Algoritmos para Análises de Seqüências

Melissa Lemos
Departamento de Informática, PUC-Rio
e-mail: melissa@inf.puc-rio.br

Marco Antonio Casanova
Departamento de Informática, PUC-Rio
e-mail: casanova@inf.puc-rio.br

PUC-RioInf.MCC05/00 Janeiro, 2000

Abstract

Sequence comparison is one of the most important primitive operations in Computational Biology. It is the basis for more elaborated manipulations. The algorithm for comparing two sequences is based on dynamic programming. Both the basic algorithm and its extensions have quadratic time complexity, which makes them unsuitable for database searches. For this reason, families of heuristic algorithms were developed to improve the response time of database searches. The FAST and BLAST are the best known families. This works describes the basic algorithm, its extensions and the FAST and BLAST families.

Keywords: Computational Biology, Algorithms for sequence comparison, FAST, BLAST.

Resumo

A comparação de seqüências é uma das operações primitivas mais importantes na área de biologia computacional. Ela serve de base para outras manipulações mais elaboradas. O algoritmo básico de comparação de duas seqüências é baseado no método de programação dinâmica. Tanto o algoritmo básico quanto suas extensões possuem complexidade de tempo quadrática. Isto faz com que eles sejam inviáveis para consultas em bancos de dados de seqüências. Por este motivo surgiram famílias de algoritmos baseados em heurísticas que trouxeram uma grande melhora nos tempos de respostas das consultas em bancos de dados. As famílias mais conhecidas são FAST e BLAST. Este trabalho apresenta o algoritmo básico, suas extensões, e as famílias FAST e BLAST.

Palavras-chave: Biologia Computacional, Algoritmos de Comparação de Seqüências, FAST, BLAST.

Índice

Figuras

1. Introdução

O objetivo principal deste trabalho é o estudo de algoritmos que analisam biosseqüências em bancos de dados.

Inicialmente será apresentado o algoritmo básico para comparação de biosseqüências, que é uma das operações primitivas mais importantes na área da biologia computacional e que serve de base para muitas outras manipulações mais elaboradas. Este algoritmo é baseado no método de programação dinâmica. Algumas extensões dele foram feitas para que o algoritmo se tornasse mais adequado para determinadas tarefas. Elas também serão apresentadas.

A complexidade quadrática do algoritmo básico e de suas extensões faz com que eles se tornem inviáveis em determinadas aplicações. Um exemplo importante deste tipo de aplicação é a busca em bases de dados moleculares. Algumas destas bases possuem milhares de seqüências. O problema típico desta aplicação é a comparação de uma dada seqüência nova com todas as depositadas na base de dados. Isto significa que milhares de comparações precisam ser feitas [Meidanis, 94].

Por este motivo, vários métodos alternativos mais rápidos surgiram. Geralmente eles são baseados em heurísticas. Alguns deles são baseados em uma janela da seqüência s, que é simplesmente um fator de s. A idéia destes métodos é que, se duas seqüências se parecem, então elas terão muitas janelas em comum.

Árvore de sufixos [Weiner,73][McCreight, 76] e vetores de sufixos [Manber, 90] são exemplos de estruturas que podem ser usadas para obter janelas comuns de modo eficiente. A árvore de sufixos guarda em uma árvore todos os sufixos da seqüência, enquanto o vetor de sufixos guarda todos os sufixos em ordem alfabética. Desta forma fica mais fácil responder a pergunta: "A janela w encontra-se na seqüência?".

Uma maneira eficiente de obter as janelas comuns é construir a estrutura relativa a uma das seqüências e percorrer a outra tentando localizar todas as suas janelas na estrutura. Cada janela localizada é uma janela comum. Outra possibilidade é usar técnicas de *hashing* para obter o número de janelas comuns [Meidanis, 94].

Como as técnicas de seqüenciamento de nucleotídeos e proteínas têm melhorado muito e conseqüentemente os repositórios destes dados estão crescendo, a aplicação de busca em bases de dados moleculares tem se tornado a cada dia que passa mais importante. Por

este motivo algumas seções deste trabalho estão dedicadas especialmente para as famílias de algoritmos chamadas FAST e BLAST. Os programas destas famílias estão baseados em heurísticas que trouxeram uma grande melhora nos tempos de respostas das buscas em bases de biosseqüências.

Referências adicionais sobre Bioinformática podem ser encontradas em http://www.inf.puc-rio.br/~melissa/.

O trabalho está dividido da seguinte forma. A seção 2 introduz conhecimentos básicos de biologia, apresenta as definições que serão usadas ao longo do texto e a motivação para a comparação de biosseqüências. A seção 3 descreve o algoritmo básico de comparação de biosseqüências e suas extensões. A seção 4 discute os algoritmos que fazem comparação de biosseqüências em bancos de dados, focalizando as famílias FAST e BLAST. Finalmente, a seção 5 contém as conclusões.

2. Preliminares

Esta seção está dedicada especialmente a conceitos preliminares que serão usados ao longo do texto.

2.1. Introdução à Biologia Celular e Molecular

Esta seção foi baseada em [Roberts, 85] e [Helio, 99].

