i,j-1] + \omega(\epsilon_H^{(i,j)}), & se \quad \exists \quad \epsilon_H^{(i,j)} \in E, \\ B[i-1,j-1] + \omega(\epsilon_D^{(i,j)}), & se \quad \exists \quad \epsilon_D^{(i,j)} \in E, \\ \max(B[i',j'] + \omega(\epsilon_{(i',j')}^{(i,j)})), & \forall\ \epsilon_{(i',j')}^{(i,j)} \in E. \end{cases}$$

O Algoritmo 7 constrói a matriz B. A sua execução é ilustrada na Figura 4.7 e é descrita a seguir.

O laço controlado pela linha 4 calcula os valores da linha i de B. Os valores de $B[i,j]$ calculados nas linhas 5 a 20 não consideram a operação de inversão. Estas linhas levam tempo $O(m)$ para serem executadas.

Na linha 25, para i e i' fixados, são calculados os valores dos alinhamentos ótimos de $\overline{s[i'+1\,..\,i]} \times t[j'+1\,..\,j]$, para todo j e j' tais que $0 \leq j' \leq j \leq m$. Para tanto, já se conhece $W_{\overline{G}}^{n-i,n-(i'+1)}$ da iteração anterior e portanto, como já visto na seção 4.3, a execução desta linha leva tempo $O(m \log m)$. Convém observar que neste algoritmo é usada a construção incremental de $W_{\overline{G}}^{n-i,n-(i'+1)}$ proposta por Jeanette em [54]. O acesso a qualquer elemento

Algoritmo 7 Algoritmo $O(n^3 \log n)$ para obtenção da matriz B

BIMN3LOGN(s, t)

```
1  Sejam G = (V, E, ω) o grafo de edição estendido de s e t e
2  ▷ Ḡ o grafo de edição de s̄ e t.
3  ▷ B[i, j] = ω((i', j'), (i, j)) em G
4  para i de 0 até |s| faça
5          se i = 0 então
6                  B[0, 0] ← 0
7          senão
8                  ▷ Na 1ª coluna ∄ arestas horizontais nem diagonais
9                  B[i, 0] ← B[i − 1, 0] + ω(ε_V^(i,j))
10         para j de 1 até |t| faça
11                 se i = 0 então
12                         ▷ Na 1ª linha ∄ arestas verticais nem diagonais
13                         B[0, j] ← B[0, j − 1] + ω(ε_H^(i,j))
14                 senão
15                         ▷ Obtém o máximo entre os caminhos que termi-
16                         ▷ nam nas arestas horizontal, vertical e diagonal
17                         h ← B[i, j − 1] + ω(ε_H^(i,j))
18                         v ← B[i − 1, j] + ω(ε_V^(i,j))
19                         d ← B[i − 1, j − 1] + ω(ε_D^(i,j))
20                         B[i, j] ← max(h, v, d)
21         ▷ Obtém, para cada j da linha i, o máximo entre os
22         ▷ caminhos que terminam numa aresta estendida
23         W_Ḡ^(n−i,n−(i'+1)) ← ∅
24         para i' de i descendo até 0 faça
25                 W_Ḡ^(n−i,n−i') ← ConstroiW(W_Ḡ^(n−i,n−(i'+1)), Ḡ, n − i, n − i')
26                 Define A^i_i'[1 .. |t|, 1 .. |t|] através de W_Ḡ^(n−i,n−i') e
27                   da linha i' de B
28                 ▷ A^i_i'[j', j] = W_Ḡ^(n−i,n−i')[j', j] + B[i', j'] + ω_inv
29                 ▷ MaxCol^i_i'[1 .. |t|] é o vetor de máximos de A^i_i'
30                 maxTotMonotonica(A^i_i', MaxCol^i_i')
31                 para j de 0 até |t| faça
32                         B[i, j] ← max(B[i, j], A^i_i'[MaxCol^i_i'[j], j])
33 devolva B
```

da matriz $W_{\overline{G}}^{n-i,n-i'}$, é feito em tempo $O(\log m)$.

A matriz $A_{i'}^{i}$ definida na linha 27 não é realmente construída, mas como o acesso a um elemento de $A_{i'}^{i}$ necessita do acesso a um elemento de $W_{\overline{G}}^{n-i,n-i'}$, o acesso a qualquer elemento de $A_{i'}^{i}$ é feito em tempo $O(\log m)$. O elemento $A_{i'}^{i}[j', j]$ é o peso de um caminho ótimo de $(0,0)$ a (i,j) em G que usa a aresta $\epsilon_{(i',j')}^{(i,j)}$. Como já vimos, a matriz $W_{\overline{G}}^{n-i,n-i'}$ é uma matriz de monge inversa. Dado i', temos que $B[i', j'] + \omega_{inv}$ é um vetor, portanto a matriz $A_{i'}^{i}$ é uma matriz resultante da soma de uma matriz de monge com um vetor. Logo $A_{i'}^{i}$ também é uma matriz de monge, ou seja, $A_{i'}^{i}$ é totalmente monotônica, conforme visto na seção 2.4. Portanto podemos obter os máximos de $A_{i'}^{i}$ com o Algoritmo 3 ($maxTotMonotonica$). O Algoritmo 3 faz $O(m)$ acessos a matriz, portanto a linha 30 leva tempo $O(m \log m)$. Após a chamada da função $maxTotMonotonica$, o elemento $MaxCol_{i'}^{i}[j]$ contém o índice j' associado ao máximo da coluna j em $A_{i'}^{i}$.

O laço controlado pela linha 31 analisa se o valor calculado em $A_{i'}^{i}[j', j]$ é maior do que todos os valores de $B[i, j]$ já encontrados anteriormente. Esse laço leva tempo $O(m)$.

Portanto, o laço controlado pela linha 24 leva tempo $O(nm \log m)$ e o algoritmo leva tempo $O(n^2 m \log m)$. Quando $m = O(n)$ o algoritmo tem tempo de execução $O(n^3 \log n)$.

4.6 Algoritmo $O(n^3)$

Sejam $s = s[1..n]$ e $t = t[1..m]$ as seqüências a serem alinhadas.

Sejam $G = (V, E, \omega)$ o grafo de edição estendido de s e t e $\overline{G} = (V, \overline{E}, \overline{\omega})$ o grafo de edição de $\overline{s}$ e t.

Consideraremos que a pontuação para uma operação de edição de inversão é uma constante e independe do trecho sendo invertido. Denotaremos esta pontuação por ω_{inv}.

Vamos considerar, nesta seção, que o peso de uma aresta estendida $\epsilon_{(i',j')}^{(i,j)}$ do grafo de edição estendido G de s e t é igual à pontuação de um alinhamento ótimo entre $\overline{s[i'+1..i]}$ e $t[j'+1..j]$ mais a pontuação de uma operação de edição de inversão, ou seja,

$$\omega(\epsilon_{(i',j')}^{(i,j)}) = \overline{\omega}((n-i, j'), (n-i', j)) + \omega_{inv}.$$

Seja a matriz B $(n+1) \times (m+1)$ tal que $B[i, j]$ é a pontuação de um alinhamento ótimo com inversões não sobrepostas entre $s[1..i]$ e $t[1..j]$,

para quaisquer i e j tais que $0 \leq i \leq n$, $0 \leq j \leq m$, ou seja, $B[i,j]$ é o peso de um caminho ótimo entre $(0,0)$ e (i,j) em G.

Podemos dizer que um caminho ótimo entre $(0,0)$ e (i,j) em G, para qualquer (i,j) tal que $(i,j) \neq (0,0)$, termina com uma aresta horizontal, vertical, diagonal ou estendida.

Ou seja, $B[0,0] = 0$ e $\forall\ (i,j) \in V \mid (i,j) \neq (0,0)$ temos que

$$B[i,j] = \max \begin{cases} B[i-1,j] + \omega(\epsilon_V^{(i,j)}), & se \quad \exists \quad \epsilon_V^{(i,j)} \in E, \\ B[i,j-1] + \omega(\epsilon_H^{(i,j)}), & se \quad \exists \quad \epsilon_H^{(i,j)} \in E, \\ B[i-1,j-1] + \omega(\epsilon_D^{(i,j)}), & se \quad \exists \quad \epsilon_D^{(i,j)} \in E, \\ \max(B[i',j'] + \omega(\epsilon_{(i',j')}^{(i,j)})), & \forall\ \epsilon_{(i',j')}^{(i,j)} \in E. \end{cases}$$

No algoritmo que iremos descrever nesta seção, vamos obter em tempo $O(n\psi)$, para um dado vértice (i,j), o vértice (i',j') tal que $(B[i',j']+\omega(\epsilon_{(i',j')}^{(i,j)}))$ é máximo, onde ψ é constante para muitos sistemas de pontuação para alinhamentos, inclusive aqueles comumente usados com pesos constantes. Juntamente com este vértice (i',j') vamos obter o valor $(B[i',j'] + \omega(\epsilon_{(i',j')}^{(i,j)}))$ e portanto o valor de $B[i,j]$. Logo, como existem mn vértices (i,j), vamos construir a matriz B em tempo $O(mn^2\psi)$.

Fixados o vértice (i,j) de um grafo de edição G qualquer e uma linha i' de G, tal que $i' \leq i$, sabemos pela Proposição 4.21 que $\text{hDif}_G((i',j'),(i,j))$ não decresce quando j' varia de 0 a $j-1$, ou seja, $\text{hDif}_G((i',j'),(i,j)) \geq \text{hDif}_G((i',j'-1),(i,j))$ para todo j' tal que $0 < j' < j$. Portanto, iremos dizer que $\text{hDif}_G((i',j'),(i,j))$ assume no máximo $\psi_G(i',i,j)$ valores distintos, sendo que $\psi_G(i',i,j)$ é tal que $1 \leq \psi_G(i',i,j) \leq j$.

Muitas vezes, os pesos das arestas de um grafo de edição são inteiros e constantes. Por exemplo, para qualquer grafo de edição G onde todas as arestas verticais e horizontais recebem peso constante $-g$ e as arestas diagonais recebem peso a ou $-g$, onde $a \geq 0$ e $g \geq 0$, temos que[1] $\omega((i',j'),(i,j)) \geq \omega((i',j'),(i,j-1)) - g$ e $\omega((i',j'),(i,j-1)) \geq \omega((i',j'),(i,j)) - g - a$, portanto $\omega((i',j'),(i,j)) - \omega((i',j'),(i,j-1)) \in [-g, a+g]$ e $\omega((i',j'),(i,j)) - \omega((i',j'),(i,j-1)) \in \mathbb{Z}$. Logo, $\psi_G(i',i,j) \leq a + 2g + 1$ e podemos dizer que $\psi_G(i',i,j)$ é constante.

[1] A maioria do sistemas de pontuação de alinhamento atribuem uma pontuação negativa (penalidade) para os *gaps* e *mismatches*, e uma pontuação positiva para os *matches*.

Definição 4.25 (Ponto de mudança de hDif$_G$ fixados i, i' e j) *Seja G um grafo de edição. Dados um vértice (i, j) e uma linha i' de G tais que $0 \leq i' \leq i \leq n$, diremos que j' é um ponto de mudança de $hDif_G$ fixados i, i' e j se $1 \leq j' \leq j-1$ e $hDif_G((i', j'), (i, j)) \neq hDif_G((i', j'-1), (i, j))$.*

Cada um destes pontos de mudança de hDif$_G$ são chamados em [43] de *Borderline point*.

Definição 4.26 (Lista $BL_G^{i',i,j}$) *Fixados o grafo de edição G e as linhas i' e i de G tais que $0 \leq i' \leq i \leq n$, para todo j de 1 a m definimos a lista $BL_G^{i',i,j}$ como a lista ordenada cujo primeiro elemento é o zero e os demais são os pontos de mudança de $hDif_G$ fixados i, i' e j.*

Dados um grafo de edição G, uma linha i' e um vértice (i, j) de G, podemos dizer que a lista $BL_G^{i',i,j}$ armazena os valores de j' onde inicia um novo valor de hDif$_G((i', j'), (i, j))$ e que o comprimento da lista $BL_G^{i',i,j}$ é $\psi_G(i', i, j)$[2].

A Figura 4.8 mostra um exemplo das listas $BL_G^{0,n,j}$ para diversos valores de j, onde G é um grafo de edição de $s = AATG$ e $t = TTCATGACG$ que utiliza um sistema de pontuação de *gap* linear. Com os valores do sistema de pontuação da Figura 4.8 temos que $\psi_G(i', i, j) \leq 4$.

Sejam as linhas i e i' fixadas. Suponha conhecidas as listas $BL_G^{i',i-1,j}$ para toda coluna j de G. Podemos obter em tempo e espaço $O(m)$ todas as $m+1$ listas $BL_G^{i',i,j}$, através de um algoritmo descrito em [54]. Além disto, este mesmo algoritmo também pode informar o valor de hDif$_G((i', j_1), (i, j))$ para cada elemento j_1 de $BL_G^{i',i,j}$.

Definição 4.27 (Função out) *Dados um grafo de edição estendido $G = (V, E, \omega)$ de s e t e um grafo de edição $\overline{G} = (V, \overline{E}, \overline{\omega})$ de $\overline{s}$ e t, a função $out : V \times V \to \mathbb{R} \cup \{-\infty\}$ é definida por*

$$out((i', j'), (i, j)) = \omega((0, 0), (i', j')) + \overline{\omega}((n - i, j'), (n - i', j))$$

Repare que se existe a aresta $\epsilon_{(i',j')}^{(i,j)}$ então

$$\text{out}((i', j'), (i, j)) + \omega_{inv} = \omega((0, 0), (i', j')) + \omega(\epsilon_{(i',j')}^{(i,j)}),$$

[2] $\psi_G(i', i, j)$ é limitado por uma constante que depende unicamente do sistema de pontuação para o alinhamento

Pesos de caminhos ótimos de $(0,j')$ a (n,j)

	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
j'=0	-4	-2	-2	-2	-3	-2	0	-1	-2	-3
j'=1	-	-4	-2	-2	-3	-1	1	0	-1	-2
j'=2	-	-	-4	-4	-2	0	2	1	0	-1
j'=3	-	-	-	-4	-2	0	2	1	0	0
j'=4	-	-	-	-	-4	-2	0	-1	-2	-1
j'=5	-	-	-	-	-	-4	-2	-2	-2	0
j'=6	-	-	-	-	-	-	-4	-2	-2	0
j'=7	-	-	-	-	-	-	-	-4	-4	-2
j'=8	-	-	-	-	-	-	-	-	-4	-2
j'=9	-	-	-	-	-	-	-	-	-	-4

hDif((0,j'),(n,j))

	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
j'=0	-	2	0	0	-1	1	2	-1	-1	-1
j'=1	-	-	**2**	0	-1	**2**	2	-1	-1	-1
j'=2	-	-	-	0	**2**	2	2	-1	-1	-1
j'=3	-	-	-	-	2	2	2	-1	-1	**0**
j'=4	-	-	-	-	-	2	2	-1	-1	**1**
j'=5	-	-	-	-	-	-	2	**0**	**0**	**2**
j'=6	-	-	-	-	-	-	-	**2**	0	2
j'=7	-	-	-	-	-	-	-	-	0	2
j'=8	-	-	-	-	-	-	-	-	-	2
j'=9	-	-	-	-	-	-	-	-	-	-

BLj

s=AATG	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
t=TTCATGACG	0	-	0	0	0	0	0	0	0	0
Peso do *gap* = -1	1	-	-	1	-	2	1	-	5	5	3
Peso do *mismatch* = -1	2	-	-	-	-	-	-	-	6	-	4
Peso do *match* = +1	3	-	-	-	-	-	-	-	-	-	5

Figura 4.8: Exemplo de *Borderline points*: os elementos de $\mathrm{hDif}((0,j'),(n,j))$ em negrito são *Borderline points*. As colunas j' onde ocorrem estes *Borderline points* estão no vetor BL_j. Os elementos do vetor BL_j são os elementos da lista $BL_G^{0,n,j}$, onde G é um grafo de edição de $s = AATG$ e $t = TTCATGACG$.

ou seja, o valor $\text{out}((i', j'), (i, j)) + \omega_{inv}$ é o peso máximo dentre os pesos dos caminhos de $(0, 0)$ a (i, j) em G que contêm a aresta estendida $\epsilon^{(i,j)}_{(i',j')}$.

A Figura 4.9 ilustra um caminho de $(0, 0)$ a (i, j) num grafo de edição estendido cujo peso é $\text{out}((i', j'), (i, j)) + \omega_{inv}$.

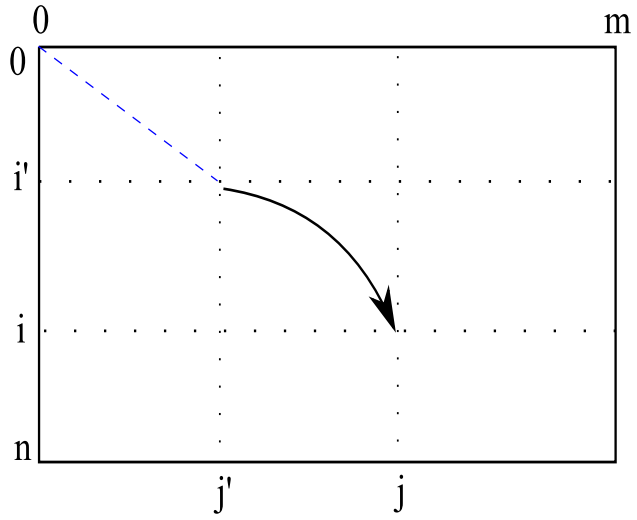

Figura 4.9: Representação de um caminho de $(0, 0)$ a (i, j) num grafo de edição estendido com peso $\text{out}((i', j'), (i, j)) + \omega_{inv}$. A linha tracejada representa um caminho ótimo de $(0, 0)$ a (i', j') e a seta é a aresta estendida $\epsilon^{(i,j)}_{(i',j')}$.

Proposição 4.28 *Para todo par de vértices (i', j') e (i, j) do grafo de edição estendido G de s e t, tais que $j' < j$ e $i' \leq i$, temos que*

$$out((i', j'), (i, j)) - out((i', j'), (i, j - 1)) = hDif_{\overline{G}}((n - i, j'), (n - i', j)).$$

Prova. Sejam (i', j') e (i, j) um par de vértices do grafo de edição estendido G de s e t, tais que $j' < j$ e $i' \leq i$.

Utilizando a definição da função $\text{hDif}_{\overline{G}}$ (definição 4.19) e a definição da

função out (definição 4.27), temos que,

$$\begin{aligned}
&\text{out}((i',j'),(i,j)) - \text{out}((i',j'),(i,j-1)) = \\
&\quad \omega((0,0),(i',j')) + \overline{\omega}((n-i,j'),(n-i',j)) - \\
&\qquad (\omega((0,0),(i',j')) + \overline{\omega}((n-i,j'),(n-i',j-1))) = \\
&\quad \overline{\omega}((n-i,j'),(n-i',j)) - \overline{\omega}((n-i,j'),(n-i',j-1)) = \\
&\quad \text{hDif}_{\overline{G}}((n-i,j'),(n-i',j)) \quad \blacksquare
\end{aligned}$$

Devido a esta proposição, podemos fazer a seguinte observação.

Observação 4.29 *Para todos três vértices (i',j_1), (i',j_2) e (i,j) do grafo de edição estendido G de s e t, tais que $j_1 < j_2 < j$ e $i' \leq i$, temos que $out((i',j_2),(i,j)) - out((i',j_1),(i,j)) - (out((i',j_2),(i,j-1)) - out((i',j_1),(i,j-1))) = hDif_{\overline{G}}((n-i,j_2),(n-i',j)) - hDif_{\overline{G}}((n-i,j_1),(n-i',j))$.*

Fixados um grafo de edição estendido G de s e t, um vértice (i,j) de G e uma linha i' de G, tal que $i' \leq i$, descreveremos a seguir como obter o valor máximo de $\text{out}((i',j'),(i,j))$ em tempo $O(\psi_{\overline{G}}(i',i,j))$. Para obter esta complexidade, iremos analisar somente as colunas j' de G tais que $\text{hDif}_{\overline{G}}((n-i,j'),(n-i',j)) \neq \text{hDif}_{\overline{G}}((n-i,j'),(n-i',j-1))$, onde $\overline{G}$ é o grafo de edição de $\overline{s}$ e t. Ou seja, iremos analisar somente as colunas j' de G que estão em $BL_{\overline{G}}^{n-i,n-i',j}$.

Proposição 4.30 *Seja G o grafo de edição estendido de s e t. Dados três vértices (i',j_1), (i',j_2) e (i,j) de G, tais que $i' \leq i$ e $j_1 < j_2 < j$ temos que:*

- *se $out((i',j_2),(i,j-1)) \geq out((i',j_1),(i,j-1))$ então $out((i',j_2),(i,j)) \geq out((i',j_1),(i,j))$,*
- *se $out((i',j_2),(i,j-1)) > out((i',j_1),(i,j-1))$ então $out((i',j_2),(i,j)) > out((i',j_1),(i,j))$.*

Prova. Sejam os vértices (i',j_1), (i',j_2) e (i,j) de G, tais que $i' \leq i$, $j_1 < j_2 < j$.

Para facilitar o leitura da demonstração, vamos dizer que:

- $\text{out}((i',j_2),(i,j)) = a$
- $\text{out}((i',j_2),(i,j-1)) = b$
- $\text{out}((i',j_1),(i,j)) = c$

- $\text{out}((i', j_1), (i, j-1)) = d$

Seja $\overline{G}$ o grafo de edição de $\overline{s}$ e t.
Pela Proposição 4.21, sabemos que

$$\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j)) \geq \text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j)).$$

Utilizando a Proposição 4.28, substituímos os valores de $\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j))$ e $\text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j))$ por $a-b$ e $c-d$, respectivamente, e obtemos

$$a - b \geq c - d, \text{ ou seja, } a - c \geq b - d.$$

Logo, se

$b \geq d$, ou seja, $b - d \geq 0$, então $a - c \geq 0$, ou seja, $a \geq c$ e se

$b > d$, ou seja, $b - d > 0$, então $a - c > 0$, ou seja, $a > c$. ∎

Com esta proposição podemos afirmar que uma coluna j_1, tal que $j_1 < j$, não poderá ser a coluna com o máximo $\text{out}((i', j'), (i, j))$, quando estão fixados i', i e j, se existir alguma outra coluna j_2 tal que $j_1 < j_2 < j$ e $\text{out}((i', j_2), (i, j-1)) > \text{out}((i', j_1), (i, j-1))$.

A seguir, vamos descrever estruturas e dar definições que são relativas a um grafo de edição estendido G de s e t e a linhas fixas i e i' de G, tais que $0 \leq i' \leq i \leq n$

Portanto, para facilitar a leitura e não sobrecarregar nos índices das definições e notações, vamos supor que G é um grafo de edição estendido de s e t e que i e i' são linhas dadas de G, tais que $i' \leq i$.

Definição 4.31 (Lista Out_j) *Fixadas as linhas i e i' de um grafo de edição estendido G, diremos que, para uma dada coluna j de G, Out_j é a lista com os valores de $out((i', j'), (i, j))$, para todo j' tal que $0 \leq j' \leq j$.*

Ou seja, dadas uma coluna j e as linhas i e i' de G, Out_j é a lista com todos os valores de $\text{out}((i', j'), (i, j))$ que são diferentes de $-\infty$.

Definição 4.32 (Candidato a máximo de Out_j) *Dada a coluna j de G, diremos que j_1 tal que, $0 \leq j_1 \leq j$, é um candidato a máximo de Out_j se não existe outra coluna j_2 de G tal que $j_1 < j_2 \leq j$ e $out((i', j_2), (i, j)) \geq out((i', j_1), (i, j))$.*

Repare que se j_1 é um candidato a máximo de Out_j e existe uma outra coluna j_2 tal que $\text{out}((i', j_2), (i, j)) = \text{out}((i', j_1), (i, j))$ então $j_2 < j_1$ e j_2 não é um candidato a máximo de Out_j. Logo, se j_1 é um candidato a máximo de Out_j tal que $\text{out}((i', j_1), (i, j))$ é máximo de Out_j, então até pode existir uma outra coluna j_2 tal que $\text{out}((i', j_2), (i, j)) = \text{out}((i', j_1), (i, j))$, porém neste caso $j_2 < j_1$.

Definição 4.33 (Lista de candidatos CL_j) *Dada a coluna j de G, diremos que CL_j é a lista ordenada com todos os candidatos a máximo de Out_j.*

A seguir seguem algumas observações sobre a lista CL_j decorrentes diretamente das definições da lista de candidatos CL_j e do candidato a máximo de Out_j.

Observação 4.34 *Seja j_1 tal que j_1 é o menor índice de coluna de G candidato a máximo de Out_j, ou seja, j_1 é o primeiro elemento de CL_j. Então $out((i', j_1), (i, j))$ é um máximo de Out_j.*

Observação 4.35 *Para toda coluna j de G temos que $j \in CL_j$.*

Observação 4.36 *Se j_1 é tal que $j_1 \notin CL_j$ e $0 \leq j_1 < j$ então existe um índice de coluna j_2 tal que $j_2 \in CL_j$, $j_2 > j_1$ e $out((i', j_2), (i, j)) \geq out((i', j_1), (i, j))$, pois, se $j_1 \notin CL_j$ então existe pelo menos um índice de coluna j_2 tal que $j_2 > j_1$ e $out((i', j_2), (i, j)) \geq out((i', j_1), (i, j))$, e pelo menos o maior dentre tais índices está em CL_j.*

Proposição 4.37 *Dados j e j_1, tais que $0 \leq j_1 < j \leq m$ temos que, se $j_1 \in CL_j$ então $j_1 \in CL_{j-1}$.*

Prova. Sejam j e j_1 tais que $j_1 \in CL_j$ e $0 \leq j_1 < j \leq m$.

Suponha por absurdo que $j_1 \notin CL_{j-1}$. Portanto, existe alguma coluna j_2 tal que $j_1 < j_2 \leq j - 1$ e $\text{out}((i', j_2), (i, j - 1)) \geq \text{out}((i', j_1), (i, j - 1))$. Pela Proposição 4.30 então $\text{out}((i', j_2), (i, j)) \geq \text{out}((i', j_1), (i, j))$, o que é uma contradição com a hipótese que j_1 é um candidato a máximo de Out_j. Portanto a suposição que $j_1 \notin CL_{j-1}$ é falsa. ∎

Devido a esta proposição, sabemos que todos os candidatos a máximo de Out_j, exceto o candidato j, são candidatos a máximo de Out_{j-1}.

Devido a Proposição 4.37 e a Observação 4.35, podemos fazer a observação a seguir.

Observação 4.38 *Sempre existe um e somente um novo elemento em CL_j em relação à CL_{j-1}: o próprio j.*

Assim, para toda coluna j_1 de G, existe uma coluna j_2 de G tal que:

- $j_2 \geq j_1$,
- para todo j tal que $j \in [j_1, j_2]$ temos que $j_1 \in CL_j$ e
- para todo j' tal que $j' \in [0, j_1[\cup]j_2, m]$, temos que $j_1 \notin CL_{j'}$.

A seguir vamos ver quais os candidatos a máximo de Out_{j-1} se mantêm candidatos a máximo de Out_j.

Lema 4.39 *Sejam as colunas j e j_1 de G, tais que $0 < j \leq m$ e $j_1 \in CL_{j-1}$. Temos que, $j_1 \in CL_j$ se e somente se:*

1. *não existe uma coluna j_2 de G tal que $j_2 \in CL_{j-1}$, $j_2 > j_1$ e $out((i', j_2), (i, j-1)) - out((i', j_1), (i, j-1)) + hDif_{\overline{G}}((n-i, j_2), (n-i', j)) - hDif_{\overline{G}}((n-i, j_1), (n-i', j)) \geq 0$.*
2. *$out((i', j), (i, j)) - out((i', j_1), (i, j)) < 0$.*

Prova. Primeiro vamos provar que se as condições 1 e 2 são satisfeitas então $j_1 \in CL_j$. Em seguida vamos provar que se $j_1 \in CL_j$ então as condições 1 e 2 são satisfeitas.

Para facilitar a leitura da demonstração, vamos dizer que:

- $\text{out}((i', j_2), (i, j)) = a$
- $\text{out}((i', j_2), (i, j-1)) = b$
- $\text{out}((i', j_1), (i, j)) = c$
- $\text{out}((i', j_1), (i, j-1)) = d$

1. Se as condições 1 e 2 são satisfeitas então $j_1 \in CL_j$.

   Suponha que as condições 1 e 2 são satisfeitas.

   Como $j_1 \in CL_{j-1}$ então $j_1 < j$ e portanto para que $j_1 \in CL_j$ é necessário demonstrar que não existe j_2 tal que $a \geq c$ e $j_1 < j_2 \leq j$.

   Vamos dividir a demonstração em duas partes e demonstrar que para todo j_2 tal que $j_2 \in]j_1, j[$ temos que $a < c$, e que para $j_2 = j$ temos que $a < c$.

(a) não existe j_2 em $]j_1, j[$ tal que $a \geq c$.
Pela condição 1 não existe uma coluna j_2 tal que $j_2 \in CL_{j-1}$, $j_2 > j_1$ e

$$b-d+\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j))-\text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j)) \geq 0.$$

Utilizando a Proposição 4.28 temos que:

$$b - d + a - b - (c - d) \geq 0, \text{ ou seja, } a - c \geq 0.$$

Portanto não existe uma coluna j_2 tal que $j_2 \in CL_{j-1}$, $j_2 > j_1$ e $a \geq c$.
Agora, suponha por absurdo que existe um j_2 tal que $j_2 \notin CL_{j-1}$, $j_1 < j_2 < j$ e $a \geq c$.
Sabemos pela Observação 4.35 que $j - 1 \in CL_{j-1}$. Como $j_2 \notin CL_{j-1}$ temos que $j_2 \neq j - 1$ e portanto $j_2 < j - 1$ e, pela Observação 4.36, existe uma coluna j_3 de G tal que $j_3 \in CL_{j-1}$, $j_3 > j_2$ e $\text{out}((i', j_3), (i, j - 1)) \geq b$.
Como $a \geq c$ temos que

$$a - c \geq 0, \text{ ou seja, } a - c + b - b + d - d \geq 0, \text{ ou seja,}$$

$$b-d+\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j))-\text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j)) \geq 0.$$

Como $j_3 > j_2 > j_1$ e $\text{out}((i', j_3), (i, j - 1)) \geq b$, temos que $\text{out}((i', j_3), (i, j-1))-d+\text{hDif}_{\overline{G}}((n-i, j_3), (n-i', j))-\text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j)) \geq 0$. Porém, isto é uma contradição com a condição 1 e portanto não existe j_2 tal que $j_2 \notin CL_{j-1}$, $j_1 < j_2 < j$ e $a \geq c$.

(b) se $j_2 = j$ então $a < c$.
Devido a condição 2, temos que

$$a - c < 0.$$

Portanto, se $j_2 = j$ temos que $a < c$.

2. Se $j_1 \in CL_j$ então as condições 1 e 2 são satisfeitas.

Seja j_1 tal que $j_1 \in CL_j$, ou seja, pela definição de candidato a máximo de Out_j, temos que, não existe outra coluna j_2 de G tal que $j_1 < j_2 \leq j$ e $\text{out}((i', j_2), (i, j)) \geq \text{out}((i', j_1), (i, j))$.

Portanto a condição 2 é satisfeita.

Utilizando a Proposição 4.28 e a definição de candidato a máximo de Out_j, podemos dizer que não existe outra coluna j_2 de G tal que $j_1 < j_2 < j$ e

$b+\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j)) \geq d+\text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j))$, ou seja,

$b + \text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j)) - d - \text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j)) \geq 0.$

Portanto a condição 1 também é satisfeita. ∎

Devido a este lema, podemos afirmar que conseguimos obter a lista CL_j a partir da lista CL_{j-1} fazendo apenas verificações de diferenças de out e de diferenças de $\text{hDif}_{\overline{G}}$.

Repare que se j_1 e j_2 são duas colunas de G tais que $j_1 \in CL_{j-1}$, $j_2 \in CL_{j-1}$, $j_1 < j_2$ e $\text{hDif}_{\overline{G}}((n-i, j_2), (n-i', j)) = \text{hDif}_{\overline{G}}((n-i, j_1), (n-i', j))$ então $\text{out}((i', j_2), (i, j)) < \text{out}((i', j_1), (i, j))$ e j_2 não é uma coluna tal que a condição 1 do Lema 4.39 é falsa.

Observação 4.40 *Para encontrar um j_2 que torne a condição 1 do Lema 4.39 falsa é necessário que $hDif_{\overline{G}}((n-i, j_2), (n-i', j)) \neq hDif_{\overline{G}}((n-i, j_1), (n-i', j))$. Logo, é necessário que haja pelo menos uma coluna j_3 tal que $j_3 \in BL_{\overline{G}}^{n-i, n-i', j}$ e $j_1 < j_3 \leq j_2$.*

A seguir iremos definir a função parcial Δout_G, que representa as diferenças $\text{out}((i', j_2), (i, j)) - \text{out}((i', j_1), (i, j))$ para cada dois elementos consecutivos j_1 e j_2 de CL_j. Se j_2 é o primeiro elemento da lista CL_j, então assumimos que existe um fictício elemento j_1 em CL_j tal que $\text{out}((i', j_1), (i, j)) = 0$ e $j_1 < j_2$.

Dizemos que o elemento j_1 é o *antecessor de j_2 em CL_j* se $j_1 < j_2$, $j_1 \in CL_j$ e não existe uma coluna j_3 de G tal que $j_1 < j_3 < j_2$ e $j_3 \in CL_j$.

Definição 4.41 (Função Δout_G) *Dado um grafo de edição estendido $G = (V, E, \omega)$ de s e t, a função parcial $\Delta out_G : V \times V \to \mathbb{R}$ é definida por*

1. *$\Delta out_G((i', j_2), (i, j)) = out((i', j_2), (i, j)) - out((i', j_1), (i, j))$, se $i' \leq i$, $j_2 \in CL_j$ e j_2 não é o primeiro elemento de CL_j, onde j_1 é o antecessor de j_2 em CL_j;*

2. *$\Delta out_G((i', j_1), (i, j)) = out((i', j_1), (i, j))$, se $i' \leq i$ e j_1 é o primeiro elemento de CL_j.*

Observação 4.42 *Se $j_1 \in CL_j$, $j_2 \in CL_j$ e $j_2 > j_1$, então*

$$out((i', j_2), (i, j)) - out((i', j_1), (i, j)) = \sum_{j_3 \in CL_j, j_1 < j_3 \leq j_2} \Delta out_G((i', j_3), (i, j)).$$

Sabemos que $\text{out}((i', j_2), (i, j)) - \text{out}((i', j_1), (i, j)) < 0$, para cada dois elementos consecutivos j_1 e j_2 de CL_j, logo, para todo j_2 tal que $j_2 \in CL_j$ e j_2 não é o primeiro elemento de CL_j temos que $\Delta\text{out}_G((i', j_2), (i, j))$ é negativo.

Repare que existem sempre $|CL_j|$ valores válidos de $\Delta\text{out}_G((i', j_2), (i, j))$ quando i', i e j são fixos.

De acordo com a Observação 4.29 e a definição de Δout_G, podemos fazer a observação a seguir.

Observação 4.43 *Sejam j, j_1 e j_2 tais que, $0 \leq j_1 < j_2 < j$, $j_1 \in CL_j$, $j_2 \in CL_j$ e j_1 é o antecessor de j_2 em CL_j. Temos que $\Delta out_G((i', j_2), (i, j)) = \Delta out_G((i', j_2), (i, j-1)) + hDif_{\overline{G}}((n-i, j_2), (n-i', j)) - hDif_{\overline{G}}((n-i, j_1), (n-i', j))$.*

Portanto, o valor de $\Delta\text{out}_G((i', j_2), (i, j))$ é diferente de $\Delta\text{out}_G((i', j_2), (i, j-1))$ quando existe pelo menos um elemento j_3 contido em $BL_G^{i',i,j}$ tal que $j_1 < j_3 \leq j_2$ e j_1 é o antecessor de j_2 na lista CL_j.

De acordo com a definição de Δout_G e com a Observação 4.42 podemos fazer a seguinte observação para o cálculo do primeiro elemento de Δout_G.

Observação 4.44 *Sejam j e j_2 tais que, $0 \leq j_2 < j$, $j_2 \in CL_j$ e j_2 é o primeiro elemento de CL_j. Temos que*
$\Delta out_G((i', j_2), (i, j)) = hDif_{\overline{G}}((n-i, j_2), (n-i', j)) +$

$$\sum_{j_1 | j_1 \in CL_{j-1}, j_1 \leq j_2} \Delta out_G((i', j_1), (i, j-1)).$$

Para o cálculo de $\Delta\text{out}_G((i', j), (i, j))$ usaremos a seguinte observação.

Observação 4.45 *Para todo j tal que $1 \leq j \leq m$ temos que $out((i', j), (i, j)) - out((i', j-1), (i, j)) = out((i', j), (i, j)) - hDif_{\overline{G}}((n-i, j-1), (n-i', j)) - out((i', j-1), (i, j-1))$.*

Algoritmo 8 Obtém o máximo de Out_j para cada j

obtémMaxOut(BL, OUT_G)

1 ▷ $OUT_G[j] = B[i', j] + \overline{\omega}((n-i, j), (n-i', j))$
2 ▷ $BL[j].\psi = \psi_{\overline{G}}(n-i, n-i', j)$
3 ▷ $BL[j].j'[\alpha] = BL_j^{[,\alpha,]}$
4 ▷ $BL[j].\text{hDif}_{\overline{G}}[\alpha] = \text{hDif}_{\overline{G}}((n-i, BL_j^{[,\alpha,]}), (n-i', j))$
5 ▷ Coloca o zero na lista de candidatos
6 $CL.add(0)$
7 $\Delta\text{out}_G[0] \leftarrow OUT_G[0]$
8 $maxOut[0] \leftarrow \Delta\text{out}_G[0]$
9 **para** j **de** 1 **até** m **faça**
10 $\alpha \leftarrow 0$
11 $\text{hDif}_{\overline{G}}Prev \leftarrow 0$
12 **enquanto** $\alpha < BL[j].\psi$ **faça**
13 ▷ Obtém o menor j_1 em CL tal que $j_1 \geq BL[j].j'[\alpha]$
14 $j_1 \leftarrow CL.find(BL[j].j'[\alpha])$
15 **enquanto** $\alpha < BL[j].\psi - 1$ e $CL.find(BL[j].j'[\alpha+1]) = j_1$ **faça**
16 $\alpha \leftarrow \alpha + 1$
17 ▷ $\Delta\text{out}_G[\text{j}_1] = \Delta\text{out}_G((i', j_1), (i, j))$
18 $\Delta\text{out}_G[j_1] \leftarrow \Delta\text{out}_G[j_1] + BL[j].\text{hDif}_{\overline{G}}[\alpha] - \text{hDif}_{\overline{G}}Prev$
19 **enquanto** $(\Delta\text{out}_G[j_1] \geq 0)$ e $(j_1 \neq CL.first())$ **faça**
20 $j_2 \leftarrow CL.previous(j_1)$
21 $CL.remove(j_2)$
22 $\Delta\text{out}_G[j_1] \leftarrow \Delta\text{out}_G[j_1] + \Delta\text{out}_G[j_2]$
23 $\text{hDif}_{\overline{G}}Prev \leftarrow BL[j].\text{hDif}_{\overline{G}}[\alpha]$
24 $\alpha \leftarrow \alpha + 1$
25 ▷ Insere j na lista de candidatos
26 $CL.add(j)$
27 ▷ Obtém $\Delta\text{out}_G[j]$
28 $\Delta\text{out}_G[j] \leftarrow OUT_G[j] - BL[j].\text{hDif}_{\overline{G}}[\alpha-1] - OUT_G[j-1]$
29 **enquanto** $(\Delta\text{out}_G[j] \geq 0)$ e $(j \neq CL.first())$ **faça**
30 $j_2 \leftarrow CL.previous(j)$
31 $CL.remove(j_2)$
32 $\Delta\text{out}_G[j] \leftarrow \Delta\text{out}_G[j] + \Delta\text{out}_G[j_2]$
33 $maxOut[j] \leftarrow \Delta\text{out}_G[CL.first()]$
34 **devolva** $maxOut$

O Algoritmo 8 obtém, para i e i' fixos tais que $0 \leq i \leq i' \leq n$, o máximo da lista Out_j para cada j tal que, $0 \leq j \leq m$ e funciona como descrito a seguir.

O Algoritmo 8 considera que as linhas i e i' do grafo de edição estendido G de s e t estão fixas.

O algoritmo constrói a lista de candidatos CL para cada j de 0 a m e mantém um vetor Δout_G de tal forma que, para um dado j, cada valor válido de $\Delta\text{out}_G((i', j_1), (i, j))$ está em $\Delta\text{out}_G[j_1]$. No algoritmo, o vetor $maxOut$ contém os valores máximos de Out_j e será devolvido pelo algoritmo.

Quando $j = 0$ a lista CL_j contém somente um elemento, que é o zero (linha 6). O valor de $\Delta\text{out}_G((i', 0), (i, 0))$ é $\text{out}((i', 0), (i, 0))$ que é igual a $B[i', 0] + \overline{\omega}((n - i, 0), (n - i', 0))$ (linha 7). Portanto, o máximo de Out_0 é $\text{out}((i', 0), (i, 0))$ (linha 8).

Durante a execução dos comandos das linhas 10 a 33 que estão dentro do laço da linha 9, a lista CL_j é obtida usando a lista CL_{j-1} e os valores válidos de $\Delta\text{out}_G((i', j_1), (i, j))$ são atualizados em Δout_G. Além disso, após a construção da lista CL_j e de atualizar Δout_G, o último comando dentro deste laço (linha 33) coloca em $maxOut$ o valor máximo de Out_j.

A seguir vamos dar uma breve explicação de como a lista CL_j é construída e como Δout_G é atualizado dentro dos comandos que estão dentro do laço da linha 9.

A lista CL_j é construída da seguinte forma:

- Na linha 26 do algoritmo é inserida a coluna j na lista CL_{j-1}, que de acordo com a observação 4.38 é a única coluna que deve ser inserida na lista CL_{j-1} para obter a lista CL_j.

- A linha 21 do algoritmo remove as colunas de CL_{j-1} que tornam a condição 1 do Lema 4.39 falsa.

- A linha 31 do algoritmo remove as colunas de CL_{j-1} que tornam a condição 2 do Lema 4.39 falsa.

Os valores de $\Delta\text{out}_G((i', j_1), (i, j))$ são alterados no vetor Δout_G, para cada $j_1 \in CL_j$, nas linhas 18, 22, 28 e 32, de acordo com as observações 4.42, 4.43, 4.44 e 4.45.

A lista de candidatos CL pode ser implementada como uma lista ligada, de tal forma que as operações de remoção de um elemento ($CL.remove()$), inclusão de um novo elemento ($CL.add()$) e obtenção do elemento anterior

a um elemento da lista ($CL.previous(j_1)$) podem ser executadas em tempo constante. Além disto, utilizando uma estrutura do tipo *disjoint sets* para armazenar os elementos da lista de candidatos, podemos realizar k operações de inclusão(*make-set*), de remoção (*union*) e de encontrar (*find*) na lista de candidatos em tempo $O(k)$, se utilizarmos o algoritmo proposto por [30][3]. Portanto, qualquer operação na lista de candidatos pode ser executada em tempo amortizado constante.

Portanto, qualquer linha do algoritmo pode ser executada em tempo constante.

A linha 9 é executada m vezes.

Para cada j de 1 a m o número de vezes que a linha 12 é executada mais o número de vezes que a linha e 15 é executada é igual ou menor a $2\psi_{\overline{G}}(n-i, n-i', j)+1$, ou seja, estas linhas são executadas $O(m)$ vezes.

O número de remoções dos candidatos da lista CL é igual ou menor a m. Portanto, o número de vezes que a linha 21 é executada mais o número de vezes que a linha 31 é executada é igual ou menor a m.

O espaço consumido pelo algoritmo é composto pela lista CL e pelos vetores Δout_G e $maxOut$, os quais consomem espaço $O(m)$, e por algumas variáveis com espaço constante.

Assim o Algoritmo 8 executa em tempo e espaço $O(m)$.

O Algoritmo 9 constrói a matriz B, descrita anteriormente, e funciona como descrito a seguir.

O laço controlado pela linha 4 calcula os valores da linha i de B. Os valores de $B[i,j]$ calculados nas linhas 5 a 20 não consideram que a última aresta seja uma aresta estendida. Estas linhas levam tempo $O(m)$ para serem executadas.

O laço controlado pela linha 24 calcula o valor de $B[i,j]$ considerando que a última aresta é uma aresta estendida. Os comandos controlados por este laço calculam, para cada i' e i, os valores máximos de $\text{out}((i',j'),(i,j))$ e atualiza $B[i,j]$ se necessário.

O algoritmo *constroiBL* para construir o vetor BL executado na linha 25 pode ser desenvolvido como descrito em [54] na seção 6 com tempo de execução de $O(m)$.

Os comandos controlados pelo laço da linha 34 calculam os valores de $\text{out}((i',j),(i,j))$ e executam em tempo $O(m)$ para i e i' fixos.

[3]Na implementação que realizamos do algoritmo, utilizamos a estrutura clássica descrita em [18] que tem tempo de execução $O(\alpha(k))$, sendo α a função inversa de Ackermann.