2.1.1. A Célula: Organização Estrutural

O estudo do mundo vivo mostra que a evolução produziu uma imensa variedade de formas. Existem em torno de quatro milhões de espécies diferentes de bactérias, protozoários, vegetais e animais, que diferem em sua morfologia, função e comportamento. Entretanto sabemos agora que, quando os organismos vivos são estudados a nível celular e molecular, observa-se um plano único principal de organização. O objetivo da biologia celular e molecular é precisamente este plano unificado de organização – isto é, a análise das células e moléculas que constituem as unidades estruturais de todas as formas de vida.

Há muito tempo atrás observou-se que uma única célula poderia constituir um organismo inteiro, como no caso dos protozoários, ou ser uma das muitas, agrupadas e diferenciadas em tecidos e órgãos, para formar um organismo multicelular.

Assim sendo, a célula é a unidade estrutural e funcional básica dos organismos vivos, da mesma forma que o átomo é a unidade fundamental das estruturas químicas.

Células Procarióticas e Eucarióticas

A vida manifesta-se em milhões de diferentes espécies, que possuem sua própria morfologia e informação genética específica. As espécies podem ser reunidas em grupos progressivamente mais abrangentes – gêneros, ordens, famílias – até o nível dos reinos clássicos, vegetal e animal. Um dos esquemas de classificação, o de Whittaker, postula a divisão em cinco **reinos – Monera, Protista, Fungi, Plantae e Animalia**, com as suas correspondentes subdivisões.

Esta aparente complexidade simplifica-se com o exame das formas vivas a nível celular. As células são identificadas como pertencentes a dois grupos: **procarióticas e eucarióticas**. Somente os seres pertencentes ao reino *Monera* (i.e. bactérias, algas azuis – cianofíceas) possuem céluas procarióticas, enquanto que todos os outros reinos constituem-se de organismos formados por células eucarióticas. A principal diferença entre estes dois tipos celulares é a ausência de um envoltório nuclear nas células procarióticas. O cromossomo desta célula ocupa um espaço denominado nucleóide, estando em contato direto com o protoplasma. As células eucarióticas possuem um núcleo verdadeiro com um envoltório nuclear elaborado, através do qual ocorrem trocas entre o núcleo e o citoplasma.

Veja a seguir as figuras das células animal e vegetal, e note a complexidade dos eucariontes.

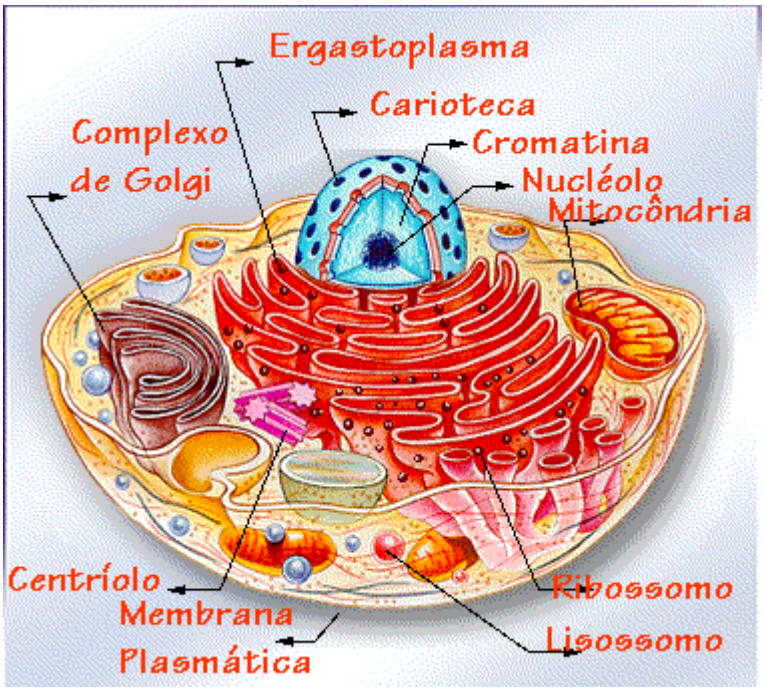


Figura 1. Célula animal.