Algoritmo 9 Algoritmo $O(n^3)$ para obtenção da matriz B

BimN3(s, t)

```
1  ▷ G = (V, E, ω) é o grafo de edição estendido de s e t e
2  ▷ Ḡ = (V, Ē, ω̄) o grafo de edição de s̄ e t
3  ▷ B[i, j] = ω((0, 0), (i, j)) em G
4  para i de 0 até |s| faça
5          se i = 0 então
6                  B[0, 0] ← 0
7          senão
8                  ▷ Na 1ª coluna ∄ arestas horizontais nem diagonais
9                  B[i, 0] ← B[i − 1, 0] + ω(ε_V^(i,j))
10         para j de 1 até |t| faça
11                 se i = 0 então
12                         ▷ Na 1ª linha ∄ arestas verticais nem diagonais
13                         B[0, j] ← B[0, j − 1] + ω(ε_H^(i,j))
14                 senão
15                         ▷ Obtém o máximo entre os caminhos que termi-
16                         ▷ nam nas arestas horizontal, vertical e diagonal
17                         h ← B[i, j − 1] + ω(ε_H^(i,j))
18                         v ← B[i − 1, j] + ω(ε_V^(i,j))
19                         d ← B[i − 1, j − 1] + ω(ε_D^(i,j))
20                         B[i, j] ← max(h, v, d)
21         ▷ Obtém, para cada j da linha i, o máximo entre os
22         ▷ caminhos que terminam numa aresta estendida
23         BL ← ∅
24         para i′ de i descendo até 0 faça
25                 BL ← constroiBL(Ḡ, BL, i, i′)
26                 ▷ Obtém OUT_G[j] = B[i′, j] + ω̄((n − i, j), (n − i′, j))
27                 para j de 0 até |t| faça
28                         se i′ = i então
29                                 W[j] ← 0
30                         senão
31                                 W[j] ← W[j] + ω̄(ε̄_V^(n−i′,j))
32                         OUT_G[j] ← B[i′, j] + W[j]
33                 maxOut ← obtemMaxOut(BL, OUT_G)
34                 para j de 0 até |t| faça
35                         B[i, j] ← max(B[i, j], maxOut[j] + ω_inv)
36 devolva B
```

A linha 33 executa o Algoritmo 8, o qual já vimos que executa em tempo $O(m)$ para i e i' fixos.

Como os laços nas linhas 4 e 24 variam, respectivamente, as linhas i e i' temos que o Algoritmo 9 executa em tempo $O(n^2m)$.

Se quisermos considerar a constante $\psi_{\overline{G}}(n-i, n-i', j)$, que depende do sistema de pontuação, na análise da complexidade do algoritmo, teremos que o algoritmo tem tempo de execução $O(\psi n^2 m)$, onde ψ é o maior $\psi_{\overline{G}}(n-i, n-i', j)$, para $0 \leq i' \leq i \leq n$ e $0 \leq j \leq m$.

Testes realizados com dados reais para alinhamento de duas seqüências mostraram que o algoritmo $O(n^3)$ é, como seria esperado, mais rápido que os algoritmos $O(n^3 \log n)$ e $O(n^4)$ [21]. Para matrizes com alto grau de esparsidade o algoritmo esparso [21] é praticamente instantâneo e o mais rápido de todos.

Antes de desenvolvermos os algoritmos 9 ($O(n^3)$) e 7 ($O(n^3 \log n)$) os algoritmos com as melhores complexidades no tempo de execução eram algoritmos que executavam em tempo $O(n^4)$ [20, 31]. Portanto, com os nossos algoritmos conseguimos melhorar a complexidade de tempo de execução para solucionar o problema de obtenção de um alinhamento ótimo com inversões não sobrepostas.

Capítulo 5

Alinhamento com duplicações

5.1 Introdução

Neste capítulo, consideraremos que s e t são duas seqüências de comprimentos n e m, respectivamente.

Os eventos biológicos típicos que são considerados normalmente nos procedimentos de alinhamentos atuais de seqüências de DNA são: substituição, remoção e inserção de nucleotídeos. Um outro evento biológico que ocorre, mas que normalmente não é considerado nos alinhamentos usuais, é a duplicação, a qual iremos considerar sob algumas restrições nos algoritmos de alinhamentos deste capítulo. São dois os tipos de duplicações que consideraremos que ocorrem: duplicações encadeadas (em *tandem*) ou transposições.

Sejam duas seqüências tais que somente uma delas sofreu um evento de duplicação. Ao analisarmos estas seqüências através de um alinhamento usual, é muito provável que neste alinhamento haja uma seqüência de colunas que correspondam a inserções (ou remoções) no trecho que corresponde a duplicação. Neste caso, pretendemos mostrar, num alinhamento com duplicações, que houve uma duplicação em uma das seqüências, ao invés de mostrar a seqüência de colunas que correspondem a inserções (ou remoções) como é mostrado num alinhamento comum.

Em 1997, Benson [11] propôs um modelo para o alinhamento de seqüências que considera a presença de repetições em *tandem* de mesmo tamanho nas seqüências. Ele propôs dois algoritmos exatos para obter um tal alinhamento ótimo que considera estas repetições. O primeiro algoritmo proposto executa em tempo $O(n^5)$ e espaço $O(n^2)$. O segundo algoritmo proposto

executa em tempo $O(n^4)$ e espaço $O(n^3)$.

Iremos propor modelos parecidos com este modelo apresentado por Benson e algoritmos exatos que executam em tempo $O(n^3)$ e memória $O(n^2)$. Utilizando as técnicas utilizadas no algoritmo $O(n^3)$ para a obtenção de um alinhamento ótimo com inversões não sobrepostas, existe um algoritmo que, em tempo $O(n^3)$ e memória $O(n^2)$, obtém um alinhamento ótimo com duplicações em *tandem* segundo o mesmo modelo proposto pode Benson. Porém acreditamos que os modelos que iremos propor, são modelos mais gerais e que consideram mais casos de duplicações.

Em seqüências biológicas é comum, com o decorrer do tempo, ocorrerem eventos que alteram estas seqüências. Estas alterações podem ocorrer inclusive em trechos das seqüências que participaram de uma duplicação. Sendo assim, ao analisarmos duplicações em seqüências biológicas devemos considerar a possibilidade da seqüência original e da repetição terem sofrido mutações e não serem mais iguais.

Definição 5.1 (Intervalo de repetição em A) *Seja A uma matriz de 3 linhas e $r+1$ colunas tal que, $A[2,k] \in \{+,!,\#\}$ para todo k tal que, $0 \leq k \leq r$. Dizemos que o intervalo $[k_1,k_2]$ de colunas de A é um intervalo de repetição em A, ou mais especificamente é o intervalo de repetição $[k_1,k_2]$ de A, se k_1 e k_2 são tais que $0 \leq k_1 \leq k_2 \leq r$, $A[2,k_1] =!$, $A[2,k'] = \#$, para todo k' tal que $k_1 < k' \leq k_2$, e não existe uma coluna k_2+1 em A tal que $A[2,k_2+1] = \#$.*

Este conceito de intervalo de repetição será utilizado em um alinhamento com duplicações para associarmos um trecho do alinhamento a uma duplicação.

Definição 5.2 (Função exclude) *A função exclude $: \Sigma_1^* \times \Sigma_1^* \rightarrow \Sigma_1^*$, onde Σ_1^* é o conjunto com todas as possíveis seqüências de símbolos de um alfabeto Σ_1 qualquer, é definida por exclude$(w,X) = w'$, onde $w' = w[i_1]w[i_2]\ldots w[i_r]$, $w'[i'] \notin X$ para todo índice i' de w' e $w[i] \in X$ para todo índice i de w tal que $i \notin (i_1,i_2,\ldots,i_r)$.*

Ou seja, exclude(w,X) gera uma nova seqüência, excluindo da seqüência w todos e somente os símbolos de w que estão em X.

Seja o conjunto $\Upsilon = \Upsilon_s \cup \Upsilon_t$, onde $\Upsilon_s = \{(s,x) \mid x \in [1,n]\}$ e $\Upsilon_t = \{(t,y) \mid y \in [1,m]\}$.

Definição 5.3 (Alinhamento com duplicações de s e t) *Um alinhamento com duplicações de s e t é uma matriz A de 3 linhas e r colunas, tal que:*

- $r \geq \max(m, n)$ *e* $r \leq m + n$*;*
- *para toda coluna k de A temos que:*
  - $A[0, k] \in [1, n] \cup \{-\} \cup \Upsilon$*,*
  - $A[1, k] \in [1, m] \cup \{-\} \cup \Upsilon$*,*
  - $A[2, k] \in \{+, !, \#\}$*,*
  - *se* $A[0, k] = A[1, k]$ *então* $A[0, k] \in [1, n]$*,*
  - *se* $A[0, k] \in \Upsilon$ *então* $A[1, k] \notin \Upsilon$*,*
  - *se* $A[0, k] \in \Upsilon$ *ou* $A[1, k] \in \Upsilon$ *então* $A[2, k] \in \{!, \#\}$*;*
  - *se* $A[0, k] \in [1, n]$ *e* $A[1, k] \in [1, m]$ *então* $A[2, k] = +$*;*
  - *se* $A[2, k] = \#$ *então* $A[2, k - 1] \in \{!, \#\}$*;*
- *exclude*$(A[0, 0\,..\,r - 1]\Upsilon \cup \{-\}) = (1, 2, \ldots, n)$*;*
- *exclude*$(A[1, 0\,..\,r - 1], \Upsilon \cup \{-\}) = (1, 2, \ldots, m)$*;*
- *Para todo intervalo de repetição $[k_1, k_2]$ em A temos que:*
  1. *se* $A[0, k_1] \in [1, n] \cup \{-\}$ *e* $A[1, k_1] \in \Upsilon \cup \{-\}$ *então exclude*$(A[0, k_1\,..\,k_2], \{-\}) = (i_1, i_1+1, \ldots, i_1+x_1)$*, onde* $1 \leq i_1 \leq i_1+x_1 \leq n$*, e exclude*$(A[1,\ k_1\,..\,k_2], \{-\}) = X$*, onde X é uma das seguintes seqüências:*
     (a) $((s, i_2), (s, i_2+1), \ldots, (s, i_2+x_2))$*, onde* $1 \leq i_2 \leq i_2+x_2+1 \leq i_1$ *ou* $i_1 + x_1 + 1 \leq i_2 \leq i_2 + x_2 + 1 \leq n + 1$*,*
     (b) $((s, i_3), (s, i_3+1), \ldots, (s, i_1-1), (s, i_1+x_1+1), \ldots, (s, i_1+x_1+x_3))$*, onde* $1 \leq i_3 \leq i_1$ *e* $i_1+x_1+1 \leq i_1+x_1+x_3+1 \leq n+1$*,*
     (c) $((t, j), (t, j+1), \ldots, (t, j+x_4))$*, onde* $1 \leq j \leq j+x_4+1 \leq m+1$*;*
  2. *se* $A[1, k_1] \in [1, m] \cup \{-\}$ *e* $A[0, k_1] \in \Upsilon \cup \{-\}$ *então exclude*$(A[1, k_1\,..\,k_2], \{-\}) = (j_1, j_1+1, \ldots, j_1+y_1)$*, onde* $1 \leq j_1 \leq j_1+y_1 \leq m$*, e exclude*$(A[0, k_1\,..\,k_2], \{-\}) = Y$*, onde Y é uma das seguintes seqüências:*
     (a) $((t, j_2), (t, j_2+1), \ldots, (t, j_2+y_2))$*, onde* $1 \leq j_2 \leq j_2+y_2+1 \leq j_1$ *ou* $j_1 + y_1 + 1 \leq j_2 \leq j_2 + y_2 + 1 \leq m + 1$*,*

(b) $((t, j_3), (t, j_3+1), \ldots, (t, j_1-1), (t, j_1+y_1+1), \ldots, (t, j_1+y_1+y_3))$, *onde* $1 \leq j_3 \leq j_1$ *e* $j_1+y_1+1 \leq j_1+y_1+y_3+1 \leq m+1$,

(c) $((s, i), (s, i+1), \ldots, (s, i+y_4))$, *onde* $1 \leq i \leq i+y_4+1 \leq n+1$.

Os Exemplos 5.5, 5.6 e 5.7 mostram alinhamentos com duplicações.

O número de colunas de um alinhamento com duplicações A é o comprimento de A.

Seja Λ_D o conjunto de todos os possíveis alinhamentos com duplicações de duas seqüências.

Definição 5.4 (Função parcial simb) *Seja a função parcial* $simb : \Lambda_D \times \{0,1\} \times \mathbb{N} \to \Sigma \cup \{-\}$ *definida por:*

- $simb(A, 0, k) = s[A[0, k]]$ *se* $A[0, k] \in [1, n]$ *e* k *é o índice de uma coluna de* A*, onde* A *é um alinhamento com duplicações de* s *e* t*;*
- $simb(A, 1, k) = t[A[1, k]]$ *se* $A[1, k] \in [1, m]$ *e* k *é o índice de uma coluna de* A*, onde* A *é um alinhamento com duplicações de* s *e* t*;*
- *para* $x \in \{0, 1\}$, $simb(A, x, k) = s[i]$ *se* $A[x, k] = (s, i)$ *e* k *é o índice de uma coluna de* A*, onde* A *é um alinhamento com duplicações de* s *e* t*;*
- *para* $x \in \{0, 1\}$, $simb(A, x, k) = t[j]$ *se* $A[x, k] = (t, j)$ *e* k *é o índice de uma coluna de* A*, onde* A *é um alinhamento com duplicações de* s *e* t*;*
- *para* $x \in \{0, 1\}$, $simb(A, x, k) = -$ *se* $A[x, k] = -$ *e* k *é o índice de uma coluna de* A*;*

Para todo intervalo de repetição $[k_1, k_2]$ de um alinhamento com duplicações A de s e t temos que,

- se $A[0, k_1] \in [1, n] \cup \{-\}$ e $A[1, k_1] \in \Upsilon \cup \{-\}$ então dizemos que
  - o intervalo de repetição $[k_1, k_2]$ de A é um *intervalo de repetição por excisão*,
  - $\text{exclude}(\text{simb}(A, 0, k_1)\text{simb}(A, 0, k_1+1) \ldots \text{simb}(A, 0, k_2), \{-\})$ é a *repetição do intervalo de repetição* $[k_1, k_2]$ *de* A,
  - $\text{exclude}(\text{simb}(A, 1, k_1)\text{simb}(A, 1, k_1+1) \ldots \text{simb}(A, 1, k_2), \{-\})$ é a *seqüência base do intervalo de repetição* $[k_1, k_2]$ *de* A;

- se $A[1,k] \in [1,m] \cup \{-\}$ e $A[0,k] \in \Upsilon \cup \{-\}$ então dizemos que
  - o intervalo de repetição $[k_1, k_2]$ de A é um *intervalo de repetição por duplicação*,
  - $\text{exclude}(\text{simb}(A,1,k_1)\text{simb}(A,1,k_1+1)\ldots\text{simb}(A,1,k_2),\{-\})$ é a *repetição do intervalo de repetição* $[k_1,k_2]$ *de* A,
  - $\text{exclude}(\text{simb}(A,0,k_1)\text{simb}(A,0,k_1+1)\ldots\text{simb}(A,0,k_2),\{-\})$ é a *seqüência base do intervalo de repetição* $[k_1,k_2]$ *de* A;

O Exemplo 5.5 mostra um alinhamento com duplicações de $s = s[1..7]$ e $t = t[1..5]$. O intervalo de repetição $[1,4]$ de A é um intervalo de repetição por excisão e está de acordo com o caso 1a de um intervalo de repetição na Definição 5.3. A seqüência $s[2\,..\,4]$ é a repetição e $s[5\,..\,7]$ é a seqüência base do intervalo de repetição $[1,4]$ de A.

Exemplo 5.5 (Alinhamento com duplicações caso 1a)

$$A = \begin{bmatrix} 1 & 2 & 3 & - & 4 & 5 & 6 & - & 7 \\ 1 & (s,5) & - & (s,6) & (s,7) & 2 & 3 & 4 & 5 \\ + & ! & \# & \# & \# & + & + & + & + \end{bmatrix}$$

No alinhamento com duplicações do Exemplo 5.6 o intervalo de repetição $[2,5]$ de A é um intervalo de repetição por excisão e está de acordo com o caso 1b de um intervalo de repetição na Definição 5.3. A seqüência $s[2\,..\,4]$ é a repetição e $s[1]s[5\,..\,6]$ é a seqüência base do intervalo de repetição $[2,5]$ de A.

Exemplo 5.6 (Alinhamento com duplicações caso 1b)

$$A = \begin{bmatrix} - & 1 & 2 & 3 & - & 4 & 5 & 6 & - & 7 \\ 1 & 2 & (s,1) & - & (s,5) & (s,6) & 3 & 4 & 5 & 6 \\ + & + & ! & \# & \# & \# & + & + & + & + \end{bmatrix}$$

No alinhamento com duplicações do Exemplo 5.7 o intervalo de repetição $[5,8]$ de A é um intervalo de repetição por excisão e está de acordo com o caso 1c de um intervalo de repetição na Definição 5.3. A seqüência $s[4\,..\,6]$ é a repetição e $t[3\,..\,5]$ é a seqüência base do intervalo de repetição $[5,8]$ de A.

Exemplo 5.7 (Alinhamento com duplicações caso 1c)

$$A = \begin{bmatrix} - & 1 & 2 & 3 & - & 4 & 5 & 6 & - & 7 \\ 1 & 2 & 3 & 4 & 5 & (t,3) & - & (t,4) & (t,5) & 6 \\ + & + & + & + & + & ! & \# & \# & \# & + \end{bmatrix}$$

Seja A um alinhamento com duplicações de s e t e o intervalo de repetição $[k_1, k_2]$ em A. Se o intervalo de repetição $[k_1, k_2]$ em A é um intervalo de repetição por excisão, então as colunas $(k_1, k_1+1, \ldots, k_2)$ do alinhamento estão mostrando a ocorrência de uma duplicação em s que não ocorreu em t ou a ocorrência de uma excisão em t que não ocorreu em s. Equivalentemente, se o intervalo de repetição $[k_1, k_2]$ em A é um intervalo de repetição por duplicação, então as colunas $(k_1 \,..\, k_2)$ do alinhamento estão mostrando a ocorrência de uma excisão em s que não ocorreu em t ou a ocorrência de uma duplicação em t que não ocorreu em s.

Associaremos a cada coluna k de um alinhamento com duplicações A uma ou duas operações de edição, similarmente como é feito num alinhamento com inversões não sobrepostas através da função opi, da seguinte forma:

- se $A[0,k] = -$ então associamos a coluna k de A uma operação de edição de inserção de $\text{simb}(A, 1, k)$;

- se $A[1,k] = -$ então associamos a coluna k de A uma operação de edição de remoção de $\text{simb}(A, 0, k)$;

- se $A[0,k] \neq -$ e $A[1,k] \neq -$ então associamos a coluna k de A uma operação de edição de substituição de $\text{simb}(A, 0, k)$ por $\text{simb}(A, 1, k)$;

- se a coluna k é tal que existe um intervalo de repetição por duplicação $[k, k']$ em A, então também associamos a coluna k de A uma operação de edição de duplicação tal que, a seqüência original da operação de edição de duplicação é igual a seqüência base do intervalo de repetição $[k, k']$ de A;

- se a coluna k é tal que existe um intervalo de repetição por excisão $[k, k']$ em A, então também associamos a coluna k de A uma operação de edição de excisão tal que, a seqüência original da operação de edição de excisão é igual a seqüência base do intervalo de repetição $[k, k']$ de A;

Repare que sempre associamos a uma coluna k de um alinhamento com duplicações uma operação de edição pontual e, na primeira coluna de um intervalo de repetição por duplicação, além desta operação de edição pontual, também associamos uma operação de edição de duplicação, assim como na primeira coluna de um intervalo de repetição por excisão, além dessa operação de edição pontual, também associamos uma operação de edição de excisão.

Como um alinhamento com duplicações é uma seqüência de colunas, associamos a um alinhamento com duplicações uma seqüência de operações de edição.

Definição 5.8 (Transformação com duplicações de s e t) *Dadas duas seqüências s e t, dizemos que uma transformação com duplicações de s e t é qualquer seqüência de operações de edição $\Psi = (\gamma_0, \gamma_1, \ldots, \gamma_r)$ tal que $\phi_\Psi(s) = t$ e γ_k é uma operação de edição de inserção, remoção, substituição, duplicação ou excisão, para todo k tal que $0 \leq k \leq r$.*

Definição 5.9 (Grafo de duplicações de A) *Dado um alinhamento com duplicações A de s e t, o grafo de duplicações de A é um grafo dirigido $G_A = (V, E)$ tal que, cada vértice de V tem um rótulo distinto em Υ e $(u, v) \in E$ se uma das seguintes condições é satisfeita:*

- *$u = (s, i)$ e existe uma coluna k em A tal que $A[0, k] = i$ e $A[1, k] = v$;*
- *$u = (t, j)$ e existe uma coluna k em A tal que $A[1, k] = j$ e $A[0, k] = v$;*

Diferente de um alinhamento com inversões não sobrepostas, a seqüência de operações de edição associada a um alinhamento com duplicações de s e t, nem sempre é uma transformação com duplicações de s e t. Mais especificamente, a seqüência de operações de edição associada a um alinhamento com duplicações A de s e t, tal que o grafo de duplicações de A tem ciclos, não é uma transformação com duplicações de s e t.

Repare que cada aresta do grafo de duplicações G de A corresponde a uma coluna de A cuja operação de edição pontual associada é uma substituição. Se alterarmos uma das colunas de A, correspondente a uma aresta e de um ciclo de G, para uma coluna com *gap*, ou seja, cuja operação de edição pontual associada é uma inserção ou uma remoção, então a aresta e é removida de G e este ciclo deixa de existir.

Se o grafo de duplicações de A é acíclico, então podemos, com algumas (ou nenhuma) alterações na seqüência de operações de edição Ψ associada a

A, obter uma transformação com duplicações de s e t. As possíveis alterações em Ψ podem ser:

- alterar a ordem das operações de edição de Ψ
- dividir um intervalo de repetição em dois ou mais intervalos de repetições, ou seja, associar mais de uma operação de edição de duplicação (ou excisão) a um intervalo de repetição.

Em suma, com possivelmente algumas alterações num alinhamento com duplicações, podemos obter, a partir deste alinhamento com duplicações, uma transformação com duplicações.

Definição 5.10 (Pontuação de um alinhamento com duplicações) *A pontuação de um alinhamento com duplicações A é a soma das pontuações das colunas deste alinhamento e será denotado por ω^A.*

Dizemos que um alinhamento com duplicações de s e t é ótimo se ele tem pontuação máxima dentre todos os possíveis alinhamentos com duplicações de s e t.

Em alinhamentos que levam em conta somente as substituições, as remoções e as inserções, é comum que para a determinação da pontuação de uma coluna, considere-se somente a operação de edição pontual associada a esta coluna. Porém, num alinhamento com duplicações, além da operação de edição pontual associada a uma coluna, vamos considerar também se a coluna está ou não em um intervalo de repetição para a determinação da pontuação desta coluna. Além disso, para a primeira coluna de um intervalo de repetição, vamos considerar na pontuação da coluna, além da operação de edição pontual, a operação de edição de duplicação ou excisão associada a esta coluna e a repetição do intervalo de repetição.

Sejam $\omega_v(s[i])$, $\omega_h(t[j])$ e $\omega_d(s[i], t[j])$ as pontuações das operações de edição de remoção do símbolo $s[i]$, inserção do símbolo $t[j]$ e substituição do símbolo $s[i]$ pelo símbolo $t[j]$, respectivamente.

Considerando que um intervalo de repetição em um alinhamento com duplicações representa uma duplicação (ou uma excisão), e que o tamanho desta duplicação (ou excisão) é igual ao comprimento da repetição do intervalo de repetição então, propomos um sistema de pontuação que considera que as probabilidades da ocorrência de duplicações e excisões de tamanhos diferentes podem ser diferentes, ou seja, a pontuação para as operações de edição de

duplicação e excisão podem considerar o tamanho da duplicação ou excisão, aliás não só o tamanho como a própria seqüência da repetição. Para tanto, consideramos que a pontuação da operação de edição de duplicação ou de excisão associada a um intervalo de repetição é $\omega_x(s[i_1 \,..\, i_2])$ ou $\omega_u(t[j_1 \,..\, j_2])$, respectivamente, onde $s[i_1 \,..\, i_2]$ ou $t[j_1 \,..\, j_2]$ é a repetição deste intervalo de repetição.

Portanto a pontuação de uma coluna k de um alinhamento com duplicações A é igual a:

- $\omega_v(s[i])+z$ se a coluna k está associada a operação de edição de remoção do símbolo $s[i]$,
- $\omega_h(t[j])+z$ se a coluna k está associada a operação de edição de inserção do símbolo $t[j]$,
- $\omega_d(s[i], t[j]) + z$ se a coluna k está associada a operação de edição de substituição do símbolo $s[i]$ pelo símbolo $t[j]$.

onde z é igual a:

- $\omega_x(s[i_1 \,..\, i_2])$, se existe um intervalo de repetição por excisão $[k, k']$ em A, onde $s[i_1 \,..\, i_2]$ é a repetição deste intervalo de repetição,
- $\omega_u(t[j_1 \,..\, j_2])$, se existe um intervalo de repetição por duplicação $[k, k']$ em A, onde $t[j_1 \,..\, j_2]$ é a repetição deste intervalo de repetição,
- 0 (zero), se não existe um intervalo de repetição $[k, k']$ em A.

Observe que os valores $\omega_x(s[i_1 \,..\, i_2])$ e $\omega_u(t[j_1 \,..\, j_2])$ são utilizados somente na primeira coluna de um intervalo de repetição.

Assim como muitas vezes ocorre com outros eventos, simplesmente podemos adotar uma penalidade constante para uma duplicação/excisão.

Dados índices i_1, i_2, j_1 e j_2 tais que $0 < i_1 \leq i_2 + 1 \leq n + 1$, $0 < j_1 \leq j_2 + 1 \leq m + 1$, vamos denotar a pontuação de um alinhamento ótimo considerando duplicações de $s[i_1 \,..\, i_2] \times t[j_1 \,..\, j_2]$ por $\omega_D(s[i_1 \,..\, i_2], t[j_1 \,..\, j_2])$.

Dados i e j tais que $0 \leq i \leq n$ e $0 \leq j \leq m$, definimos a matriz M por $M[i, j] = \omega_D(s[1 \,..\, i], t[1 \,..\, j])$.

Definimos $M_t[i, j]$ como a pontuação máxima de um alinhamento com duplicações A de $s[1 \,..\, i] \times t[1 \,..\, j]$ tal que, a última coluna de A pertence a um intervalo de repetição por duplicação, $\forall i, j \mid 0 \leq i \leq n$ e $1 \leq j \leq m$. De forma

análoga definimos $M_s[i,j]$ como a pontuação máxima de um alinhamento com duplicações A de $s[1\,..\,i] \times t[1\,..\,j]$ tal que, a última coluna de A pertence a um intervalo de repetição por excisão, $\forall i,j \mid 1 \leq i \leq n$ e $0 \leq j \leq m$.

De acordo com as possibilidades da última coluna de um alinhamento com duplicações, podemos obter os valores de $M[i,j]$ da seguinte forma:

$$\begin{aligned}
&M[0,0] = 0 \\
&M[i,0] = \max\left\{\begin{array}{l} M[i-1,0] + \omega_v(s[i]), \\ M_s[i,0] \end{array}\right\},\ i > 0 \\
&M[0,j] = \max\left\{\begin{array}{l} M[0,j-1] + \omega_h(t[j]), \\ M_t[0,j] \end{array}\right\},\ j > 0 \\
&M[i,j] = \max\left\{\begin{array}{l} M[i,j-1] + \omega_h(t[j]), \\ M[i-1,j] + \omega_v(s[i]), \\ M[i-1,j-1] + \omega_d(s[i],t[j]), \\ M_s[i,j], \\ M_t[i,j] \end{array}\right\},\ i > 0 \text{ e } j > 0
\end{aligned} \tag{5.1}$$

Utilizando o sistema de pontuação de *gap* linear, descrito na página 50 da seção 3.1, tal que para qualquer coluna k de um alinhamento A temos que $\varphi(A[0,k], A[1,k])$ é igual a:

- $\omega_v(s[i])$ se $A[0,k] = s[i]$ e $A[1,k] = -$,
- $\omega_h(t[j])$ se $A[0,k] = -$ e $A[1,k] = t[j]$,
- $\omega_d(s[i],t[j])$ se $A[0,k] = s[i]$ e $A[1,k] = t[j]$,

diremos que $\omega_L(s',t')$ é a pontuação do alinhamento ótimo sem duplicações de s' com qualquer fator de t' e $\omega^*(s',t')$ é a pontuação do alinhamento global ótimo sem duplicações de $s' \times t'$.

Utilizaremos este sistema de pontuação para obter a pontuação de um alinhamento ótimo entre a seqüência base e a repetição de um intervalo de repetição num alinhamento com duplicações.

Definição 5.11 (Função parcial dup) *Dadas as seqüências s e t, definimos a função parcial $dup : \{s,t\} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ da seguinte forma:*

- $dup(t,j',j) = \omega_u(t[j'\,..\,j]) + \max(\omega_L(t[j'\,..\,j],s), \omega_L(t[j'\,..\,j], t[1\,..\,j'-1]t[j+1\,..\,m]))$, *se* $1 \leq j' \leq j \leq m$,
- $dup(s,i',i) = \omega_x(s[i'\,..\,i]) + \max(\omega_L(s[i'\,..\,i],t), \omega_L(s[i'\,..\,i], s[1\,..\,i'-1]s[i+1\,..\,n]))$, *se* $1 \leq i' \leq i \leq n$.

Ou seja, o valor de $\mathrm{dup}(t, j', j)$ é o valor máximo dentre as pontuações associadas aos possíveis intervalo de repetição por duplicação $[k_1, k_2]$ de um alinhamento com duplicações A de s e t tal que, $\mathrm{exclude}(A[1, k_1 \,..\, k_2], \{-\}) = (j', j'+1, \ldots, j)$. De modo similar, o valor de $\mathrm{dup}(s, i', i)$ é o valor máximo dentre as pontuações associadas aos possíveis intervalo de repetição por excisão $[k_1, k_2]$ de um alinhamento com duplicações A de s e t tal que, $\mathrm{exclude}(A[0, k_1 \,..\, k_2], \{-\}) = (i', i'+1, \ldots, i)$.

Portanto, temos que

$$M_t[i, j] = \max_{\forall j' | 1 \leq j' \leq j} (M[i, j'-1] + \mathrm{dup}(t, j', j))$$

Analogamente, temos que

$$M_s[i, j] = \max_{\forall i' | 1 \leq i' \leq i} (M[i'-1, j] + \mathrm{dup}(s, i', i))$$

Seja $G = (V, E, \omega)$ um grafo de edição estendido de s e t, onde ω é tal que:

- $\omega(\epsilon_H^{(i,j)}) = \omega_h(t[j])$,
- $\omega(\epsilon_D^{(i,j)}) = \omega_d(s[i], t[j])$,
- $\omega(\epsilon_V^{(i,j)}) = \omega_v(s[i])$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = \mathrm{dup}(s, i'+1, i)$, se $i' \neq i$ e $j' = j$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = \mathrm{dup}(t, j'+1, j)$, se $i' = i$ e $j' \neq j$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = -\infty$ se $i' \neq i$ e $j' \neq j$.

Pode-se verificar que o peso de um caminho ótimo em G é igual a pontuação de um alinhamento ótimo com duplicações de s e t.

Dado que, existem $O(n^3)$ arestas estendidas $\epsilon_{(i',j')}^{(i,j)}$ tal que $i' \neq i$ ou $j' \neq j$, e que a obtenção do valor $\mathrm{dup}(s, i', i)$ assim como do valor $\mathrm{dup}(t, j', j)$ levam tempo $O(n^2)$, temos que, dadas as seqüências s e t, um algoritmo trivial, que obtém o peso de um caminho ótimo em G, executa em tempo $O(n^5)$.

A seguir descreveremos um algoritmo que, dadas as seqüências s e t, obtém o peso de um caminho ótimo em G em tempo $O(n^3)$ e utilizando $O(n^2)$ de espaço.

5.2 Algoritmo para alinhamento com duplicações

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ definimos os conjuntos de fatores das seqüências s e t, $A_t^{j',j}$, $B_t^{j',j}$, $C_t^{j',j}$ e D_s da seguinte forma:

1. $A_t^{j',j} = \{t[j_1 \,..\, j_2] \mid 0 < j_1 \leq j_2 + 1 \leq j'\}$,
2. $B_t^{j',j} = \{t[j_3 \,..\, j_4] \mid j < j_3 \leq j_4 + 1 \leq |t| + 1\}$,
3. $C_t^{j',j} = \{t[j_5 \,..\, j' - 1]t[j + 1 \,..\, j_6] \mid 0 < j_5 \leq j' \text{ e } j \leq j_6 \leq |t|\}$ e
4. $D_s = \{s[i_1 \,..\, i_2] \mid 1 \leq i_1 \leq i_2 + 1 \leq |s| + 1\}$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ definimos, para cada conjunto $A_t^{j',j}$, $B_t^{j',j}$, $C_t^{j',j}$ e D_s, a pontuação máxima do alinhamento ótimo sem duplicações de $t[j' \,..\, j]$ com um elemento do conjunto da seguinte forma:

$$\begin{aligned}
\omega_t^A(j', j) &= \max(\omega^*(t[j' \,..\, j], x) \mid x \in A_t^{j',j}) \\
&= \max_{\forall j_1, j_2 | 0 < j_1 \leq j_2 + 1 \leq j'} (\omega^*(t[j' \,..\, j], t[j_1 \,..\, j_2]))\,,
\end{aligned}$$

$$\begin{aligned}
\omega_t^B(j', j) &= \max(\omega^*(t[j' \,..\, j], x) \mid x \in B_t^{j',j}) \\
&= \max_{\forall j_3, j_4 | j < j_3 \leq j_4 + 1 \leq m+1} (\omega^*(t[j' \,..\, j], t[j_3 \,..\, j_4]))\,,
\end{aligned}$$

$$\begin{aligned}
\omega_t^C(j', j) &= \max(\omega^*(t[j' \,..\, j], x) \mid x \in C_t^{j',j}) \\
&= \max_{\forall j_5, j_6 | 0 < j_5 \leq j' \text{ e } j \leq j_6 \leq m} (\omega^*(t[j' \,..\, j], t[j_5 \,..\, j' - 1]t[j + 1 \,..\, j_6]))\ \text{ e}
\end{aligned}$$

$$\begin{aligned}
\omega_{t|s}^D(j', j) &= \max(\omega^*(t[j' \,..\, j], x) \mid x \in D_s) \\
&= \max_{\forall i_1, i_2 | 1 \leq i_1 \leq i_2 + 1 \leq n+1} (\omega^*(t[j' \,..\, j], s[i_1 \,..\, i_2]))\,.
\end{aligned}$$

Portanto, para todos j' e j tais que $1 \leq j' \leq j \leq m$, temos que

$$\text{dup}(t, j', j) = \max \begin{pmatrix} \omega_t^A(j', j) \\ \omega_t^B(j', j) \\ \omega_t^C(j', j) \\ \omega_{t|s}^D(j', j) \end{pmatrix} + \omega_u(t[j' \,..\, j]).$$

A seguir, analisaremos separadamente cada um destes valores que podem compor a função $\mathrm{dup}(t, j', j)$: $\omega_t^A(j', j)$, $\omega_t^B(j', j)$, $\omega_t^C(j', j)$ e $\omega_{t|s}^D(j', j)$; e vamos propor algoritmos para construir matrizes com os valores pré-computados destas funções.

5.2.1 Função $\omega_t^A(j', j)$

Seja A um alinhamento ótimo de $t[j'+1\,..\,j] \times t[j_1+1\,..\,j_2]$, tal que não existe outro alinhamento de $t[j'+1\,..\,j] \times t[j_1'+1\,..\,j_2]$ com pontuação maior que a pontuação de A, onde j', j, j_1, j_2 e j_1' são tais que $0 \leq j_1 \leq j_2 \leq j' \leq j \leq m$ e $0 \leq j_1' \leq j_2$. Diremos que este é um *alinhamento ótimo para j_2 em $A_t^{j'+1,j}$*. Se o comprimento de A é igual a zero, o que acontece somente se $j_1 = j_2$ e $j' = j$, então a pontuação de A também é zero. Se o comprimento de A é $r+1$ e é maior que zero então a última coluna de A é de uma das seguintes formas:

- $A[0, r] = t[j]$ e $A[1, r] = -$. Neste caso, o alinhamento formado pelas primeiras $r-1$ linhas de A é um alinhamento ótimo para j_2 em $A_t^{j'+1,j-1}$ e portanto a pontuação de A é igual a pontuação de um alinhamento ótimo para j_2 em $A_t^{j'+1,j-1}$ mais o valor $\omega_v(t[j])$.

- $A[1, r] = t[j_2]$ e $A[0, r] = -$. Neste caso, o alinhamento formado pelas primeiras $r-1$ linhas de A é um alinhamento ótimo para j_2-1 em $A_t^{j'+1,j}$ e portanto a pontuação de A é igual a pontuação de um alinhamento ótimo para $j_2 - 1$ em $A_t^{j'+1,j}$ mais o valor $\omega_h(t[j_2])$.

- $A[0, r] = t[j]$ e $A[1, r] = t[j_2]$. Neste caso, o alinhamento formado pelas primeiras $r-1$ linhas de A é um alinhamento ótimo para $j_2 - 1$ em $A_t^{j'+1,j-1}$ e portanto a pontuação de A é igual a pontuação de um alinhamento ótimo para $j_2 - 1$ em $A_t^{j'+1,j-1}$ mais o valor $\omega_d(t[j], t[j_2])$.

A Figura 5.1 ilustra os caminhos num grafo de edição para a obtenção da pontuação de um alinhamento ótimo para j_2 em $A_t^{j'+1,j}$.

Com isto, vamos definir a matriz W_t^A tal que, $W_t^A[j_2, j', j]$ contém a pontuação de um alinhamento ótimo para j_2 em $A_t^{j'+1,j}$.

Seja W_t^A a matriz $(m+1) \times (m+1) \times (m+1)$ definida por

- $W_t^A[0, j', j] = 0$, se $0 \leq j' = j \leq m$,

- $W_t^A[0, j', j] = W_t^A[0, j', j-1] + \omega_v(t[j])$, se $0 \leq j' < j \leq m$,

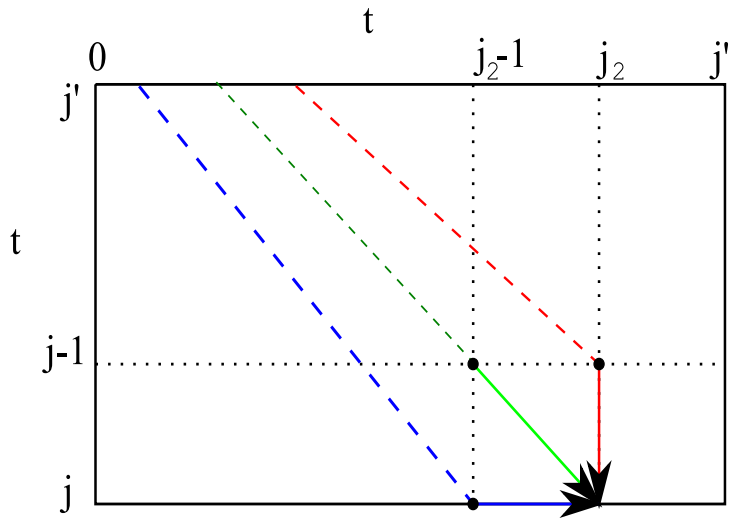

Figura 5.1: Ilustração das 3 possibilidades de um caminho ótimo até (j, j_2) a partir de qualquer vértice da linha j'. As linhas tracejadas indicam caminhos ótimos de qualquer vértice da linha j' até (j, j_2-1), $(j-1, j_2-1)$ e $(j-1, j_2)$.

- $W_t^A[j_2, j', j] = \max \begin{pmatrix} 0, \\ W_t^A[j_2-1, j', j] + \omega_h(t[j_2]) \end{pmatrix}$, se $0 \leq j' = j \leq m$ e $1 \leq j_2 \leq j'$,
- $W_t^A[j_2, j', j] = \max \begin{pmatrix} W_t^A[j_2-1, j', j] + \omega_h(t[j_2]), \\ W_t^A[j_2, j', j-1] + \omega_v(t[j]), \\ W_t^A[j_2-1, j', j-1] + \omega_d(t[j_2], t[j]) \end{pmatrix}$, se $0 \leq j' < j \leq m$ e $1 \leq j_2 \leq j'$,
- $W_t^A[j_2, j', j] = -\infty$, se $0 \leq j < j' \leq m$ ou $j' < j_2 \leq m$.

Seja $\widehat{W_t^A}$ a matriz $(m+1) \times (m+1)$ tal que

$$\widehat{W_t^A}[j', j] = \max_{\forall j_2 | 0 \leq j_2 \leq j'} \left(W_t^A[j_2, j', j] \right).$$

Ou seja, $\widehat{W_t^A}[j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1 \,..\, j] \times x$, $\forall\, x \in A_t^{j'+1,j}$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que $\omega_t^A(j', j) = \widehat{W_t^A}[j'-1, j]$.

O Algoritmo 10 constrói a matriz $\widehat{W_t^A}$ utilizando as recorrências descritas acima. O algoritmo utiliza programação dinâmica e executa em tempo $O(n^3)$ e espaço $O(n^2)$.

Algoritmo 10 Algoritmo para a construção da matriz $\widehat{W_t^A}$

```
BUILDWA(t)
 1  ▷ W_t^{A,j'}[j_2, j] = W_t^A[j_2, j', j]
 2  para j' de 0 até |t| faça
 3          j ← j'
 4          j_2 ← 0
 5          W_t^{A,j'}[j_2, j] ← 0
 6          Ŵ_t^A[j', j] ← W_t^{A,j'}[j_2, j]
 7          para j_2 de 1 até j' faça
 8                  W_t^{A,j'}[j_2, j] ← max(0, W_t^{A,j'}[j_2 − 1, j] + ω_h(t[j_2]))
 9                  Ŵ_t^A[j', j] ← max(Ŵ_t^A[j', j], W_t^{A,j'}[j_2, j])
10          para j de j' + 1 até |t| faça
11                  j_2 ← 0
12                  W_t^{A,j'}[j_2, j] ← W_t^{A,j'}[j_2, j − 1] + ω_v(t[j])
13                  Ŵ_t^A[j', j] ← W_t^{A,j'}[j_2, j]
14                  para j_2 de 1 até j' faça
15                          h ← W_t^{A,j'}[j_2 − 1, j] + ω_h(t[j_2])
16                          v ← W_t^{A,j'}[j_2, j − 1] + ω_v(t[j])
17                          d ← W_t^{A,j'}[j_2 − 1, j − 1] + ω_d(t[j_2], t[j])
18                          W_t^{A,j'}[j_2, j] ← max(h, v, d)
19                          Ŵ_t^A[j', j] ← max(Ŵ_t^A[j', j], W_t^{A,j'}[j_2, j])
20  devolva Ŵ_t^A
```

5.2.2 Função $\omega_t^B(j', j)$

Equivalentemente como feito para a matriz W_t^A, vamos definir a matriz W_t^B tal que, $W_t^B[j_3, j', j]$ contém a pontuação de um alinhamento ótimo A de $t[j'+1\,..\,j]\times t[j_3+1\,..\,j_4]$ e não existe outro alinhamento ótimo de $t[j'+1\,..\,j]\times t[j_3+1\,..\,j_4']$ cuja pontuação seja maior que a de A, onde $t[j_3+1\,..\,j_4] \in B_t^{j'+1,j}$ e $t[j_3+1\,..\,j_4'] \in B_t^{j'+1,j}$.

Seja W_t^B a matriz $(m+1)\times(m+1)\times(m+1)$ definida por

- $W_t^B[m, j', j] = 0$, se $0 \leq j' = j \leq m$,
- $W_t^B[m, j', j] = W_t^B[m, j'+1, j] + \omega_v(t[j'+1])$, se $0 \leq j' < j \leq m$,
- $W_t^B[j_3, j', j] = \max \begin{pmatrix} 0, \\ W_t^B[j_3+1, j', j] + \omega_h(t[j_3+1]) \end{pmatrix}$, se $0 \leq j' = j \leq m$ e $j \leq j_3 < m$,
- $W_t^B[j_3, j', j] = \max \begin{pmatrix} W_t^B[j_3+1, j', j] + \omega_h(t[j_3+1]), \\ W_t^B[j_3, j'+1, j] + \omega_v(t[j'+1]), \\ W_t^B[j_3+1, j'+1, j] + \omega_d(t[j_3+1], t[j'+1]) \end{pmatrix}$, se $0 \leq j' < j \leq m$ e $j \leq j_3 < m$,
- $W_t^B[j_3, j', j] = -\infty$, se $0 \leq j < j' \leq m$ ou $0 \leq j_3 < j$.

Ou seja, $W_t^B[j_3, j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times t[j_3+1\,..\,j_4]$, $\forall\ j_4 \mid\ j_3 \leq j_4 \leq m$.

A Figura 5.2 ilustra os caminhos num grafo de edição para a obtenção do valor de $W_t^B[j_3, j', j]$.