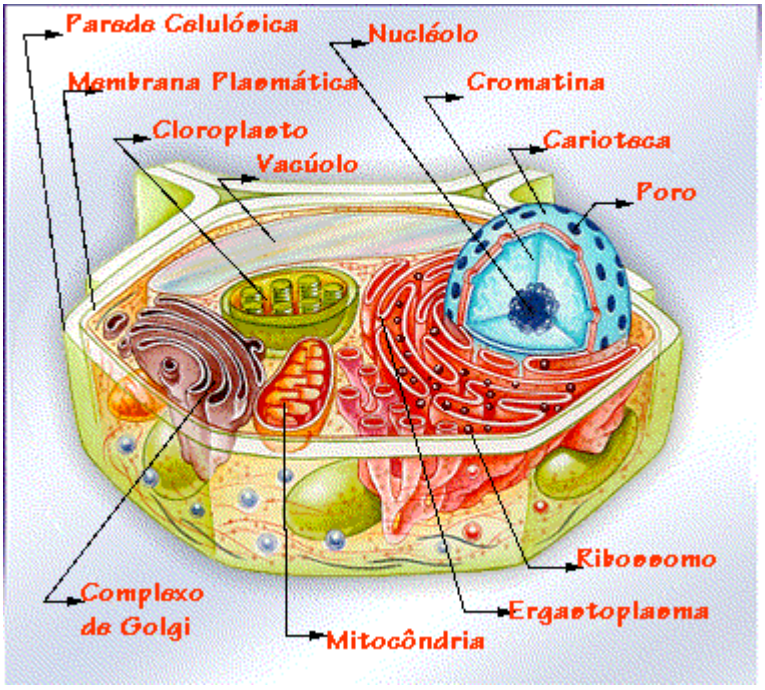


Figura 2. Célula vegetal.

2.1.2. A Célula: Organização Molecular

A estrutura celular visível aos microscópios óptico e eletrônico é resultante de um arranjo de moléculas numa ordem bastante precisa. Apesar de haver muito ainda a ser aprendido,

começaram a surgir os princípios gerais da organização molecular de algumas estruturas celulares, como menbranas, ribossomos, cromossomos, mitocôndrias e cloroplastos. A biologia celular não pode ser separada da molecular, pois da mesma maneira que as células são os "tijolos" de tecidos e organismos, as moléculas são os "tijolos" das células.

A organização molecular dos seres vivos está baseada na química do carbono. Proteínas, carboidratos, lipídios, etc., são diferentes macromoléculas que contém um grande número de átomos de carbono arranjados de forma distinta.

Numerosas estruturas celulares são constituídas por moléculas bastante grandes denominadas **macromoléculas** ou **polímeros**, compostas por unidades repetidas, chamadas **monômeros**.

Existem três exemplos importantes de polímeros nos organismos vivos. São eles:

- **Ácidos nucléicos**, que resultam da repetição de quatro diferentes unidades denominadas **nucleotídeos**. A seqüência linear de quatro nucleotídeos na molécula de DNA é a fonte básica da informação genética.
- **Polissacarídeos** podem ser polímeros da glicose, formando amido, celulose ou glicogênio, ou podem também envolver a repetição de outras moléculas para formar polissacarídeos mais complexos.
- **Proteínas** ou **polipeptídeos** são compostos por aproximadamente 20 aminoácidos, presentes em diversas proporções, unidos por *ligações peptídicas*. A ordem em que estes 20 monômeros podem se unir dá origem a um número astronômico de combinações em diferentes moléculas protéicas, determinando não só sua especificidade, mas também sua atividade biológica. Como exemplo deste número astronômico imagine uma proteína de 100 aminoácidos, que é até pequena. Suponha que é necessário obter todas as combinações possíveis de aminoácidos para tal proteína. Imagine agora que você tem 1 bilhão de galaxias, cada uma com 1 bilhão de planetas, cada um com 1 bilhão de computadores, cada um dos quais produz 1 bilhão de combinações por segundo. Calcule o tempo gasto para escrever todas as possíveis combinações. A resposta é a seguinte. Se considerar a idade do universo como sendo de dez bilhões de anos, deve levar mais ou menos 10^{48} vezes a idade do universo.

Ácidos Nucléicos

Os ácidos nucléicos são macromoléculas de suma importância biológica. Todos os organismos vivos contêm ácidos nucléicos na forma de ácido desoxirribonucléico (DNA) e ácido ribonucléico (RNA).

O DNA é o principal armazenador da informação genética. Esta informação é copiada ou transcrita para moléculas de RNA, cujas as seqüências de nucleotídeos contém o "código" para a ordenação específica de aminoácidos. As proteínas são então sintetizadas num processo que envolve a tradução do RNA. Refere-se freqüentemente à série de eventos acima relacionada como o dogma central da biologia molecular; ela pode ser resumida na forma:

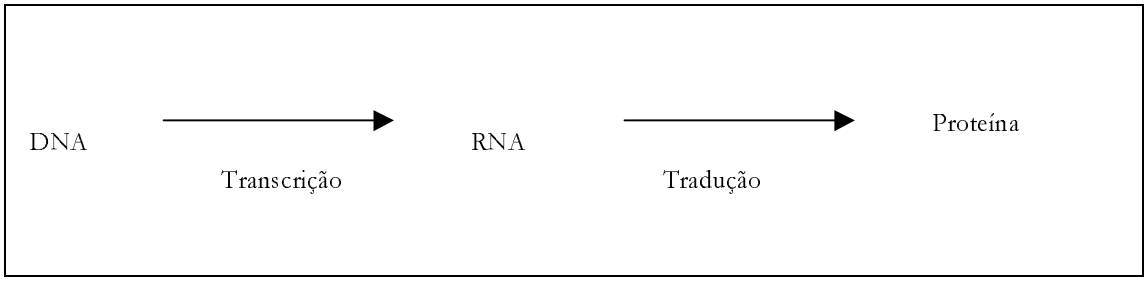


Figura 3. Processos transcrição e tradução.