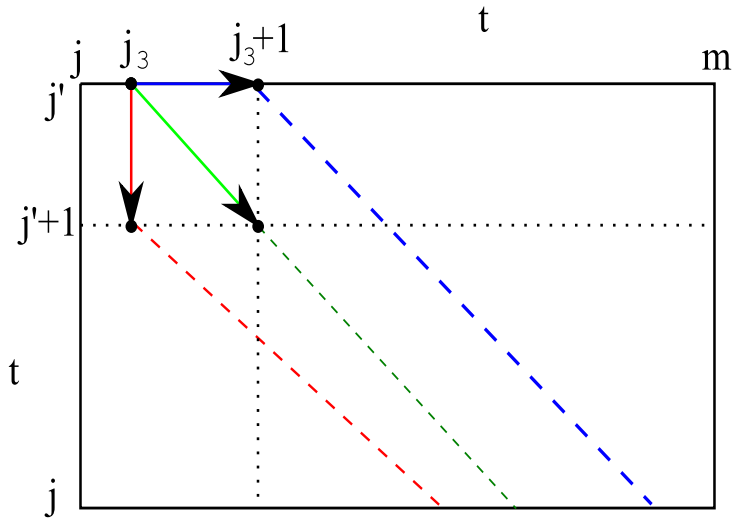

Figura 5.2: Ilustração das 3 possibilidades de um caminho ótimo de pontuação máxima a partir de (j', j_3) até qualquer vértice da linha j. As linhas tracejadas indicam caminhos ótimos de pontuação máxima de (j', j_3+1), $(j'+1, j_3+1)$ e $(j'+1, j_3)$ até qualquer vértice da linha j.

Seja $\widehat{W_t^B}$ a matriz $(m+1) \times (m+1)$ tal que

$$\widehat{W_t^B}[j', j] = \max_{\forall j_3 | j \leq j_3 \leq m} \left(W_t^B[j_3, j', j] \right).$$

Ou seja, $\widehat{W_t^B}[j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times x$, $\forall\ x \in B_t^{j'+1,j}$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que $\omega_t^B(j', j) = \widehat{W_t^B}[j'-1, j]$.

O Algoritmo 11 constrói a matriz $\widehat{W_t^B}$ utilizando as recorrências descritas acima. O algoritmo utiliza programação dinâmica e executa em tempo $O(n^3)$ e espaço $O(n^2)$.

Algoritmo 11 Algoritmo para a construção da matriz $\widehat{W_t^B}$

```
BUILDWB(t)
 1  ▷ W_t^{B,j}[j_3, j'] = W_t^B[j_3, j', j]
 2  para j de 0 até |t| faça
 3          j_3 ← |t|
 4          j' ← j
 5          W_t^{B,j}[j_3, j'] ← 0
 6          Ŵ_t^B[j', j] ← 0
 7          para j_3 de |t| − 1 descendo até j faça
 8                  W_t^{B,j}[j_3, j'] ← max(0, W_t^{B,j}[j_3 + 1, j'] + ω_h(t[j_3 + 1]))
 9                  Ŵ_t^B[j', j] ← max(Ŵ_t^B[j', j], W_t^{B,j}[j_3, j'])
10          para j' de j − 1 descendo até 0 faça
11                  j_3 ← |t|
12                  W_t^{B,j}[j_3, j'] ← W_t^{B,j}[j_3, j' + 1] + ω_v(t[j' + 1])
13                  Ŵ_t^B[j', j] ← W_t^{B,j}[j_3, j']
14                  para j_3 de |t| − 1 até j faça
15                          h ← W_t^{B,j}[j_3 + 1, j'] + ω_h(t[j_3 + 1])
16                          v ← W_t^{B,j}[j_3, j' + 1] + ω_v(t[j' + 1])
17                          d ← W_t^{B,j}[j_3 + 1, j' + 1] + ω_d(t[j_3 + 1], t[j' + 1])
18                          W_t^{B,j}[j_3, j'] ← max(h, v, d)
19                          Ŵ_t^B[j', j] ← max(Ŵ_t^B[j', j], W_t^{B,j}[j_3, j'])
20  devolva Ŵ_t^B
```

5.2.3 Função $\omega_t^C(j', j)$

Seja W_t^{C1} a matriz $(m+1) \times (m+1)$ tal que

$$W_t^{C1}[j', j] = W_t^A[j', j', j], \forall\ j', j \mid\ 0 \leq j' \leq j \leq m.$$

Ou seja, $W_t^{C1}[j', j]$, $0 \leq j' \leq j$, contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times t[j_5+1\,..\,j']$, $\forall\ j_5 \mid 0 \leq j_5 \leq j'$.

Seja W_t^{C2} a matriz $(m+1) \times (m+1)$ tal que

$$W_t^{C2}[j', j] = W_t^B[j, j', j], \forall\ j', j \mid\ 0 \leq j' \leq j \leq m.$$

Ou seja, $W_t^{C2}[j', j]$, $0 \leq j' \leq j$, contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times t[j+1\,..\,j_6]$, $\forall\ j_6 \mid j \leq j_6 \leq m$.

Seja $\widehat{W_t^C}$ a matriz $(m+1) \times (m+1)$ tal que

$$\widehat{W_t^C}[j', j] = \max_{\forall j_7 \mid j' \leq j_7 \leq j} \left(W_t^{C1}[j', j_7] + W_t^{C2}[j_7, j] \right), \forall\ j', j \mid\ 0 \leq j' \leq j \leq m.$$

Ou seja, $\widehat{W_t^C}[j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times x$, $\forall\ x \in C_t^{j'+1,j}$.

A Figura 5.3 ilustra os caminhos num grafo de edição para a obtenção do valor de $\widehat{W_t^C}[j', j]$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que $\omega_t^C(j', j) = \widehat{W_t^C}[j'-1, j]$.

Os Algoritmos 12 e 13 constroem as matrizes W_t^{C1}e W_t^{C2}, respectivamente. Com estas matrizes, o Algoritmo 14 constrói a matriz $\widehat{W_t^C}$. Esses algoritmos utilizam as equações descritas acima. Os algoritmos utilizam programação dinâmica e executam em tempo $O(n^3)$ e espaço $O(n^2)$.

5.2.4 Função $\omega_{t|s}^D(j', j)$

Equivalentemente como foi feito para a matriz W_t^A, vamos definir a matriz $W_{t|s}^D$ tal que, $W_{t|s}^D[i_2, j', j]$ contém a pontuação de um alinhamento ótimo A de $t[j'+1\,..\,j] \times s[i_1+1\,..\,i_2]$ e não existe outro alinhamento ótimo de $t[j'+1\,..\,j] \times s[i'_1+1\,..\,i_2]$ cuja pontuação seja maior que a de A, onde $s[i_1+1\,..\,i_2] \in D_s$ e $s[i'_1+1\,..\,i_2] \in D_s$.

Seja $W_{t|s}^D$ a matriz $(n+1) \times (m+1) \times (m+1)$ definida por

Algoritmo 12 Algoritmo para a construção da matriz W_t^{C1}

BUILDWC1(t)

1 ▷ $W_t^{A,j'}[j_2, j] = W_t^A[j_2, j', j]$
2 ▷ $W_t^{C1}[j', j] = W_t^{A,j'}[j', j] = W_t^A[j', j', j]$
3 **para** j' **de** 0 **até** $|t|$ **faça**
4 $\quad j \leftarrow j'$
5 $\quad j_2 \leftarrow 0$
6 $\quad W_t^{A,j'}[j_2, j] \leftarrow 0$
7 $\quad$ **para** j_2 **de** 1 **até** j' **faça**
8 $\quad\quad W_t^{A,j'}[j_2, j] \leftarrow \max(0, W_t^{A,j'}[j_2 - 1, j] + \omega_h(t[j_2]))$
9 $\quad W_t^{C1}[j', j] \leftarrow W_t^{A,j'}[j', j]$
10 $\quad$ **para** j **de** $j' + 1$ **até** $|t|$ **faça**
11 $\quad\quad j_2 \leftarrow 0$
12 $\quad\quad W_t^{A,j'}[j_2, j] \leftarrow W_t^{A,j'}[j_2, j - 1] + \omega_v(t[j])$
13 $\quad\quad$ **para** j_2 **de** 1 **até** j' **faça**
14 $\quad\quad\quad h \leftarrow W_t^{A,j'}[j_2 - 1, j] + \omega_h(t[j_2])$
15 $\quad\quad\quad v \leftarrow W_t^{A,j'}[j_2, j - 1] + \omega_v(t[j])$
16 $\quad\quad\quad d \leftarrow W_t^{A,j'}[j_2 - 1, j - 1] + \omega_d(t[j_2], t[j])$
17 $\quad\quad\quad W_t^{A,j'}[j_2, j] \leftarrow \max(h, v, d)$
18 $\quad\quad W_t^{C1}[j', j] \leftarrow W_t^{A,j'}[j', j]$
19 **devolva** W_t^{C1}

Algoritmo 13 Algoritmo para a construção da matriz W_t^{C2}

BUILDWC2(t)

1 ▷ $W_t^{B,j}[j_3, j'] = W_t^B[j_3, j', j]$
2 ▷ $W_t^{C2}[j', j] = W_t^{B,j}[j, j'] = W_t^B[j, j', j]$
3 **para** j **de** 0 **até** $|t|$ **faça**
4 $j_3 \leftarrow |t|$
5 $j' \leftarrow j$
6 $W_t^{B,j}[j_3, j'] \leftarrow 0$
7 **para** j_3 **de** $|t| - 1$ **descendo até** j **faça**
8 $W_t^{B,j}[j_3, j'] \leftarrow \max(0, W_t^{B,j}[j_3 + 1, j'] + \omega_h(t[j_3 + 1]))$
9 $W_t^{C2}[j', j] \leftarrow W_t^{B,j}[j, j']$
10 **para** j' **de** $j - 1$ **descendo até** 0 **faça**
11 $j_3 \leftarrow |t|$
12 $W_t^{B,j}[j_3, j'] \leftarrow W_t^{B,j}[j_3, j' + 1] + \omega_v(t[j' + 1])$
13 **para** j_3 **de** $|t| - 1$ **até** j **faça**
14 $h \leftarrow W_t^{B,j}[j_3 + 1, j'] + \omega_h(t[j_3 + 1])$
15 $v \leftarrow W_t^{B,j}[j_3, j' + 1] + \omega_v(t[j' + 1])$
16 $d \leftarrow W_t^{B,j}[j_3 + 1, j' + 1] + \omega_d(t[j_3 + 1], t[j' + 1])$
17 $W_t^{B,j}[j_3, j'] \leftarrow \max(h, v, d)$
18 $W_t^{C2}[j', j] \leftarrow W_t^{B,j}[j, j']$
19 **devolva** W_t^{C2}

Algoritmo 14 Algoritmo para a construção da matriz $\widehat{W_t^C}$

BUILDWC(t)

1 $W_t^{C1} \leftarrow BUILDWC1(t)$
2 $W_t^{C2} \leftarrow BUILDWC2(t)$
3 **para** j' **de** 0 **até** $|t|$ **faça**
4 **para** j **de** j' **até** $|t|$ **faça**
5 $\widehat{W_t^C}[j', j] \leftarrow W_t^{C1}[j', j'] + W_t^{C2}[j', j]$
6 **para** j_7 **de** $j' + 1$ **até** j **faça**
7 $\widehat{W_t^C}[j', j] \leftarrow \max(\widehat{W_t^C}[j', j], W_t^{C1}[j', j_7] + W_t^{C2}[j_7, j])$
8 **devolva** $\widehat{W_t^C}$

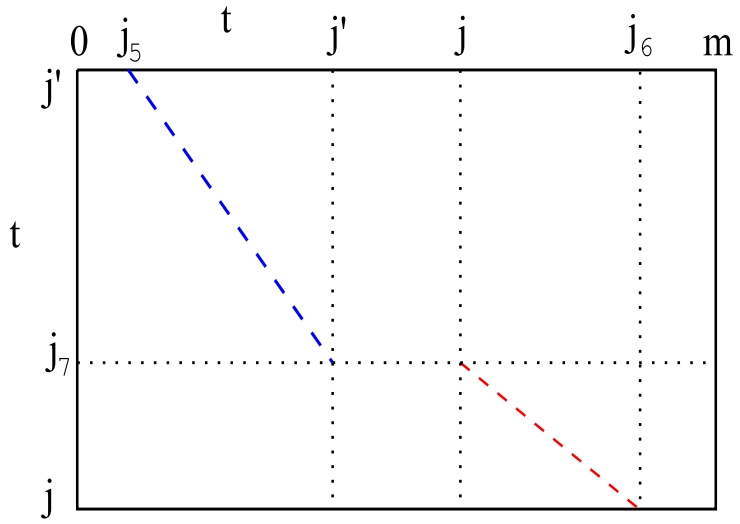

Figura 5.3: Os caminhos ótimos de (j', j_5) a (j_7, j') e de (j_7, j) a (j, j_6) são utilizados para obter o valor de $\widehat{W_t^C}[j', j]$.

- $W_{t|s}^D[0, j', j] = 0$, se $0 \leq j' = j \leq m$,
- $W_{t|s}^D[0, j', j] = W_{t|s}^D[0, j', j-1] + \omega_v(t[j])$, se $0 \leq j' < j \leq m$,
- $W_{t|s}^D[i_2, j', j] = \max \begin{pmatrix} 0, \\ W_{t|s}^D[i_2 - 1, j', j] + \omega_h(s[i_2]) \end{pmatrix}$, se $0 \leq j' = j \leq m$ e $1 \leq i_2 \leq n$,
- $W_{t|s}^D[i_2, j', j] = \max \begin{pmatrix} W_{t|s}^D[i_2 - 1, j', j] + \omega_h(s[i_2]), \\ W_{t|s}^D[i_2, j', j-1] + \omega_v(t[j]), \\ W_{t|s}^D[i_2 - 1, j', j-1] + \omega_d(s[i_2], t[j]) \end{pmatrix}$, se $0 \leq j' < j \leq m$ e $1 \leq i_2 \leq n$,
- $W_{t|s}^D[i_2, j', j] = -\infty$, se $0 \leq j < j' \leq m$.

Ou seja, $W_{t|s}^D[i_2, j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times s[i_1+1\,..\,i_2]$, $\forall\ i_1 \mid\ 0 \leq i_1 \leq n$.

A Figura 5.4 ilustra os caminhos num grafo de edição para a obtenção do valor de $W_{t|s}^D[i_2, j', j]$.

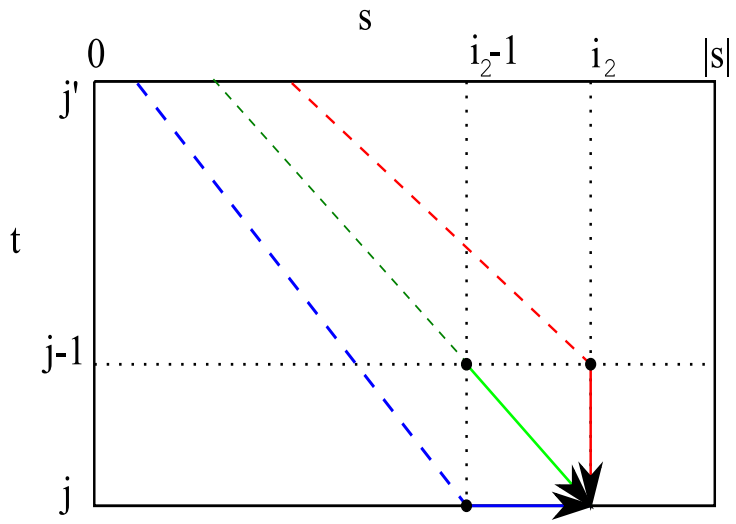

Figura 5.4: Ilustração das 3 possibilidades de um caminho ótimo de pontuação máxima a partir de qualquer vértice da linha j' até (j, i_2). As linhas tracejadas indicam caminhos ótimos de pontuação máxima de qualquer vértice da linha j' até $(j, i_2 - 1)$, $(j - 1, i_2 - 1)$ e $(j - 1, i_2)$.

Seja $\widehat{W^D_{t|s}}$ a matriz $(m + 1) \times (m + 1)$ tal que

$$\widehat{W^D_{t|s}}[j', j] = \max_{\forall i_2 | 0 \leq i_2 \leq n} \left(W^D_{t|s}[i_2, j', j] \right).$$

Ou seja, $\widehat{W^D_{t|s}}[j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j' + 1 \,..\, j] \times x$, $\forall\ x \in D_s$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que $\omega^D_{t|s}(j', j) = \widehat{W^D_{t|s}}[j' - 1, j]$.

O Algoritmo 15 constrói a matriz $\widehat{W^D_{t|s}}$ utilizando as recorrências descritas acima. O algoritmo utiliza programação dinâmica e executa em tempo $O(n^3)$ e espaço $O(n^2)$.

Algoritmo 15 Algoritmo para a construção da matriz $\widehat{W^D_{t|s}}$

BUILDWD(t, s)

1 $\triangleright\ W^{D,j'}_{t|s}[i_2, j] = W^D_{t|s}[i_2, j', j]$
2 **para** j' **de** 0 **até** $|t|$ **faça**
3 $\quad i_2 \leftarrow 0$
4 $\quad j \leftarrow j'$
5 $\quad W^{D,j'}_{t|s}[i_2, j] \leftarrow 0$
6 $\quad \widehat{W^D_{t|s}}[j', j] \leftarrow W^{D,j'}_{t|s}[i_2, j]$
7 $\quad$ **para** i_2 **de** 1 **até** $|s|$ **faça**
8 $\quad\quad W^{D,j'}_{t|s}[i_2, j] \leftarrow \max(0, W^{D,j'}_{t|s}[i_2 - 1, j] + \omega_h(s[i_2]))$
9 $\quad\quad \widehat{W^D_{t|s}}[j', j] \leftarrow \max(\widehat{W^D_{t|s}}[j', j], W^{D,j'}_{t|s}[i_2, j])$
10 $\quad$ **para** j **de** $j' + 1$ **até** $|t|$ **faça**
11 $\quad\quad i_2 \leftarrow 0$
12 $\quad\quad W^{D,j'}_{t|s}[i_2, j] \leftarrow W^{D,j'}_{t|s}[i_2, j - 1] + \omega_v(t[j])$
13 $\quad\quad \widehat{W^D_{t|s}}[j', j] \leftarrow W^{D,j'}_{t|s}[i_2, j]$
14 $\quad\quad$ **para** i_2 **de** 1 **até** $|s|$ **faça**
15 $\quad\quad\quad h \leftarrow W^{D,j'}_{t|s}[i_2 - 1, j] + \omega_h(s[i_2])$
16 $\quad\quad\quad v \leftarrow W^{D,j'}_{t|s}[i_2, j - 1] + \omega_v(t[j])$
17 $\quad\quad\quad d \leftarrow W^{D,j'}_{t|s}[i_2 - 1, j - 1] + \omega_d(s[i_2], t[j])$
18 $\quad\quad\quad W^{D,j'}_{t|s}[i_2, j] \leftarrow \max(h, v, d)$
19 $\quad\quad\quad \widehat{W^D_{t|s}}[j', j] \leftarrow \max(\widehat{W^D_{t|s}}[j', j], W^{D,j'}_{t|s}[i_2, j])$
20 **devolva** $\widehat{W^D_{t|s}}$

5.2.5 Função dup(t, j', j) e $M_t[i, j]$

Utilizando as matrizes definidas anteriormente, temos que

$$\mathrm{dup}(t, j', j) = \max \begin{pmatrix} \widehat{W_t^A}[j'-1, j] \\ \widehat{W_t^B}[j'-1, j] \\ \widehat{W_t^C}[j'-1, j] \\ \widehat{W_{t|s}^D}[j'-1, j] \end{pmatrix} + \omega_u(t[j' \,..\, j]).$$

Analogamente, temos que

$$\mathrm{dup}(s, i', i) = \max \begin{pmatrix} \widehat{W_s^A}[i'-1, i] \\ \widehat{W_s^B}[i'-1, i] \\ \widehat{W_s^C}[i'-1, i] \\ \widehat{W_{s|t}^D}[i'-1, i] \end{pmatrix} + \omega_x(s[i' \,..\, i]).$$

Logo, temos que

$$M_t[i, j] = \max_{\forall j' | 1 \leq j' \leq j} \left(M[i, j'-1] + \max \begin{pmatrix} \widehat{W_t^A}[j'-1, j] \\ \widehat{W_t^B}[j'-1, j] \\ \widehat{W_t^C}[j'-1, j] \\ \widehat{W_{t|s}^D}[j'-1, j] \end{pmatrix} + \omega_u(t[j' \,..\, j]) \right),$$

e que

$$M_s[i, j] = \max_{\forall i' | 1 \leq i' \leq i} \left(M[i'-1, j] + \max \begin{pmatrix} \widehat{W_s^A}[i'-1, i] \\ \widehat{W_s^B}[i'-1, i] \\ \widehat{W_s^C}[i'-1, i] \\ \widehat{W_{s|t}^D}[i'-1, i] \end{pmatrix} + \omega_x(s[i' \,..\, i]) \right).$$

5.2.6 Algoritmo DUP

O Algoritmo 16 constrói a matriz M de acordo com a recorrência 5.1 da página 126 e executa em tempo $O(n^3)$ e espaço $O(n^2)$. Os valores de $M_t[i, j]$ e $M_s[i, j]$ são obtidos de acordo com as equações descritas na seção 5.2.5.

Algoritmo 16 Algoritmo $O(n^3)$ para obter a pontuação de um alinhamento com duplicações

```
DUP(s, t)
 1  ▷ M[i, j] = ω_D(s[1 . . i], t[1 . . j])
 2  ▷ dup(t, j′, j) = MAXW_t[j′ − 1, j] + ω_u(t[j′ . . j])
 3  ▷ dup(s, i′, i) = MAXW_s[i′ − 1, i] + ω_x(s[i′ . . i])
 4  MAXW_t ← BUILDMAXW(t, s)
 5  MAXW_s ← BUILDMAXW(s, t)
 6  M[0, 0] ← 0
 7  para j de 1 até |t| faça
 8          M[0, j] ← M[0, j − 1] + ω_h(t[j])
 9          ▷ Obtém M_t[0, j]
10          para j′ de 0 até j − 1 faça
11                  dup_t ← MAXW_t[j′, j] + ω_u(t[j′ + 1 . . j])
12                  M[0, j] ← max(M[0, j], M[0, j′] + dup_t)
13  para i de 1 até |s| faça
14          M[i, 0] ← M[i − 1, 0] + ω_v(s[i]))
15          ▷ Obtém M_s[i, 0]
16          para i′ de 0 até i − 1 faça
17                  dup_s ← MAXW_s[i′, i] + ω_x(s[i′ + 1 . . i])
18                  M[i, 0] ← max(M[i, 0], M[i′, 0] + dup_s)
19          para j de 1 até |t| faça
20                  h ← M[i, j − 1] + ω_h(t[j])
21                  v ← M[i − 1, j] + ω_v(s[i])
22                  d ← M[i − 1, j − 1] + ω_d(s[i], t[j])
23                  M[i, j] ← max(h, v, d)
24                  ▷ Obtém M_t[i, j]
25                  para j′ de 0 até j − 1 faça
26                          dup_t ← MAXW_t[j′, j] + ω_u(t[j′ + 1 . . j])
27                          M[i, j] ← max(M[i, j], M[i, j′] + dup_t)
28                  ▷ Obtém M_s[i, j]
29                  para i′ de 0 até i − 1 faça
30                          dup_s ← MAXW_s[i′, i] + ω_x(s[i′ + 1 . . i])
31                          M[i, j] ← max(M[i, j], M[i′, j] + dup_s)
32  devolva M
```

Algoritmo 17 Algoritmo para a construção da matriz $MAXW_t$

BUILDMAXW(t, s)

1 $\widehat{W_t^A} \leftarrow BUILDWA(t)$
2 $\widehat{W_t^B} \leftarrow BUILDWB(t)$
3 $\widehat{W_t^C} \leftarrow BUILDWC(t)$
4 $\widehat{W_{t|s}^D} \leftarrow BUILDWD(t, s)$
5 **para** j **de** 0 **até** $|t|$ **faça**
6 **para** j' **de** 0 **até** $|t|$ **faça**
7 $W[j', j]] \leftarrow \max(\widehat{W_t^A}[j', j], \widehat{W_t^B}[j', j], \widehat{W_t^C}[j', j], \widehat{W_{t|s}^D}[j', j])$
8 **devolva** W

O Algoritmo 17 constrói as matrizes $MAXW_t$ e $MAXW_s$ quando chamados nas linhas 4 e 5 do Algoritmo 16, respectivamente. O algoritmo executa em tempo $O(n^3)$ e espaço $O(n^2)$. O elemento $MAXW_t[j', j]$ contém o máximo de $(\widehat{W_t^A}[j', j], \widehat{W_t^B}[j', j], \widehat{W_t^C}[j', j], \widehat{W_{t|s}^D}[j', j])$, assim como o elemento $MAXW_s[i', i]$ contém o máximo de $(\widehat{W_s^A}[i', i], \widehat{W_s^B}[i', i], \widehat{W_s^C}[i', i], \widehat{W_{s|t}^D}[i', i])$.

Fazendo algumas alterações no algoritmo, podemos obter também, além da pontuação do alinhamento com duplicações de s e t, o alinhamento propriamente dito sem alterar as complexidades de tempo e espaço de execução.

5.3 Duplicações em *tandem*

Há um tipo de ocorrência muito comum em seqüências biológicas que espera-se estar relacionado às duplicações: a ocorrência de um mesmo trecho da seqüência várias vezes um ao lado do outro. Estes trechos são chamados de repetições em *tandem*. Quando a repetição de uma operação de edição de duplicação é inserida imediatamente após (ou antes) da seqüência original, produzimos uma repetição em *tandem*. Nesta seção, vamos considerar que as operações de edição de duplicação e excisão somente inserem e removem, respectivamente, repetições em *tandem*. Vamos considerar somente estes tipos de duplicação ou excisão.

O exemplo 5.12 mostra uma seqüência real onde aparecem repetições em *tandem*. A seqüência $TGGCTG$ aparece 11 vezes em seguida na seqüência de DNA da *Pseudomonas aeruginosa PA01* das posições 98902 a 99067.

Exemplo 5.12 (Repetições em *tandem*) *Seqüência da* Pseudomonas aeruginosa PA01 *das posições 98902 a 99067:*
TGGCTG*TGGCTG****TGGCTG****TGGCTG****TGGCTG****TGGCTG*
*TGGCTG****TGGCTG****TGGCTG****TGGCTG***

Vamos considerar que a seqüência da repetição é a seqüência que está na posição posterior a da seqüência original. Apesar da repetição poder ser colocada na posição anterior a da seqüência original em uma operação de edição de duplicação, o resultado é sempre o mesmo se colocarmos a repetição na posição posterior em uma operação de edição de duplicação. Portanto, é indiferente a escolha por uma ou outra destas operações de edição de duplicação. Isto também é válido para a operação de edição de excisão.

Como já visto, $\Upsilon = \Upsilon_s \cup \Upsilon_t$, onde $\Upsilon_s = \{(s, x) \mid x \in [1, n]\}$ e $\Upsilon_t = \{(t, y) \mid y \in [1, m]\}$.

A seguir vamos definir um alinhamento com duplicações em *tandem* de s e t. Esta definição é muito parecida com a definição de alinhamento com duplicações de s e t, sendo que a única diferença é o espaço de busca da seqüência base de um intervalo de repetição, ou seja, são as possibilidades das seqüências X ou Y na definição a seguir.

Definição 5.13 (Alinhamento com duplicações em *tandem* de s e t) *Um alinhamento com duplicações em* tandem *de s e t é uma matriz A de 3 linhas e r colunas, tal que:*

- $r \geq \max(m, n)$ *e* $r \leq m + n$*;*
- *para toda coluna k de A temos que:*
  - $A[0, k] \in [1, n] \cup \{-\} \cup \Upsilon$*,*
  - $A[1, k] \in [1, m] \cup \{-\} \cup \Upsilon$*,*
  - $A[2, k] \in \{+, !, \#\}$*,*
  - *se* $A[0, k] = A[1, k]$ *então* $A[0, k] \in [1, n]$*,*
  - *se* $A[0, k] \in \Upsilon$ *então* $A[1, k] \notin \Upsilon$*,*
  - *se* $A[0, k] \in \Upsilon$ *ou* $A[1, k] \in \Upsilon$ *então* $A[2, k] \in \{!, \#\}$*;*
  - *se* $A[0, k] \in [1, n]$ *e* $A[1, k] \in [1, m]$ *então* $A[2, k] = +$*;*
  - *se* $A[2, k] = \#$ *então* $A[2, k - 1] \in \{!, \#\}$*;*

- *exclude*$(A[0, 0\,..\,r-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, n)$*;*
- *exclude*$(A[1, 0\,..\,r-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, m)$*;*
- *Para todo intervalo de repetição* $[k_1, k_2]$ *em* A *temos que:*
  1. *se* $A[0, k_1] \in [1, n] \cup \{-\}$ *e* $A[1, k_1] \in \Upsilon \cup \{-\}$ *então exclude*$(A[0, k_1\,..\,k_2], \{-\}) = (i_1, i_1+1, \ldots, i_1+x_1)$*, onde* $1 \leq i_1 \leq i_1+x_1 \leq n$*, e exclude*$(A[1, k_1\,..\,k_2], \{-\}) = X$*, onde* X *é uma das seguintes seqüências:*
     (a) $((s, i_2), (s, i_2+1), \ldots, (s, i_1-1))$*, onde* $1 \leq i_2 \leq i_1$*,*
     (b) $((t, j), (t, j+1), \ldots, (t, j+x_2))$*, onde* $1 \leq j \leq j+x_2+1 \leq m+1$ *e exclude*$(A[1, 0\,..\,k_1-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, j+x_2)$*;*
  2. *se* $A[1, k_1] \in [1, m] \cup \{-\}$ *e* $A[0, k_1] \in \Upsilon \cup \{-\}$ *então exclude*$(A[1, k_1\,..\,k_2], \{-\}) = (j_1, j_1+1, \ldots, j_1+y_1)$*, onde* $1 \leq j_1 \leq j_1+y_1 \leq m$*, e exclude*$(A[0, k_1\,..\,k_2], \{-\}) = Y$*, onde* Y *é uma das seguintes seqüências:*
     (a) $((t, j_2), (t, j_2+1), \ldots, (t, j_1-1))$*, onde* $1 \leq j_2 \leq j_1$*,*
     (b) $((s, i), (s, i+1), \ldots, (s, i+y_2))$*, onde* $1 \leq i \leq i+y_2+1 \leq n+1$ *e exclude*$(A[0, 0\,..\,k_1-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, i+y_2)$*.*

Repare que um alinhamento com duplicações em *tandem* de s e t é sempre um alinhamento com duplicações de s e t, porém o inverso nem sempre é verdade. Ou seja, o alinhamento com duplicações em *tandem* é um subcaso do alinhamento com duplicações.

O alinhamento A com duplicações do Exemplo 5.14, que é exatamente igual ao alinhamento com duplicações do Exemplo 5.7, que também é um alinhamento com duplicações em *tandem*. O intervalo de repetição $[5, 8]$ de A é um intervalo de repetição por excisão e está de acordo com o caso 1b de um intervalo de repetição na Definição 5.13. A seqüência $s[4\,..\,6]$ é a repetição e $t[3\,..\,5]$ é a seqüência base do intervalo de repetição $[5, 8]$ de A. Repare que $t[3\,..\,5]$ está alinhado com um sufixo de $s[1\,..\,3]$, ou seja, com um trecho de s que está exatamente antes da repetição.

Exemplo 5.14 (Alinhamento com duplicações em *tandem*)

$$A = \begin{bmatrix} - & 1 & 2 & 3 & - & 4 & 5 & 6 & - & 7 \\ 1 & 2 & 3 & 4 & 5 & (t,3) & - & (t,4) & (t,5) & 6 \\ + & + & + & + & + & ! & \# & \# & \# & + \end{bmatrix}$$

Vale a pena ressaltar que o grafo de duplicações G de um alinhamento com duplicações em *tandem* é acíclico, pois para toda aresta (u, v) de G, o vértice v correspondente a um símbolo que é alinhado numa coluna cujo índice é inferior à da coluna onde é alinhado o símbolo correspondente ao vértice u.

Utilizaremos neste capítulo, o mesmo sistema de pontuação definido na seção 5.1 e que já é utilizado no capítulo de alinhamento com duplicações.

Vamos redefinir as matrizes $M_t[i, j]$ e $M_s[i, j]$ de tal forma que os alinhamentos com duplicações estão restritos aos alinhamentos com duplicações em *tandem*.

Definimos $M_t[i, j]$ como a pontuação máxima de um alinhamento com duplicações em *tandem* A de $s[1\,..\,i] \times t[1\,..\,j]$ tal que, a última coluna de A pertence a um intervalo de repetição por duplicação, $\forall i, j \mid 0 \leq i \leq n$ e $1 \leq j \leq m$. De forma análoga, definimos $M_s[i, j]$ como a pontuação máxima de um alinhamento com duplicações em *tandem* A de $s[1\,..\,i] \times t[1\,..\,j]$ tal que, a última coluna de A pertence a um intervalo de repetição por excisão, $\forall i, j \mid 1 \leq i \leq n$ e $0 \leq j \leq m$.

Para obter os valores de $M[i, j]$, utilizaremos a mesma recorrência 5.1 da página 126 já utilizada no algoritmo 16.

Vamos dizer que $\text{dupt}(t, j', j, s, i)$ é o valor máximo para a soma das pontuações das colunas do intervalo de repetição por duplicação $[k_1, k_2]$ de um alinhamento com duplicações em *tandem* A de s e t tal que, $\text{exclude}(A[1, k_1\,..\,k_2], \{-\}) = (j', j'+1, \ldots, j)$ e $\text{exclude}(A[0, 1\,..\,k_1-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, i)$. De modo similar, vamos dizer que, o valor de $\text{dupt}(s, i', i, t, j)$ é o valor máximo para a soma das pontuações das colunas do intervalo de repetição por excisão $[k_1, k_2]$ de um alinhamento com duplicações em *tandem* A de s e t tal que, $\text{exclude}(A[0, k_1\,..\,k_2], \{-\}) = (i', i'+1, \ldots, i)$ e $\text{exclude}(A[1, 1\,..\,k_1-1], \Upsilon \cup \{-\}) = (1, 2, \ldots, j)$.

Portanto, temos que

$$M_t[i, j] = \max_{\forall j' \mid 1 \leq j' \leq j} (M[i, j'-1] + \text{dupt}(t, j', j, s, i))$$

Analogamente, temos que

$$M_s[i, j] = \max_{\forall i' \mid 1 \leq i' \leq i} (M[i'-1, j] + \text{dupt}(s, i', i, t, j))$$

Seja $G = (V, E, \omega)$ um grafo de edição estendido de s e t, onde ω é tal que:

- $\omega(\epsilon_H^{(i,j)}) = \omega_h(t[j])$,
- $\omega(\epsilon_D^{(i,j)}) = \omega_d(s[i], t[j])$,
- $\omega(\epsilon_V^{(i,j)}) = \omega_v(s[i])$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = \text{dupt}(s, i', i, t, j)$, se $i' \neq i$ e $j' = j$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = \text{dupt}(t, j', j, s, i)$, se $i' = i$ e $j' \neq j$,
- $\omega(\epsilon_{(i',j')}^{(i,j)}) = -\infty$ se $i' \neq i$ e $j' \neq j$.

Pode-se verificar que o peso de um caminho ótimo em G é igual a pontuação de um alinhamento ótimo com duplicações de s e t.

5.4 Algoritmo para alinhamento de duplicações em *tandem*

Seja $E^{t,j}$ o conjunto de todos os sufixos de $t[1\,..\,j]$, assim como $E^{s,i}$ é o conjunto de todos os sufixos de $s[1\,..\,i]$, ou seja, se i e j são tais que $1 \leq j \leq m$ e $1 \leq i \leq n$ definimos os conjuntos de fatores das seqüências s e t, $E^{t,j}$ e $E^{s,i}$ da seguinte forma:

1. $E^{t,j} = \{t[j_1\,..\,j] \mid 1 \leq j_1 \leq j+1\}$ e
2. $E^{s,i} = \{s[i_1\,..\,i] \mid 1 \leq i_1 \leq i+1\}$.

Para todos j' e j tais que $1 \leq j' \leq j \leq m$ definimos, para cada conjunto $E^{t,j}$ e $E^{s,i}$, a pontuação máxima do alinhamento ótimo sem duplicações de $t[j'\,..\,j]$ com um elemento do conjunto, da seguinte forma:

$$\begin{aligned}\omega_t^E(j', j) &= \max(\omega^*(t[j'\,..\,j], x) \mid x \in E^{t,j'-1}) \\ &= \max_{\forall j_1 \mid 1 \leq j_1 \leq j'} (\omega^*(t[j'\,..\,j], t[j_1\,..\,j'-1]))\,,\end{aligned}$$

$$\begin{aligned}\omega_{t|s}^F(j', j, i) &= \max(\omega^*(t[j'\,..\,j], x) \mid x \in E^{s,i}) \\ &= \max_{\forall i_1 \mid 1 \leq i_1 \leq i+1} (\omega^*(t[j'\,..\,j], s[i_1\,..\,i]))\,,\end{aligned}$$

Analogamente, para o fator $s[i' \,..\, i]$, definimos

$$\begin{aligned}\omega_s^E(i', i) &= \max(\omega^*(s[i' \,..\, i], x) \mid x \in E^{s,i'-1}) \\ &= \max_{\forall i_1 | 1 \leq i_1 \leq i'} (\omega^*(s[i' \,..\, i], s[i_1 \,..\, i' - 1])),\end{aligned}$$

$$\begin{aligned}\omega_{s|t}^F(i', i, j) &= \max(\omega^*(s[i' \,..\, i], x) \mid x \in E^{t,j}) \\ &= \max_{\forall j_1 | 1 \leq j_1 \leq j+1} (\omega^*(s[i' \,..\, i], t[j_1 \,..\, j])),\end{aligned}$$

Portanto, para todos i, j' e j tais que $0 \leq i \leq n$ e $1 \leq j' \leq j \leq m$, temos que

$$\text{dupt}(t, j', j, s, i) = \max \left(\begin{array}{c} \omega_t^E(j', j) \\ \omega_{t|s}^F(j', j, i) \end{array} \right) + \omega_u(t[j' \,..\, j]).$$

Analogamente, para todos j, i' e i tais que $0 \leq j \leq m$ e $1 \leq i' \leq i \leq n$, temos que

$$\text{dupt}(s, i', i, t, j) = \max \left(\begin{array}{c} \omega_s^E(i', i) \\ \omega_{s|t}^F(i', i, j) \end{array} \right) + \omega_x(s[i' \,..\, i]).$$

A seguir, vamos definir matrizes que armazenam valores pré-computados de $\omega_t^E(j', j)$ e $\omega_{t|s}^F(j', j, i)$ e propor algoritmos para construir estas matrizes.

5.4.1 Função $\omega_t^E(j', j)$

Vamos armazenar o valor de $\omega_t^E(j', j)$ em $\widehat{W_t^E}[j' - 1, j]$, ou seja, para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que $\omega_t^E(j', j) = \widehat{W_t^E}[j' - 1, j]$.

Assim, para todos j' e j tais que $1 \leq j' \leq j \leq m$ temos que, $\widehat{W_t^E}[j', j]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j' + 1 \,..\, j] \times x$, $\forall\ x \in E^{t,j'+1}$.

Para obter os valores de $\widehat{W_t^E}$, utilizaremos a matriz W_t^E $(m+1) \times (m+1) \times (m+1)$ definida por

$$W_t^E[j_2, j', j] = \max_{\forall j_1 | 0 \leq j_1 \leq j_2} (\omega^*(t[j' + 1 \,..\, j], t[j_1 + 1 \,..\, j_2]))$$

, se $0 \leq j_2 \leq j' \leq j \leq m$ e
$W_t^E[j_2, j', j] = -\infty$, caso contrário.

Portanto, $\widehat{W_t^E}$ é a matriz $(m+1) \times (m+1)$ tal que

$$\widehat{W_t^E}[j', j] = W_t^E[j', j', j].$$

Os valores dos elementos de W_t^E são obtidos da seguinte forma:

- $W_t^E[0, j', j] = 0$, se $0 \leq j' = j \leq m$,
- $W_t^E[0, j', j] = W_t^E[0, j', j-1] + \omega_v(t[j])$, se $0 \leq j' < j \leq m$,
- $W_t^E[j_2, j', j] = \max \begin{pmatrix} 0, \\ W_t^E[j_2 - 1, j', j] + \omega_h(t[j_2]) \end{pmatrix}$, se $0 \leq j' = j \leq m$ e $1 \leq j_2 \leq j'$,
- $W_t^E[j_2, j', j] = \max \begin{pmatrix} W_t^E[j_2 - 1, j', j] + \omega_h(t[j_2]), \\ W_t^E[j_2, j', j-1] + \omega_v(t[j]), \\ W_t^E[j_2 - 1, j', j-1] + \omega_d(t[j_2], t[j]) \end{pmatrix}$, se $0 \leq j' < j \leq m$ e $1 \leq j_2 \leq j'$,
- $W_t^E[j_2, j', j] = -\infty$, se $0 \leq j < j' \leq m$ ou $j' < j_2 \leq m$.

A Figura 5.5 ilustra as 3 possibilidades de um caminho ótimo num grafo de edição até (j, j') a partir de qualquer vértice da linha j'. As linhas tracejadas indicam caminhos ótimos de qualquer vértice da linha j' até $(j-1, j')$, $(j-1, j'-1)$ e $(j, j'-1)$, cujos pesos são, respectivamente, $W_t^E[j', j', j-1]$, $W_t^E[j'-1, j', j-1]$ e $W_t^E[j'-1, j', j]$.

Repare que $W_t^E[j_2, j', j] = W_t^A[j_2, j', j]$. No entanto, $\widehat{W_t^E}[j', j] \neq \widehat{W_t^A}[j', j]$, pois $\widehat{W_t^E}[j', j]$ é o valor de $W_t^A[j_2, j', j]$ quando $j_2 = j'$, diferente de $\widehat{W_t^A}[j', j]$ que é a pontuação máxima de $W_t^A[j_2, j', j]$ para j' e j fixos.

O Algoritmo 18 constrói a matriz $\widehat{W_t^E}$ utilizando as recorrências descritas acima. O algoritmo é muito parecido com o Algoritmo 10 que constrói a matriz $\widehat{W_t^A}$. O algoritmo utiliza programação dinâmica e executa em tempo $O(n^3)$ e espaço $O(n^2)$.

Algoritmo 18 Algoritmo para a construção da matriz $\widehat{W_t^E}$

```
BUILDWE(t)
 1  ▷ W_t^{E,j'}[j_2, j] = W_t^E[j_2, j', j]
 2  para j' de 0 até |t| faça
 3          j ← j'
 4          j_2 ← 0
 5          W_t^{E,j'}[j_2, j] ← 0
 6          para j_2 de 1 até j' faça
 7                  W_t^{E,j'}[j_2, j] ← max(0, W_t^{E,j'}[j_2 − 1, j] + ω_h(t[j_2]))
 8          Ŵ_t^E[j', j] ← W_t^{E,j'}[j', j]
 9          para j de j' + 1 até |t| faça
10                  j_2 ← 0
11                  W_t^{E,j'}[j_2, j] ← W_t^{E,j'}[j_2, j − 1] + ω_v(t[j])
12                  para j_2 de 1 até j' faça
13                          h ← W_t^{E,j'}[j_2 − 1, j] + ω_h(t[j_2])
14                          v ← W_t^{E,j'}[j_2, j − 1] + ω_v(t[j])
15                          d ← W_t^{E,j'}[j_2 − 1, j − 1] + ω_d(t[j_2], t[j])
16                          W_t^{E,j'}[j_2, j] ← max(h, v, d)
17                  Ŵ_t^E[j', j] ← W_t^{E,j'}[j', j]
18 devolva Ŵ_t^E
```

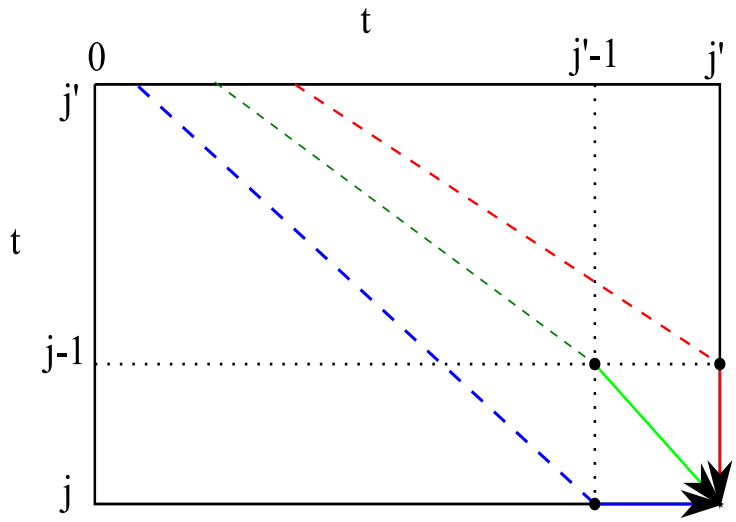

Figura 5.5: Ilustração das 3 possibilidades de um caminho ótimo num grafo de edição até (j, j') a partir de qualquer vértice da linha j' para a obtenção de $\widehat{W_t^E}[j', j] = W_t^E[j', j', j]$.