Em células superiores, o DNA localiza-se principalmente no núcleo, dentro dos cromossomos. Uma pequena quantidade de DNA fica no citoplasma, contida nas mitocôndrias e cloroplastos. O RNA é encontrado tanto no núcleo, onde é sintetizado, quanto no citoplasma, onde tem lugar a síntese protéica.

Ácidos Nucléicos: uma Pentose, um Fosfato e quatro Bases

Os ácidos nucléicos são compostos por uma molécula de açúcar (pentose), bases nitrogenadas (purinas e piridiminas) e ácido fosfórico. Veja a figura a seguir.

.

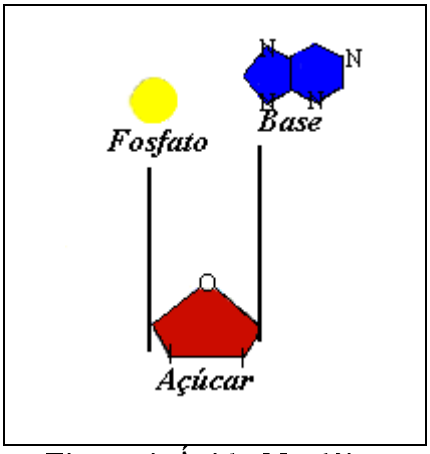


Figura 4. Ácido Nucléico

As pentoses são de dois tipos: ribose no RNA e desoxirribose no DNA. A única diferença do ponto de visto de composição da molécula entre estes dois açúcares é que a desoxirribose possui um átomo de oxigênio a menos.

As bases encontradas nos ácidos nucléicos são também de dois tipos: piridiminas e purinas. No DNA as piridiminas são timina (T) e citosina (C); as purinas são adenina (A) e guanina(G). O RNA contém uracila (U) no lugar de timina.

Existem duas diferenças principais entre o DNA e o RNA. O DNA possui uma molécula de desoxirribose e o RNA, uma de ribose; o DNA contém timina e o RNA, uracila.

	Ácido desoxirribonucléico	*Ácido ribonucléico*
Localização	Primariamente no núcleo, também nas mitocôndrias e cloroplastos	No citoplasma, nucléolo e cromossomos
Bases pirimidínicas	Citosina Timina	Citosina Uracila
Bases purínicas	Adenina Guanina	Adenina Guanina

Figura 5. Diferenças principais entre DNA eRNA.

Além de atuarem como as unidade dos ácidos nucléicos, os nucleotídeos são também importantes, pois armazenam e transferem energia química.

Composição Básica do DNA: A = T e G = C

O DNA está presente nos organismos vivos na forma de moléculas lineares de peso molecular extremamente elevado. A *E.coli*, por exemplo, possui uma molécula única circular de DNA que pesa em torno de 2,7 X 10^9 dáltons (dálton é uma unidade de massa equivalente a 1/16 da massa de um átomo de oxigênio, ou aproximadamente igual à de um átomo de hidrogênio) e tem um comprimento total de 1,4mm. Em organismos superiores a quantidade de DNA pode ser vários milhares de vezes maior; por exemplo, o DNA de uma única célula diplóide humana, se totalmente esticado, teria um comprimento total de 1,7m.

Toda a informação genética de um organismo vivo está armazenada em sua seqüência linear das quatro bases. Portanto, um alfabeto de quatro letras (A, T, C, G) deve codificar a estrutura primária (i.é., o número e a seqüência dos 20 aminoácidos) de todas as proteínas. Uma das mais extraordinárias descobertas da biologia molecular foi a elucidação deste código. Um prólogo desta descoberta, que tem ligação direta com o entendimento da estrutura do DNA, foi o achado de que existiam regularidades previsíveis no conteúdo das bases. Entre 1949 e 1953, Chargaff estudou detalhadamente a composição do DNA. Ele observou que, apesar da composição de bases variar de uma espécie para outra, a quantidade de adenina era igual à de timina (A = T) em todos os casos. Foi também notado que o número de bases de guanina e citosina era igual (G = C). Conseqüentemente, a quantidade total de purinas equivale à de piridiminas(i.é, A + G = C + T). Por outro lado, a razão AT/GC varia consideravelmente entre as espécies.

O DNA é uma Hélice Dupla

Após a descoberta da estrutura do DNA [Watson, 53] ficou explicado a regularidade de sua composição de bases e suas propriedades biológicas, particularmente sua duplicação na célula. A estrutura do DNA é mostrada na figura a seguir.

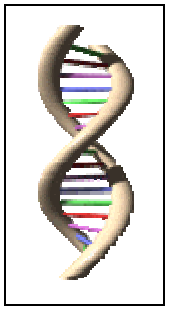


Figura 6. A dupla hélice do DNA.

Ele é composto por duas cadeias helicoidais de polinucleotídeos com giro para a direita, formando uma *hélice dupla* em torno de um mesmo eixo central. As duas fitas são antiparalelas, unidas por pontes de hidrogênio estabelecidas entre os pares de bases. Desde que existam uma distância fixa entre as duas moléculas de açúcar nas fitas opostas, somente certos pares de bases podem se encaixar na estrutura. Os únicos pares possíveis são o AT e o CG.

A *seqüência axial* de bases ao longo de uma cadeia de polinucleotídeo pode variar consideravelmente, porém na outra cadeia a seqüência deve ser complementar. Devido a esta propriedade, dada uma ordem de bases em uma cadeia, a outra é exatamente complementar. Durante a duplicação do DNA, as duas cadeias dissociam-se e cada uma age como um molde para a síntese da nova cadeia complementar. Assim sendo, são produzidas duas moléculas de DNA de cadeia dupla, possuindo exatamente a mesma constituição molecular.

Cada fita do DNA tem duas extremidades, chamadas de 3' e 5', numa alusão aos átomos de carbono que ficam livres no açúcar que compõem cada nucleotídeo. Há duas observações importantes neste contexto. A primeira é que a extremidade 3' de uma fita corresponde à extremidade 5' da outra (veja a figura a seguir). A segunda é que um A em uma fita corresponde a um T na fita oposta, e um C sempre corresponde a um G. É dito que A e T são bases complementares, assim como, C e G. Com isto, vemos que a seqüência de nucleotídeos numa das fitas determina completamente a molécula de DNA. É justamente esta propriedade que permite a auto-duplicação do DNA. A convenção adotada mundialmente para representar moléculas de DNA é escrever apenas umas das fitas na direção 5' → 3'.

Estrutura do RNA: classes e conformação

A estrutura primária do RNA é semelhante à do DNA, exceto pela substituição da ribose pela desoxirribose e da uracila pela timina. A composição de bases do RNA não segue as normas de Chargaff, pois as moléculas de RNA são compostas por uma única cadeia.

Existem três principais classes de ácido ribonucléico: o RNA mensageiro (mRNA), o RNA de transferência (tRNA) e o ribossômico (rRNA). Todos estão envolvidos na síntese protéica. O mRNA contém a informação genética para a seqüência de aminoácidos, o tRNA identifica e transporta as moléculas de aminoácidos até o

ribossomo, e o rRNA representa 50% da massa dos ribossomos, organelas que fornecem um suporte molecular para as reações químicas da montagem de um polipeptídeo.

Proteínas

As unidades constituintes das proteínas são os **aminoácidos**. Existem vinte tipos de aminoácidos. Eles estão apresentados na tabela seguinte com os códigos de números 1 a 20. Além dos códigos dos aminoácidos, existem mais três códigos usados pelos biólogos no seqüenciamento de proteínas, que também estão na tabela seguinte. O 21 e o 22 que identificam os pares de aminoácidos Asparagina/Ácido Aspartâmico (ou Ácido Aspártico) e Glutamina/Ácido Glutâmico, e o 23 que identifica todos os aminoácidos.

Código	Letra	Abreviação	Nome
1	A	Ala	Alanina
2	C	Cys	Cisteína
3	D	Asp	Ácido aspártico
4	E	Glu	Ácido glutâmico
5	F	Phe	Fenilalanina
6	G	Gly	Glicina
7	H	His	Histidina
8	I	Ile	Isoleucina
9	K	Lys	Lisina
10	L	Leu	Leucina
11	M	Met	Metionina
12	N	Asn	Asparagina
13	P	Pro	Prolina
14	Q	Gln	Glutamina
15	R	Arg	Arginina
16	S	Ser	Serina
17	T	Thr	Treonina

18	V	Val	Valina
19	W	Trp	Triptofano
20	Y	Tyr	Tirosina
21	Z	Glx	Asparagina/Ácido Aspartâmico
22	B	Asx	Glutamina/Ácido Glutâmico
23	X		Qualquer aminoácido

Figura 7. Códigos dos aminoácidos.

Proteínas Homólogas

Proteínas homólogas são aquelas que possuem relação filogenética comprovada. A evolução molecular segue alguns dos mesmos princípios da evolução dos seres vivos; golfinhos e tubarões tem mais ou menos o mesmo formato, entretanto são animais completamente diferentes (peixes X mamíferos). Da mesma forma, duas proteínas podem exercer a mesma função, e até terem uma certa semelhança, mas não terem relação nenhuma do ponto de vista de sua origem. Resumindo, duas proteínas homólogas são proteínas cuja relação de origem foi comprovada.

2.1.3. Biologia Molecular do Gene

O DNA transporta a informação genética de maneira codificada de célula a célula e dos pais para a progênie. Toda a informação necessária para a formação de um novo organismo está contida na seqüência linear das quatro bases, e a replicação fiel desta informação é assegurada pela estrutura de dupla cadeia do DNA onde o A pareia-se somente com o T e o G com o C.

O DNA não está livre dentro da célula, mas forma complexos com proteínas na estrutura denominada **cromatina**. No momento da divisão celular, a cromatina condensa-se na forma de **cromossomos**.