5.4.2 Funções $\omega_{t|s}^F(j', j, i)$ e $\omega_{s|t}^F(i', i, j)$

Vamos armazenar o valor de $\omega_{t|s}^F(j', j, i)$ em $\widehat{W_{t|s}^F}[j'-1, j, i]$, ou seja, para todos i, j' e j tais que $0 \leq i \leq n$ e $1 \leq j' \leq j \leq m$, temos que $\omega_{t|s}^F(j', j, i) = \widehat{W_{t|s}^F}[j'-1, j, i]$.

Assim temos que, para todos i, j' e j tais que $0 \leq i \leq n$ e $1 \leq j' \leq j \leq m$, $\widehat{W_{t|s}^F}[j', j, i]$ contém o valor máximo do alinhamento ótimo sem duplicações de $t[j'+1\,..\,j] \times x$, $\forall\ x \in E^{s,i}$.

Os valores dos elementos da matriz $\widehat{W_{t|s}^F}$ $(m+1) \times (m+1) \times (n+1)$ são obtidos da seguinte forma:

- $\widehat{W_{t|s}^F}[j', j, 0] = 0$, se $0 \leq j' = j \leq m$,
- $\widehat{W_{t|s}^F}[j', j, 0] = \widehat{W_{t|s}^F}[j', j-1, 0] + \omega_v(t[j])$, se $0 \leq j' < j \leq m$,
- $\widehat{W_{t|s}^F}[j', j, i] = \max \left(\begin{array}{l} 0, \\ \widehat{W_{t|s}^F}[j', j, i-1] + \omega_h(s[i]) \end{array} \right)$, se $0 \leq j' = j \leq m$ e $1 \leq i \leq n$,

- $\widehat{W^F_{t|s}}[j', j, i] = \max \begin{pmatrix} \widehat{W^F_{t|s}}[j', j, i-1] + \omega_h(s[i]), \\ \widehat{W^F_{t|s}}[j', j-1, i] + \omega_v(t[j]), \\ \widehat{W^F_{t|s}}[j', j-1, i-1] + \omega_d(t[j], s[i]) \end{pmatrix}$, se $0 \leq j' < j \leq m$ e $1 \leq i \leq n$,
- $\widehat{W^F_{t|s}}[j', j, i] = -\infty$, se $j < j'$.

A Figura 5.6 ilustra as 3 possibilidades de um caminho ótimo num grafo de edição até (j, i) a partir de qualquer vértice da linha j'. As linhas tracejadas indicam caminhos ótimos de qualquer vértice da linha j' até $(j-1, i)$, $(j-1, i-1)$ e $(j, i-1)$, cujos pesos são, respectivamente, $\widehat{W^F_{t|s}}[j', j-1, i]$, $\widehat{W^F_{t|s}}[j', j-1, i-1]$ e $\widehat{W^F_{t|s}}[j', j, i-1]$.

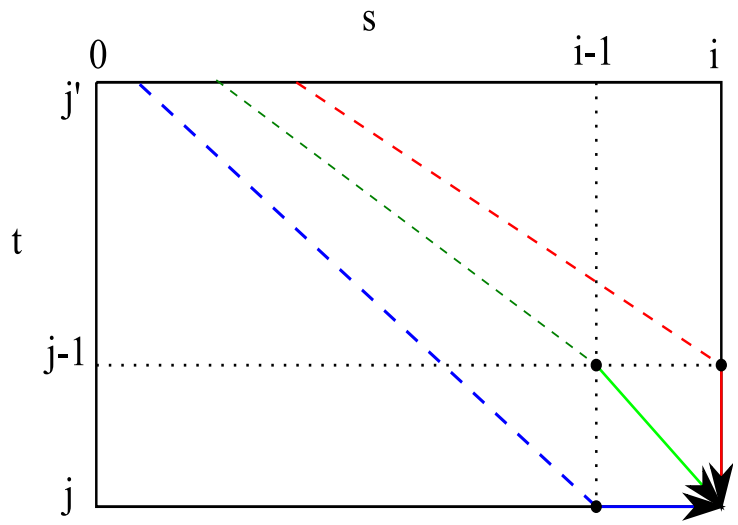

Figura 5.6: Ilustração das 3 possibilidades de um caminho ótimo num grafo de edição até (j, i) a partir de qualquer vértice da linha j' para a obtenção de $\widehat{W^F_{t|s}}[j', j, i]$.

Similarmente como feito para a matriz $\widehat{W^F_{t|s}}$, definimos a matriz $\widehat{W^F_{s|t}}$ $(n+1) \times (n+1) \times (m+1)$ da seguinte forma:

- $\widehat{W^F_{s|t}}[i', i, 0] = 0$, se $0 \leq i' = i \leq n$,
- $\widehat{W^F_{s|t}}[i', i, 0] = \widehat{W^F_{s|t}}[i', i-1, 0] + \omega_v(s[i])$, se $0 \leq i' < i \leq n$,

- $\widehat{W^F_{s|t}}[i', i, j] = \max \begin{pmatrix} 0, \\ \widehat{W^F_{s|t}}[i', i, j-1] + \omega_h(t[j]) \end{pmatrix}$, se $0 \leq i' = i \leq n$ e $1 \leq j \leq m$,
- $\widehat{W^F_{s|t}}[i', i, j] = \max \begin{pmatrix} \widehat{W^F_{s|t}}[i', i, j-1] + \omega_h(t[j]), \\ \widehat{W^F_{s|t}}[i', i-1, j] + \omega_v(s[i]), \\ \widehat{W^F_{s|t}}[i', i-1, j-1] + \omega_d(s[i], t[j]) \end{pmatrix}$, se $0 \leq i' < i \leq n$ e $1 \leq j \leq m$,
- $\widehat{W^F_{s|t}}[i', i, j] = -\infty$, se $i < i'$.

Sejam as matrizes $W^{F,i}_{t|s}$ $(|t|+1) \times (|t|+1)$ e $W^{F,i}_{s|t}$ $(|s|+1) \times (|t|+1)$ tais que, $W^{F,i}_{t|s}[j', j] = \widehat{W^F_{t|s}}[j', j, i]$ e $W^{F,i}_{s|t}[i', j] = \widehat{W^F_{s|t}}[i', i, j]$.

Os Algoritmos 19 e 20 constroem as matrizes $W^{F,i}_{t|s}$ e $W^{F,i}_{s|t}$, respectivamente, utilizando as recorrências descritas acima. Os algoritmos utilizam programação dinâmica e executam em tempo $O(n^2)$ e espaço $O(n^2)$.

5.4.3 Função dupt(t, j', j, s, i) e $M_t[i, j]$

Para todos i, j' e j tais que $0 \leq i \leq n$ e $1 \leq j' \leq j \leq m$, temos que

$$\text{dupt}(t, j', j, s, i) = \max \begin{pmatrix} \widehat{W^E_t}[j'-1, j] \\ W^{F,i}_{t|s}[j'-1, j] \end{pmatrix} + \omega_u(t[j' \,..\, j]).$$

Analogamente, para todos j, i' e i tais que $0 \leq j \leq m$ e $1 \leq i' \leq i \leq n$, temos que

$$\text{dupt}(s, i', i, t, j) = \max \begin{pmatrix} \widehat{W^E_s}[i'-1, i] \\ W^{F,i}_{s|t}[i'-1, j] \end{pmatrix} + \omega_x(s[i' \,..\, i]).$$

Temos que

$$M_t[i, j] = \max_{\forall j' | 1 \leq j' \leq j} \left(M[i, j'-1] + \max \begin{pmatrix} \widehat{W^E_t}[j'-1, j] \\ W^{F,i}_{t|s}[j'-1, j] \end{pmatrix} + \omega_u(t[j' \,..\, j]) \right).$$

Algoritmo 19 Algoritmo para a construção da matriz $W^{F,i}_{t|s}$

BUILDWFT$(t, s, i, W^{F,i-1}_{t|s})$

1 $\triangleright\ W^{F,i}_{t|s}[j', j] = \widehat{W^{F}_{t|s}}[j', j, i]$
2 **se** $i = 0$ **então**
3 **para** j' **de** 0 **até** $|t|$ **faça**
4 $j \leftarrow j'$
5 $W^{F,i}_{t|s}[j', j] \leftarrow 0$
6 **para** j **de** $j' + 1$ **até** $|t|$ **faça**
7 $W^{F,i}_{t|s}[j', j] \leftarrow W^{F,i}_{t|s}[j', j-1] + \omega_v(t[j])$
8 **senão**
9 **para** j' **de** 0 **até** $|t|$ **faça**
10 $j \leftarrow j'$
11 $W^{F,i}_{t|s}[j', j] \leftarrow \max(0, W^{F,i-1}_{t|s}[j', j] + \omega_h(s[i]))$
12 **para** j **de** $j' + 1$ **até** $|t|$ **faça**
13 $h \leftarrow W^{F,i-1}_{t|s}[j', j] + \omega_h(s[i])$
14 $v \leftarrow W^{F,i}_{t|s}[j', j-1] + \omega_v(t[j])$
15 $d \leftarrow W^{F,i-1}_{t|s}[j', j-1] + \omega_d(t[j], s[i])$
16 $W^{F,i}_{t|s}[j', j] \leftarrow \max(h, v, d)$
17 **devolva** $W^{F,i}_{t|s}$

Algoritmo 20 Algoritmo para a construção da matriz $W_{s|t}^{F,i}$

BUILDWFS$(s, t, i, W_{s|t}^{F,i-1})$

1 $\triangleright\ W_{s|t}^{F,i}[i', j] = \widehat{W_{s|t}^{F}}[i', i, j]$
2 **para** i' **de** 0 **até** i **faça**
3 $\quad j \leftarrow 0$
4 $\quad$ **se** $i' = i$ **então**
5 $\quad\quad W_{s|t}^{F,i}[i', j] \leftarrow 0$
6 $\quad$ **senão**
7 $\quad\quad W_{s|t}^{F,i}[i', j] \leftarrow W_{s|t}^{F,i-1}[i', j] + \omega_v(s[i])$
8 $\quad$ **para** j **de** 1 **até** $|t|$ **faça**
9 $\quad\quad$ **se** $i' = i$ **então**
10 $\quad\quad\quad W_{s|t}^{F,i}[i', j] \leftarrow \max(0, W_{s|t}^{F,i}[i', j-1] + \omega_h(t[j]))$
11 $\quad\quad$ **senão**
12 $\quad\quad\quad h \leftarrow W_{s|t}^{F,i}[i', j-1] + \omega_h(t[j])$
13 $\quad\quad\quad v \leftarrow W_{s|t}^{F,i-1}[i', j] + \omega_v(s[i])$
14 $\quad\quad\quad d \leftarrow W_{s|t}^{F,i-1}[i', j-1] + \omega_d(s[i], t[j])$
15 $\quad\quad\quad W_{s|t}^{F,i}[i', j] \leftarrow \max(h, v, d)$
16 **devolva** $W_{s|t}^{F,i}$

Do mesmo modo temos que

$$M_s[i,j] = \max_{\forall i' | 1 \leq i' \leq i} \left(M[i'-1,j] + \max \left(\begin{array}{c} \widehat{W_s^E}[i'-1,i] \\ W_{s|t}^{F,i}[i'-1,j] \end{array} \right) + \omega_x(s[i'\,..\,i]) \right).$$

5.4.4 Algoritmo DUPTandem

O Algoritmo 21 constrói a matriz M de acordo com a recorrência 5.1 na página 126 e executa em tempo $O(n^3)$ e espaço $O(n^2)$. Os valores de $M_t[i,j]$ e $M_s[i,j]$ são obtidos de acordo com as equações descritas na seção 5.4.3.

Fazendo algumas alterações no algoritmo, podemos obter também, além da pontuação do alinhamento com duplicações de s e t, o alinhamento propriamente dito sem alterar as complexidades de tempo e espaço de execução.

O espaço de busca das seqüências base de um intervalo de repetição utilizado neste modelo, é maior que o espaço de busca utilizado no modelo proposto por Benson [11]. Contudo a complexidade do tempo de execução $O(n^3)$ do algoritmo 21 que propomos é muito menor do que as complexidades de tempo dos algoritmos exatos propostos por Benson, que são $O(n^5)$ e $O(n^4)$, sendo que o $O(n^4)$ utiliza $O(n^3)$ de memória e o $O(n^5)$ utiliza $O(n^2)$ de memória.

Acreditamos que como verificamos mais opções de duplicações, o algoritmo que propomos deve obter alinhamentos, em geral, mais realistas que os alinhamentos propostos por Benson.

Algoritmo 21 Algoritmo $O(n^3)$ para obter a pontuação de um alinhamento com duplicações em *tandem*

DUPTANDEM(s, t)

```
1  Ŵ_t^E ← BUILDWE(t)
2  Ŵ_s^E ← BUILDWE(s)
3  W_{t|s}^{F,i} ← BUILDWFt(t, s, 0, ∅)
4  M[0, 0] ← 0
5  para j de 1 até |t| faça
6          M[0, j] ← M[0, j − 1] + ω_h(t[j])
7          ▷ Obtém M_t[0, j]
8          para j′ de 0 até j − 1 faça
9                  dup_t ← max(Ŵ_t^E[j′, j], W_{t|s}^{F,i}[j′, j]) + ω_u(t[j′ + 1 . . j])
10                 M[0, j] ← max(M[0, j], M[0, j′] + dup_t)
11 para i de 1 até |s| faça
12         M[i, 0] ← M[i − 1, 0] + ω_v(s[i]))
13         W_{t|s}^{F,i} ← BUILDWFt(t, s, i, W_{t|s}^{F,i−1})
14         W_{s|t}^{F,i} ← BUILDWFs(s, t, i, W_{s|t}^{F,i−1})
15         ▷ Obtém M_s[i, 0]
16         para i′ de 0 até i − 1 faça
17                 dup_s ← max(Ŵ_s^E[i′, i], W_{s|t}^{F,i}[i′, 0]) + ω_x(s[i′ + 1 . . i])
18                 M[i, 0] ← max(M[i, 0], M[i′, 0] + dup_s)
19         para j de 1 até |t| faça
20                 h ← M[i, j − 1] + ω_h(t[j])
21                 v ← M[i − 1, j] + ω_v(s[i])
22                 d ← M[i − 1, j − 1] + ω_d(s[i], t[j])
23                 M[i, j] ← max(h, v, d)
24                 ▷ Obtém M_t[i, j]
25                 para j′ de 0 até j − 1 faça
26                         dup_t ← max(Ŵ_t^E[j′, j], W_{t|s}^{F,i}[j′, j]) + ω_u(t[j′ + 1 . . j])
27                         M[i, j] ← max(M[i, j], M[i, j′] + dup_t)
28                 ▷ Obtém M_s[i, j]
29                 para i′ de 0 até i − 1 faça
30                         dup_s ← max(Ŵ_s^E[i′, i], W_{s|t}^{F,i}[i′, j]) + ω_x(s[i′ + 1 . . i])
31                         M[i, j] ← max(M[i, j], M[i′, j] + dup_s)
32 devolva M
```

Capítulo 6

Conclusão

Para a comparação de duas seqüências, desenvolvemos e apresentamos algoritmos exatos que acreditamos serem inéditos e que obtêm alinhamentos que consideram a possibilidade da ocorrência de outros eventos biológicos além dos eventos de inserção, remoção e substituição de um símbolo da seqüência, comumente utilizados em outros algoritmos usuais para obtenção de alinhamentos ótimos.

Consideramos, além dos eventos usuais que agem sobre um único símbolo da seqüência, alguns rearranjos comumente observados no processo evolutivo, tais como inversão, duplicação por transposição e duplicação em *tandem*, que agem sobre segmentos de vários símbolos. Esperamos com isto, que o alinhamento obtido esteja mais próximo de mostrar o que realmente ocorreu na evolução.

Apesar do problema da obtenção de um alinhamento ótimo com inversões não sobrepostas ser um problema já bem estudado e com alguns algoritmos já publicados, conseguimos desenvolver algoritmos com complexidade de tempo significativamente melhor que os já existentes.

Apesar das complexidades de tempo dos algoritmos que propomos serem melhores que as complexidades de tempo dos algoritmos até então conhecidos, ainda assim os tempos ($O(n^3)$) são elevados para seqüências de grande comprimento. Por exemplo, um teste realizado com duas seqüências de DNA com 700 bases levou 872 segundos (14,53 minutos) para executar o algoritmo $O(n^3)$ que obtém o alinhamento ótimo com inversões não sobrepostas[1]. A solução que adotamos para obter alinhamentos ótimos com inversões não

[1] A execução deste mesmo teste no algoritmo $O(n^3 \log n)$ que obtém o alinhamento ótimo com inversões não sobrepostas, levou 42,27 minutos.

sobrepostas de seqüências muito longas, foi fragmentar as seqüências (por exemplo em fragmentos de comprimento 100), estabelecer quais os fragmentos que se emparelham[2] e considerar as seqüências dos fragmentos como as seqüências para o alinhamento com inversões não sobrepostas.

No caso do alinhamento com duplicações, não conhecemos nenhum outro trabalho que utilize os modelos de duplicações que apresentamos. O modelo mais próximo ao modelo que propomos para alinhamentos com duplicações em *tandem*, que conhecemos, é o modelo proposto por Benson [11], que propôs dois algoritmos exatos para a obtenção de um alinhamento ótimo com duplicações em *tandem*: um que executa com tempo $O(n^5)$ e espaço $O(n^2)$ e outro que executa em tempo $O(n^4)$ e espaço $O(n^3)$. Portanto, estes algoritmos têm complexidades de tempo e memória muito piores que as do algoritmo que propomos. O modelo proposto por Benson para considerar duplicações em *tandem*, além da restrição de só considerar repetições em *tandem*, tem mais duas restrições, que são as seguintes.

1. Sejam s e t as seqüências a serem alinhadas. Se um trecho de s é uma repetição então a seqüência base a ser comparada (ou alinhada) com esta repetição em s deve ser um trecho de t. Portanto este modelo do Benson não considera que a seqüência original da repetição pode estar na própria seqüência onde está a repetição.

2. A seqüência base deve ser a mesma para intervalos de repetição consecutivos, ou seja, como as repetições consideradas são em *tandem* e a seqüência original considerada está sempre na outra seqüência, as seqüências base dos intervalos de repetição consecutivos devem ter o mesmo tamanho.

Nos modelos que propomos para o alinhamento com duplicações em *tandem*, não impomos estas restrições e portanto nosso espaço de busca é maior que o do Benson. No entanto, é possível obter um alinhamento ótimo com estas mesmas restrições propostas por Benson, com um algoritmo com tempo de execução $O(n^3)$ e espaço $O(n^2)$ utilizando as técnicas utilizados no algoritmo $O(n^3)$ para alinhamentos ótimos com inversões não sobrepostas, e portanto, mesmo considerando o mesmo modelo mais restritivo proposto por Benson, ainda assim conseguimos um algoritmo com complexidade de tempo $O(n^3)$, melhor que os algoritmos exatos propostos por Benson.

[2]Isto foi feito, nos testes que realizamos, com o programa BLAST e com um limite mínimo para considerar que dois fragmentos se emparelham.

Infelizmente os algoritmos de alinhamento com duplicações e com duplicações em *tandem*, não puderam ser implementados, de forma que faltam testes experimentais.

Acreditamos que com um sistema de pontuação bem elaborado, os algoritmos para obter um alinhamento ótimo com duplicações (e com duplicações em *tandem*) conseguem obter alinhamentos que estão mais próximos à realidade na comparação de seqüências de espécies próximas.

Capítulo 7

Trabalhos futuros

7.1 Alinhamento ótimo com inversões não sobrepostas e duplicações

Neste texto, desenvolvemos algoritmos para a obtenção da pontuação de alinhamentos ótimos com inversões não sobrepostas, e algoritmos para a obtenção da pontuação de alinhamentos ótimos com duplicações, mas não desenvolvemos um algoritmo para a obtenção da pontuação de um alinhamento ótimo com inversões não sobrepostas **e** duplicações, ou seja, não consideramos a possibilidade da ocorrência dos eventos de inversões não sobrepostas e de duplicações no mesmo alinhamento.

Pretendemos, num trabalho futuro, desenvolver um algoritmo para a obtenção da pontuação de alinhamentos ótimos que consideram a possibilidade da ocorrência dos eventos de inversões não sobrepostas e de duplicações no mesmo alinhamento.

Como temos um algoritmo para obter a pontuação de um alinhamento ótimo com inversões não sobrepostas em tempo $O(n^3)$ e espaço $O(n^2)$, e temos dois algoritmos para obter a pontuação de um alinhamento ótimo com duplicações (alinhamento com duplicações e alinhamento com duplicações em *tandem*) também em tempo $O(n^3)$ e espaço $O(n^2)$, pretendemos desenvolver um algoritmo que obtenha a pontuação de um alinhamento ótimo com inversões não sobrepostas e duplicações também em tempo $O(n^3)$ e espaço $O(n^2)$.

Talvez algumas hipóteses simplificadoras podem ser adotadas, como a não sobreposição de inversões e duplicações.

7.2 Implementação e testes para os algoritmos de alinhamento com duplicações

Um trabalho a ser feito é a implementação dos algoritmos apresentados para a obtenção da pontuação de alinhamentos ótimos com duplicações e com duplicações em *tandem*, assim como testes em dados reais e comparações com outras ferramentas de alinhamento, a fim de saber a relevância da consideração das duplicações em um alinhamento.

Acreditamos que estes testes sejam necessários para estabelecermos critérios para os pesos das operações de edição pontuais (inserção, remoção e substituição) nos intervalos de repetição, assim como para a operação de edição de duplicação (e excisão). Esperamos que com um sistema de pontuação com pesos bem ajustados, o problema da ocorrência dos ciclos num grafo de duplicação pode ser minimizado em dados reais.

7.3 Pesos para abertura de *gaps*

Uma das características que não são contempladas no processo de obtenção do peso do alinhamento com inversões não sobrepostas de duas seqüências utilizando a estrutura de grafos de edição, e que são muito utilizadas nos sistemas de pontuação para alinhamento de seqüências biológicas, é a pontuação fixa para a abertura de *gaps*.

Um *gap* corresponde a uma operação de inserção ou remoção. O peso para a abertura de *gaps* corresponde a um peso fixo $\phi_{gapOpen}$, para cada seqüência consecutiva de remoções e de inserções, que é adicionado ao peso do alinhamento, ou seja, dado um caminho P num grafo de edição G, o peso w_p de P é dado pela soma dos pesos das arestas que formam P mais $(k_r+k_i)\phi_{gapOpen}$, onde k_r é a quantidade de seqüências de arestas consecutivas de remoção e k_i é a quantidade de seqüências de arestas consecutivas de inserção no caminho P.

Um trabalho a ser feito é tentar adaptar os algoritmos para obtenção da pontuação de um alinhamento ótimo (ou criar novos algoritmos) que considerem a possibilidade de pesos na abertura de *gaps*.