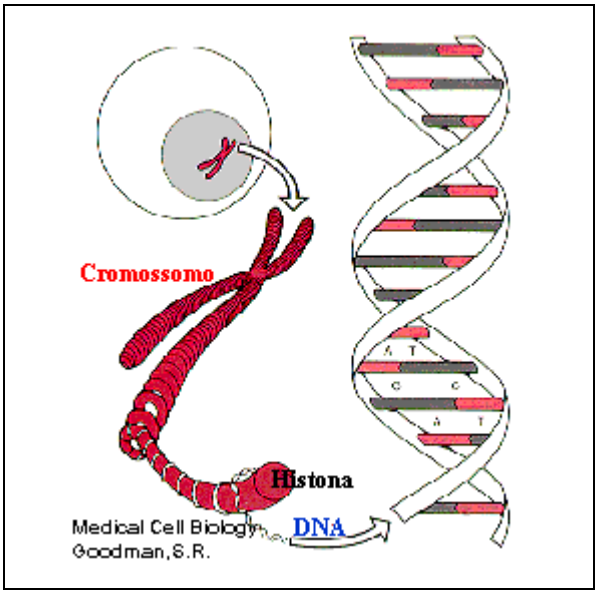


Figura 8. A célula e o cromossomo.

Os cromossomos são filamentos encontrados no interior do núcleo das células. Veja a figura anterior. Eles ocorrem normalmente em pares, têm diferentes tamanhos e formas e seu número é constante em cada espécie de ser vivo. As células humanas têm 46, divididos em 23 pares, com exceção das reprodutivas, que têm apenas 23 cromossomos. Os membros de um par recebem o nome de cromossomos homólogos.

O **gene** é uma unidade hereditária que consiste numa seqüência particular de bases no DNA e que especifica a produção de uma certa proteína (por exemplo, uma enzima).

Existem três tipo de genes. Aqueles que são apenas transcritos, os que são transcritos e traduzidos e os que não são transcritos e conseqüentemente não são traduzidos.

Os genes estão presentes em pares denominados **alelos**, sendo que cada alelo está localizado em um dos cromossomos homólogos. Eles estão situados num **locus** específico que ocupa a mesma posição em cada cromossomo.

Quando um gene se expressa, sua informação é primeiramente copiada no ácido ribonucléico (RNA), que por sua vez dirige a síntese dos produtos elementares do gene, as proteínas específicas.

O termo transcrição é empregado como sinônimo de síntese do RNA, e tradução como sinônimo de síntese protéica.

Três Nucleotídeos codificam um Aminoácido

Os códons, ou unidades hereditárias que contém o código de informação para um aminoácido, são compostos por três nucleotídeos (um trio)(Quadro 3.1[GC98]). Esta informação encontra-se no DNA, de onde é transcrita para o RNA mensageiro; assim, o mRNA possui a seqüência de bases complementar à do DNA do qual foi copiado. O DNA e o mRNA possuem somente quatro bases diferentes, enquanto que as proteínas contêm 20 diferentes aminoácidos. Dessa maneira, o código é lido em grupos de três bases, sendo três o número mínimo necessário para a codificação de 20 aminoácidos. [As permutações possíveis das quatro bases são $4^3 = 64$. Se o código genético fosse constituído por duplas, o número de códons seria insuficiente ($4^2 = 16$) e se fossem utilizados grupos de quatro bases as possibilidades ultrapassariam em muito o necessário ($4^4 = 256$).]

First Position	Second Position of Codon: T	Second Position of Codon: C	Second Position of Codon: A	Second Position of Codon: G	Third Position
T	TTT Phe [F]	TCT Ser [S]	TAT Tyr [Y]	TGT Cys [C]	T
	TTC Phe [F]	TCC Ser [S]	TAC Tyr [Y]	TGC Cys [C]	C
	TTA Leu [L]	TCA Ser [S]	TAA *Ter* [end]	TGA *Ter* [end]	A
	TTG Leu [L]	TCG Ser [S]	TAG *Ter* [end]	TGG Trp [W]	G
C	CTT Leu [L]	CCT Pro [P]	CAT His [H]	CGT Arg [R]	T
	CTC Leu [L]	CCC Pro [P]	CAC His [H]	CGC Arg [R]	C
	CTA Leu [L]	CCA Pro [P]	CAA Gln [Q]	CGA Arg [R]	A
	CTG Leu [L]	CCG Pro [P]	CAG Gln [Q]	CGG Arg [R]	G
A	ATT Ile [I]	ACT Thr [T]	AAT Asn [N]	AGT Ser [S]	T
	ATC Ile [I]	ACC Thr [T]	AAC Asn [N]	AGC Ser [S]	C
	ATA Ile [I]	ACA Thr [T]	AAA Lys [K]	AGA Arg [R]	A
	ATG Met [M]	ACG Thr [T]	AAG Lys [K]	AGG Arg [R]	G
G	GTT Val [V]	GCT Ala [A]	GAT Asp [D]	GGT Gly [G]	T
	GTC Val [V]	GCC Ala [A]	GAC Asp [D]	GGC Gly [G]	C
	GTA Val [V]	GCA Ala [A]	GAA Glu [E]	GGA Gly [G]	A
	GTG Val [V]	GCG Ala [A]	GAG Glu [E]	GGG Gly [G]	G

Figura 9. O código genético.

O comprimento da porção codificadora de um gene depende da extensão da mensagem a ser traduzida, isto é, o número de aminoácidos da proteína. Por exemplo, uma seqüência de 1.500 nucleotídeos pode conter 500 códons que codificam para uma proteína que contém 500 aminoácidos. A mensagem é lida a partir de um ponto inicial fixo sinalizado por **códons de iniciação** especiais.

A seqüência de trios determina a seqüência dos aminoácidos de uma proteína. Os aminoácidos, no entanto, não são capazes de reconhecer por si sós um dado trio do mRNA; para que isso aconteça, cada aminoácido precisa ligar-se a uma molécula adaptadora denominada RNA de transferência (tRNA). Cada molécula de tRNA possui um sítio de ligação do aminoácido e um outro local para o reconhecimento dos trios do mRNA. Este último sítio é denominado de anti-códon e consiste em três nucleotídeos que podem estabelecer um pareamento de bases com o códon complementar do mRNA. A tradução da mensagem numa proteína ocorre nos ribossomos, que asseguram a interação ordenada de todos os componentes envolvidos na síntese protéica.

Por volta de 1964 todos os 64 códons possíveis haviam sido decifrados. 61 códons correspondem a aminoácidos e 3 representam sinais para a terminação das cadeias polipeptídicas. Sabendo que existem somente 20 aminoácidos, fica evidente que vários trios podem codificar para o mesmo aminoácido; isto é, alguns dos trios são sinônimos. A prolina, por exemplo, é codificada por CCU, CCA, CCG e CCC. Note que na maioria dos casos os códons que são sinônimos diferem somente na base que ocupa a terceira posição no trio e que as duas primeiras bases são mais inflexíveis na codificação. Em conseqüência, as mutações que atingem a terceira base freqüentemente passam desapercebidas (mutações silenciosas) pois elas podem não alterar a composição de aminoácidos da proteína.

O **sinal de iniciação** para a síntese protéica é o códon AUG. O **sinal de terminação** é fornecido pelos códons UAG, UAA, UGA. Quando o ribossomo atinge o códon de terminação, a cadeia polipeptídica completa é liberada.

Seqüências Intercaladas nos Genes Eucarióticos

Inesperadamente, observou-se que nos eucariontes a informação para mRNAs covalentemente contíguos está freqüentemente localizada em segmentos de DNA não contíguos. Em outras palavras, os genes são interrompidos por inserções de DNA não codificador. Estas seqüências de DNA inseridas, que não são encontradas no mRNA

maduro, são denominadas *seqüências intercaladas* ou **íntrons**. Foram encontrados íntrons em genes da globina, albumina de ovo, imunoglobina, tRNA e muitos outros genes. Nem todos os genes eucarióticos são interrompidos; aqueles que codificam para as histonas e alguns tRNAs, por exemplo, são contínuos.

As partes da seqüência de DNA que produzem proteína são chamadas de **éxons**.

Mutação

Outro conceito importante da biologia é o de mutação, que é uma mudança no conteúdo do DNA. Os tipos de mudanças podem ser de substituição de base, inserção de base, remoção de base, e rearranjo ou troca na ordem de segmentos de base. Estas mudanças podem ser divididas em classes dependendo da escala com que elas ocorrem. Algumas mudanças são fenômenos localizados, enquanto outras ocorrem um milhão de vezes seguidas.

Genoma

O genoma é nome que se dá ao conjunto de cromossomos de uma célula de um organismo.

2.2. Definições

Esta seção destina-se a apresentar algumas definições importantes que serão usadas ao longo do texto.

Usaremos o termo **seqüência finita de caracteres**, ou simplesmente **seqüência** ou **cadeia**, no sentido restrito de uma seqüência finita de caracteres de um dado alfabeto Σ. Assim, se Σ = {A,C,T,G}, então ATTCCG e CCGA são seqüências.

Dada uma seqüência u, uma **subseqüência** de u é uma seqüência que pode ser obtida a partir de u pela remoção de alguns caracteres. Assim, TTT é subseqüência de ATATAT, mas AAAA não é.

Uma **subcadeia** de u corresponde a uma subseqüência de caracteres consecutivos de u. Assim TAC é subcadeia de AGTACA, mas não de ATGAC.

A **concatenação** de duas seqüências u e v é a seqüência uv obtida justapondo os caracteres de v aos de u. Se w = uxv, dizemos que u é um **sufixo** de w, x é um **fator** de

w, e v é um **prefixo** de w. Note que fator é o mesmo que subcadeia e que prefixos e sufixos são também fatores.

Uma **biosseqüência** é uma seqüência onde Σ = {A,C,G,T} (DNA) ou Σ = {A,C,G,U} (RNA) ou Σ é formado pelos aminoácidos da tabela de proteínas apresentada na seção anterior.

Um **buraco** é um símbolo não pertencente ao alfabeto Σ, representado pelo caracter "-". O buraco pertence ao alfabeto estendido $\Sigma^+ = \Sigma \cup \{-\}$.