Para os algoritmo que obtêm a pontuação do alinhamento ótimo de um alinhamento com duplicações (ou duplicaçõesem *tandem*), apesar de não considerarmos pesos para a abertura de *gaps*, temos idéia de como, com pequenas

modificações nos algoritmos, introduzir esta nova característica. Um trabalho a ser feito é formalizar estas idéias e comprovar a sua corretude.

7.4 Função ω_{inv} dependente do comprimento da inversão

Em 2002 Pinter e Skiena [51] desenvolveram um trabalho onde o peso de uma reversão é dependente do comprimento das seqüências revertidas. As seqüências consideradas neste trabalho são seqüências de genes. A motivação para isto vem da biologia, onde muitas vezes se verifica que pequenos rearranjos são mais comuns, ou seja, é muito mais comum pequenas regiões sofrerem rearranjos do que grandes regiões.

Em 2004 Benderet al. [10] consideraram o caso onde o peso da reversão é generalizado pela função $f(l) = l^{\alpha}$, onde l é o comprimento da reversão e α é uma constante.

Para o problema de alinhamento de duas seqüências com inversões não sobrepostas, podemos querer considerar que a penalidade ω_{inv} de uma operação de inversão tenha alguma relação com o comprimento da inversão, ou das seqüências invertidas.

Pode ser interessante estudar a viabilidade da adaptação dos algoritmos 7 e 9 para considerar este novo conceito de pontuação das inversões.

Vale a pena lembrar que nos algoritmos para alinhamento com duplicações já consideramos o tamanho da repetição na pontuação de uma operação de edição de duplicação e excisão.

7.5 Inversão com um nível de sobreposição

Para o problema de encontrar o peso de um alinhamento com inversões, assumimos a simplificação da não sobreposição das inversões, ou seja, se uma região $s[i' \,..\, i]$ é invertida para ser melhor alinhada com uma região $t[j' \,..\, j]$ então nenhuma outra inversão será considerada nas regiões $s[i' \,..\, i]$ e $t[j' \,..\, j]$. Com esta simplificação qualquer letra da seqüência s ou da seqüência t pode estar contida em uma única operação de inversão.

Quando dizemos que pode haver até um nível de sobreposição, estamos querendo dizer que qualquer letra da seqüência s ou da seqüência t pode estar contida em até duas regiões que sofreram inversões.

Esta é uma outra visão para o problema do alinhamento com inversões. Esta nova visão foi motivada por uma teoria desenvolvida por Ross et al. [53] que diz que os cromossomos X e Y do ser humano podem diferir em 4 grandes inversões, além dos outros eventos menores de mutação do DNA (inclusive inversões). Assim, necessitaríamos analisar as seqüências considerando que algumas regiões podem ser invertidas duas vezes (uma pequena inversão e uma grande inversão).

7.6 Rearranjos nas repetições

Quando fazemos um alinhamento com duplicações, consideramos que o alinhamento das repetições com a seqüência original é uma alinhamento com somente operações de edição pontuais (inserção, remoção e substituição). Acreditamos que seria interessante obter a pontuação de um intervalo de repetição utilizando outras operações de edição, tais como, as duplicações e as inversões.

Para tanto, um trabalho a ser feito é a alteração dos algoritmos para a obtenção de pontuação de um alinhamento ótimo com duplicações, ou a criação de algoritmos totalmente novos, que considerem estes rearranjos nos alinhamentos das repetições dos intervalos de repetição.

7.7 Diminuir a memória utilizada pelos algoritmos

Os algoritmos para a obtenção da pontuação de alinhamentos ótimos com inversões não sobrepostas ou duplicações necessitam ambos de espaço de memória $O(n^2)$.

Um trabalho a ser feito é tentar desenvolver algoritmos que resolvam os mesmos problemas com a mesma complexidade no tempo de execução, mas uma melhor complexidade na utilização de memória.

7.8 Análise e comparação de tempos de execução dos algoritmos

Um trabalho a ser feito é verificar na prática as diferenças de tempo entre os algoritmos 7 e 9, ou seja, implementar e executar os algoritmos contra seqüências reais de dados e analisar os tempos de execução.

Na realidade este trabalho já começou a ser feito, pois uma implementação dos algoritmos 7 ($O(n^3 \log n)$) e 9 ($O(n^3)$) já foi feita e alguns testes também, porém é necessário uma análise mais criteriosa do código implementado, assim como a obtenção de tempos de execução estatisticamente mais confiáveis.

Além destes algoritmos, também já foi implementado para fins de análise comparativa, os algoritmos descritos em [21], que têm tempo de execução $O(n^4)$ e $O(n^4 \log^2 n/M^2)$, onde $M = (n^2/\#\ arestas)$ é o fator de esparsidade. Em alguns testes que realizamos M estava na ordem de grandeza de 10^{-4}.

Verificamos, em testes preliminares que realizamos, que o algoritmo $O(n^3)$ foi mais rápido, como já era esperado, que os algoritmos $O(n^4)$ e $O(n^3 \log n)$.

7.9 Esparsidade

Se um sistema de pontuação adota o valor zero para as operações de inserção, remoção e substituição com grau de similaridade abaixo de um certo limite, podemos obter um grafo de edição G de s e t com poucas arestas diagonais com valores positivos. Se isto ocorre, o algoritmo não precisa verificar todos os vértices e todas as arestas de G para obter o peso de um caminho ótimo de $(0, 0)$ e (n, m). Quanto menos arestas com peso positivo o grafo de edição tiver, mais esparso será o grafo.

Já existem algoritmos, como em [21], [42] e [27], que se aproveitam da esparsidade do grafo de edição para obter um caminho ótimo no grafo.

O aproveitamento desta esparsidade mostrou-se bem útil em alguns testes realizados com uma implementação do algoritmo esparso para a obtenção da pontuação de um alinhamento ótimo com inversões não sobrepostas, descrito em [21]. Para alguns testes realizados, o tempo de execução foi centenas de vezes mais rápido do que os algoritmos que não se aproveitavam desta esparsidade.

Um trabalho futuro, é tentar adaptar os algoritmos para obter um alinhamento ótimo com duplicações, com duplicações em *tandem* ou com inversões não sobrepostas a utilizarem a esparsidade do grafo de edição.

Referências Bibliográficas

[1] A novel gene containing a trinucleotide repeat that is expanded and unstable on Huntington's disease chromosomes. The Huntington's Disease Collaborative Research Group. *Cell*, 72(6):971–983, Mar 1993.

[2] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(2):195–208, 1987.

[3] Alok Aggarwal and James K. Park. Notes on searching in multidimensional monotone arrays. In *Proc. 29th Symp. Foundations of Computer Science*, pages 497–512. IEEE, 1988.

[4] Carlos E. R. Alves, Alair Pereira do Lago, and Augusto F. Vellozo. Alignment with non-overlapping inversions in $O(n^3 \log n)$-time. *Electronic Notes in Discrete Mathematics*, 19:365–371, Jun 2005.

[5] Carlos Eduardo Rodrigues Alves. *Algoritmos paralelos de granularidade grossa para problemas de alinhamento de cadeias*. PhD thesis, IME-USP, 2002.

[6] Alberto Apostolico, Mikhail J. Atallah, Lawrence L. Larmore, and Scott McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM J. Comput.*, 19(5):968–988, 1990.

[7] Armour, Anttinen, May, Vega, Sajantila, Kidd, Kidd, Bertranpetit, Pääbo, and Jeffreys. Minisatellite diversity supports a recent African origin for modern humans. *Nat Genet*, 13(2):154–160, Jun 1996.

[8] David A. Bader, Bernard M. E. Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutations

with an experimental study. In *Algorithms and data structures (Providence, RI, 2001)*, volume 2125 of *Lecture Notes in Comput. Sci.*, pages 365–376. Springer, Berlin, 2001.

[9] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10(7):950–958, Jul 2000.

[10] Michael A. Bender, Dongdong Ge, Simai He, Haodong Hu, Ron Y. Pinter, Steven Skiena, and Firas 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.

[11] Gary Benson. Sequence alignment with tandem duplication. In *RECOMB '97: Proceedings of the first annual international conference on Computational molecular biology*, pages 27–36, New York, NY, USA, 1997. ACM Press.

[12] Burkard, Klinz, and Rudolf. Perspectives of monge properties in optimization. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 70, 1996.

[13] V. Campuzano, L. Montermini, M. D. Molto, L. Pianese, M. Cossee, F. Cavalcanti, E. Monros, F. Rodius, F. Duclos, A. Monticelli, F. Zara, J. Canizares, H. Koutnikova, S. I. Bidichandani, C. Gellera, A. Brice, P. Trouillas, G. de Michele, A. Filla, R. de Frutos, F. Palau, P. I. Patel, S. di Donato, J.-L. Mandel, S. Cocozza, M. Koenig, and M. Pandolfo. Friedreich's Ataxia: Autosomal Recessive Disease Caused by an Intronic GAA Triplet Repeat Expansion. *Science*, 271:1423–1427, March 1996.

[14] Alberto Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Math.*, 12(1):91–110 (electronic), 1999.

[15] Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, Stefano Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(4):302–315, 2005.

[16] David A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998)*, pages 244–252, New York, 1998. ACM.

[17] N. C. Comfort. From controlling elements to transposons: Barbara mcclintock and the nobel prize. *Endeavour*, 25(3):127–130, September 2001.

[18] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[19] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, Jun 1999.

[20] Alair Pereira do Lago, Ilya Muchnik, and Casimir Kulikowski. An $O(n^4)$ algorithm for alignment with non-overlapping inversions. In *Second Brazilian Workshop on Bioinformatics, WOB 2003*, Macaé, RJ, Brazil, 2003. http://www.ime.usp.br/∼alair/wob03.pdf.

[21] Alair Pereira do Lago, Ilya Muchnik, and Casimir Kulikowski. A sparse dynamic programming algorithm for alignment with non-overlapping inversions. *Theor. Inform. Appl.*, 39(1):175–189, 2005.

[22] I. Dubchak, M. Brudno, G. G. Loots, L. Pachter, C. Mayor, E. M. Rubin, and K. A. Frazer. Active conservation of noncoding sequences revealed by three-way species comparisons. *Genome Research*, 10(9):1304–1306, Sep 2000.

[23] Edwards, Hammond, Jin, Caskey, and Chakraborty. Genetic variation at five trimeric and tetrameric tandem repeat loci in four human population groups. *Genomics*, 12(2):241–253, Feb 1992.

[24] Nadia El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *Combinatorial pattern matching (Montreal, QC, 2000)*, volume 1848 of *Lecture Notes in Comput. Sci.*, pages 222–234. Springer, Berlin, 2000.

[25] Nadia El-Mabrouk. Sorting signed permutations by reversals and insertions/deletions of contiguous segments. *J. Discrete Algorithms*, 1(1):105–121, 2000.

[26] Nadia El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *J. Comput. System Sci.*, 65(3):442–464, 2002. Special issue on computational biology 2002.

[27] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming ii: convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992.

[28] Feuk, MacDonald, Tang, Carson, Li, Rao, Khaja, and Scherer. Discovery of human inversion polymorphisms by comparative analysis of human and chimpanzee DNA sequence assemblies. *PLoS Genet*, 1(4):e56, Oct 2005.

[29] Y. H. Fu, A. Pizzuti, R. G. Fenwick, Jr., J. King, S. Rajnarayan, P. W. Dunne, J. Dubel, G. A. Nasser, T. Ashizawa, P. de Jong, B. Wieringa, R. Korneluk, M. B. Perryman, H. F. Epstein, and C. T. Caskey. An Unstable Triplet Repeat in a Gene Related to Myotonic Muscular Dystrophy. *Science*, 255:1256–1258, March 1992.

[30] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 246–251, New York, NY, USA, 1983. ACM Press.

[31] Yong Gao, Junfeng Wu, Robert Niewiadomski1, Yang Wang, Zhi-Zhong Chen, and Guohui 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-Verlag, 2003.

[32] Robert Giegerich and David Wheeler. Pairwise sequence alignment.

[33] Anthony J. F. Griffiths, William M. Gelbart, Jeffrey H. Miller, and Richard C. Lewontin. *Modern Genetic Analysis*. W. H. Freeman and Company, 1999.

[34] H Hamada, M Seidman, B H Howard, and C M Gorman. Enhanced gene expression by the poly(dT-dG).poly(dC-dA) sequence. *Mol. Cell. Biol.*, 4(12):2622–2630, 1984.

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

[36] Hellman, Steen, Sundvall, and Pettersson. A rapidly evolving region in the immunoglobulin heavy chain loci of rat and mouse: postulated role of (dC-dA)n.(dG-dT)n sequences. *Gene*, 68(1):93–9100, Aug 1988.

[37] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

[38] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (New Orleans, LA, 1997)*, pages 344–351, New York, 1997. ACM.

[39] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Comput.*, 29(3):880–892 (electronic), 2000.

[40] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1-2):180–210, 1995.

[41] E. V. Koonin. The emerging paradigm and open problems in comparative genomics. *Bioinformatics*, 15(4):265–266, Apr 1999. Editorial.

[42] Gad M. Landau, Baruch Schieber, and Michal Ziv-Ukelson. Sparse lcs common substring alignment. *Inf. Process. Lett.*, 88(6):259–270, 2003.

[43] Gad M. Landau and Michal Ziv-Ukelson. On the common substring alignment problem. *J. Algorithms*, 41(2):338–359, 2001.

[44] Q Lu, L L Wallrath, H Granok, and S C Elgin. (CT)n (GA)n repeats and heat shock elements have distinct roles in chromatin structure and

transcriptional activation of the Drosophila hsp26 gene. *Mol. Cell. Biol.*, 13(5):2802–2814, 1993.

[45] C. Mayor, M. Brudno, J. R. Schwartz, A. Poliakov, E. M. Rubin, K. A. Frazer, L. S. Pachter, and I. Dubchak. VISTA : visualizing global DNA sequence alignments of arbitrary length. *Bioinformatics*, 16(11):1046–1047, Nov 2000.

[46] W. Messier, S. H. Li, and C. B. Stewart. The birth of microsatellites. *Nature*, 381:483, Jun 1996.

[47] G. Moore, K. M. Devos, Z. Wang, and M. D. Gale. Cereal genome evolution. Grasses, line up and form a circle. *Current Biology*, 5(7):737–739, Jul 1995. Review.

[48] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, Mar 1970.

[49] Pardue, Lowenhaupt, Rich, and Nordheim. (dC-dA)n.(dG-dT)n sequences have evolutionarily conserved chromosomal locations in Drosophila with implications for roles in chromosome structure and function. *EMBO J*, 6(6):1781–1789, Jun 1987.

[50] Pavel A. Pevzner. *Computational molecular biology: An Algorithmic Approach*. MIT Press, 2000.

[51] R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Inform*, 13:103–111, 2002.

[52] R.I. Richards, K. Holman, S. Yu, and G.R. Sutherland. Fragile X syndrome unstable element, p(CCG)n, and other simple tandem repeat sequences are binding sites for specific nuclear proteins. *Hum. Mol. Genet.*, 2(9):1429–1435, 1993.

[53] M. T. Ross, D. V. Grafham, A. J. Coffey, S. Scherer, K. McLay, D. Muzny, M. Platzer, G. R. Howell, C. Burrows, C. P. Bird, A. Frankish, F. L. Lovell, K. L. Howe, J. L. Ashurst, R. S. Fulton, R. Sudbrak, G. Wen, M. C. Jones, M. E. Hurles, T. D. Andrews, C. E. Scott, S. Searle, J. Ramser, A. Whittaker, R. Deadman, N. P. Carter, S. E. Hunt, R. Chen, A. Cree, P. Gunaratne, P. Havlak, A. Hodgson, M. L.

Metzker, S. Richards, G. Scott, D. Steffen, E. Sodergren, D. A. Wheeler, K. C. Worley, R. Ainscough, K. D. Ambrose, M. A. Ansari-Lari, S. Aradhya, R. I. Ashwell, A. K. Babbage, C. L. Bagguley, A. Ballabio, R. Banerjee, G. E. Barker, K. F. Barlow, I. P. Barrett, K. N. Bates, D. M. Beare, H. Beasley, O. Beasley, A. Beck, G. Bethel, K. Blechschmidt, N. Brady, S. Bray-Allen, A. M. Bridgeman, A. J. Brown, M. J. Brown, D. Bonnin, E. A. Bruford, C. Buhay, P. Burch, D. Burford, J. Burgess, W. Burrill, J. Burton, J. M. Bye, C. Carder, L. Carrel, J. Chako, J. C. Chapman, D. Chavez, E. Chen, G. Chen, Y. Chen, Z. Chen, C. Chinault, A. Ciccodicola, S. Y. Clark, G. Clarke, C. M. Clee, S. Clegg, K. Clerc-Blankenburg, K. Clifford, V. Cobley, C. G. Cole, J. S. Conquer, N. Corby, R. E. Connor, R. David, J. Davies, C. Davis, J. Davis, O. Delgado, D. Deshazo, P. Dhami, Y. Ding, H. Dinh, S. Dodsworth, H. Draper, S. Dugan-Rocha, A. Dunham, M. Dunn, K. J. Durbin, I. Dutta, T. Eades, M. Ellwood, A. Emery-Cohen, H. Errington, K. L. Evans, L. Faulkner, F. Francis, J. Frankland, A. E. Fraser, P. Galgoczy, J. Gilbert, R. Gill, G. Glockner, S. G. Gregory, S. Gribble, C. Griffiths, R. Grocock, Y. Gu, R. Gwilliam, C. Hamilton, E. A. Hart, A. Hawes, P. D. Heath, K. Heitmann, S. Hennig, J. Hernandez, B. Hinzmann, S. Ho, M. Hoffs, P. J. Howden, E. J. Huckle, J. Hume, P. J. Hunt, A. R. Hunt, J. Isherwood, L. Jacob, D. Johnson, S. Jones, P. J. de Jong, S. S. Joseph, S. Keenan, S. Kelly, J. K. Kershaw, Z. Khan, P. Kioschis, S. Klages, A. J. Knights, A. Kosiura, C. Kovar-Smith, G. K. Laird, C. Langford, S. Lawlor, M. Leversha, L. Lewis, W. Liu, C. Lloyd, D. M. Lloyd, H. Loulseged, J. E. Loveland, J. D. Lovell, R. Lozado, J. Lu, R. Lyne, J. Ma, M. Maheshwari, L. H. Matthews, J. McDowall, S. McLaren, A. McMurray, P. Meidl, T. Meitinger, S. Milne, G. Miner, S. L. Mistry, M. Morgan, S. Morris, I. Muller, J. C. Mullikin, N. Nguyen, G. Nordsiek, G. Nyakatura, C. N. O'Dell, G. Okwuonu, S. Palmer, R. Pandian, D. Parker, J. Parrish, S. Pasternak, D. Patel, A. V. Pearce, D. M. Pearson, S. E. Pelan, L. Perez, K. M. Porter, Y. Ramsey, K. Reichwald, S. Rhodes, K. A. Ridler, D. Schlessinger, M. G. Schueler, H. K. Sehra, C. Shaw-Smith, H. Shen, E. M. Sheridan, R. Shownkeen, C. D. Skuce, M. L. Smith, E. C. Sotheran, H. E. Steingruber, C. A. Steward, R. Storey, R. M. Swann, D. Swarbreck, P. E. Tabor, S. Taudien, T. Taylor, B. Teague, K. Thomas, A. Thorpe, K. Timms, A. Tracey, S. Trevanion, A. C. Tromans, M. d Urso, D. Verduzco, D. Villasana, L. Waldron, M. Wall, Q. Wang, J. Warren, G. L. Warry, X. Wei,

A. West, S. L. Whitehead, M. N. Whiteley, J. E. Wilkinson, D. L. Willey, G. Williams, L. Williams, A. Williamson, H. Williamson, L. Wilming, R. L. Woodmansey, P. W. Wray, J. Yen, J. Zhang, J. Zhou, H. Zoghbi, S. Zorilla, D. Buck, R. Reinhardt, A. Poustka, A. Rosenthal, H. Lehrach, A. Meindl, P. J. Minx, L. W. Hillier, H. F. Willard, R. K. Wilson, R. H. Waterston, C. M. Rice, M. Vaudin, A. Coulson, D. L. Nelson, G. Weinstock, J. E. Sulston, R. Durbin, T. Hubbard, R. A. Gibbs, S. Beck, J. Rogers, and D. R. Bentley. The dna sequence of the human x chromosome. *Nature*, 434(7031):325–337, March 17 2005.

[54] Jeanette P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM J. Comput.*, 27(4):972–992 (electronic), 1998.

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

[56] S. Schwartz, Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller. PipMaker–a web server for aligning two genomic DNA sequences. *Genome Research*, 10(4):577–586, Apr 2000.

[57] João Setubal and João Meidanis. *Introduction to computational molecular biology*. PWS Publishing Company, 1997.

[58] Smith and Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar 1981.

[59] Eric Tannier and Marie-France Sagot. Sorting by reversals in subquadratic time. In *Combinatorial pattern matching*, volume 3109, pages 1–13, 2004. CPM 2004.

[60] T. A. Tatusova and T. L. Madden. BLAST 2 Sequences, a new tool for comparing protein and nucleotide sequences. *FEMS Microbiology Letters*, 174(2):247–250, May 1999.

[61] S. A. Tishkoff, E. Dietzsch, W. Speed, A. J. Pakstis, J. R. Kidd, K. Cheung, B. Bonne-Tamir, A. S. Santachiara-Benerecetti, P. Moral, M. Krings, S. Paabo, E. Watson, N. Risch, T. Jenkins, and K. K. Kidd.

Global Patterns of Linkage Disequilibrium at the CD4 Locus and Modern Human Origins. *Science*, 271:1380–1387, March 1996.

[62] Augusto F. Vellozo, Carlos E. R. Alves, and Alair Pereira do Lago. Alignment with non-overlapping inversions in $o(n^3)$-time. In *6th Workshop on Algorithms in Bioinformatics*. Springer, 2006. Lecture Notes in Bioinformatics 4175.

[63] Verkerk, Pieretti, Sutcliffe, Fu, Kuhl, Pizzuti, Reiner, Richards, Victoria, and Zhang. Identification of a gene (FMR-1) containing a CGG repeat coincident with a breakpoint cluster region exhibiting length variation in fragile X syndrome. *Cell*, 65(5):905–914, May 1991.

[64] R. Wagner. On the complexity of the extended string-to-string correction problem. In *Seventh ACM Symposium on the Theory of Computation*. Association for Computing Machinery, 1975.

[65] Waterman and Eggert. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of Molecular Biology*, 197(4):723–728, Oct 1987.

[66] Weber and May. Abundant class of human DNA polymorphisms which can be typed using the polymerase chain reaction. *Am J Hum Genet*, 44(3):388–396, Mar 1989.

[67] H.A. Yee, A.K.C. Wong, J.H. van de Sande, and J.B. Rattner. Identification of novel single-stranded d(TC)n binding proteins in several mammalian species. *Nucl. Acids Res.*, 19(4):949–953, 1991.