2.3. Motivações

O objetivo desta seção é apresentar motivações para este estudo de comparação de seqüências. Será explicado porquê é necessário comparar seqüências e usar técnicas computacionais para isso.

A comparação de seqüências é a operação primitiva mais importante na área de biologia computacional e serve de base para muitas outras manipulações mais elaboradas. A grosso modo, esta operação consiste em encontrar trechos semelhantes nas seqüências de entrada. Contudo, por trás desta aparente simplicidade, esconde-se uma vasta gama de problemas distintos, com formalizações diversas, muitos deles exigindo algoritmos e estruturas de dados próprias para sua execução eficiente.

É possível citar alguns exemplos práticos em que aparece biologia computacional. São eles:

1. Sejam duas seqüências sobre o mesmo alfabeto com aproximadamente 10.000 caracteres. Suponha que elas possuem composições idênticas, exceto por divergências isoladas (inserções, remoções ou substituições de caracteres) que ocorrem a taxa de um erro a cada 100 caracteres. Deseja-se encontrar estes erros. Este problema aparece quando um gene é sequenciado por dois laboratórios diferentes e deseja-se comparar os resultados, ou quando a seqüência foi digitada mais de uma vez e deseja-se tratar erros de digitação.
2. Sejam duas seqüências de algumas centenas de caracteres sobre um mesmo alfabeto. Deseja-se decidir se existe um prefixo de uma delas que seja semelhante a um sufixo da outra. Em caso afirmativo um alinhamento entre as regiões semelhantes deve ser produzido. Suponha esta mesma situação, exceto que em vez de duas, existam 500

seqüências que devem ser comparadas duas a duas. Estes problemas aparecem no contexto de montagem de fragmentos em programas de auxílio a seqüenciamento de DNA em larga escala.

3. Sejam duas seqüências de algumas centenas de caracteres sobre um mesmo alfabeto. Deseja-se decidir se há algum trecho de uma delas semelhante a um trecho de tamanho aproximadamente igual na outra. A semelhança não é medida em termos de porcentagem de caracteres idênticos, mas em termos de um esquema de pontuação que atribui uma nota fixa a cada par de caracteres do alfabeto. Dois trechos são considerados semelhantes se a soma das notas dadas a caracteres alinhados for superior a um dado valor. Suponha esta mesma situação, exceto que, em vez de duas, temos uma seqüência fixa que deve ser comparada a milhares de outras. Estes problemas aparecem no contexto de buscas de semelhanças locais usando bases de dados de biosseqüências.

O uso de técnicas computacionais é justificado quando as aplicações lidam com grande volume de dados, que é o caso das buscas em bases de biosseqüências. Mesmo no caso em que se poderia resolver à mão, é mais conveniente usar técnicas computacionais porque o olho humano pode falhar e alguns fatos podem passar despercebidos.

3. O Algoritmo Básico

O objetivo principal desta seção é apresentar um algoritmo eficiente que, dadas duas seqüências, computa o melhor alinhamento entre elas. Na primeira parte será feita uma descrição sobre o algoritmo básico de alinhamento entre duas seqüências, e na segunda parte serão apresentadas algumas extensões que foram feitas nestes algoritmos.

3.1. Descrição

Veja o exemplo apresentado na figura a seguir que considera as seqüências de DNA: GACGCATTAG e GATCGGAATAG. Elas são muito semelhantes. As únicas diferenças são um T a mais na segunda seqüência e uma troca de A por T na quarta posição da direita para a esquerda. Note que foi preciso introduzir um buraco na primeira seqüência para que as bases iguais antes e depois do buraco se alinhassem nas duas seqüências.

Figura 10. Seqüências de DNA.

Informalmente, um **alinhamento** é definido como sendo a inserção de buracos em pontos arbitrários ao longo das seqüências de modo que elas fiquem com o mesmo comprimento, e na disposição das duas cadeias resultantes uma em cima da outra de maneira que todos os caracteres ou buracos de uma das cadeias seja comparado a um único caracter ou a um único buraco na outra seqüência. Além disso, não é permitido que um buraco em uma das seqüências esteja alinhado com um buraco na outra. E ainda, os buracos poderão ser colocados no início e no final das seqüências.

A disposição das seqüências alinhadas apresenta os pares de caracteres das seqüências que estão alinhados. Cada par de caracteres alinhados forma uma **coluna** do alinhamento. No exemplo da figura anterior há 11 colunas. Os caracteres alinhados são: G|G, A|A, -|T, C|C, G|G, G|G, A|A, T|A, T|T, A|A, e G|G.

Dado um alinhamento, podemos atribuir uma **pontuação**. Cada coluna do alinhamento receberá um certo número de pontos. A pontuação total do alinhamento será a soma das

pontuações das colunas. Como exemplo, será usada o seguinte esquema de pontuação: se a coluna tiver duas bases iguais receberá 1, se tiver duas bases diferentes receberá -1, e se possuir uma base e um buraco receberá -2. Estes valores são usados na prática. Procura-se valorizar bases iguais alinhadas e penalizar alinhamentos de bases desiguais e buracos.

No exemplo da figura anterior a pontuação seria $9 * (+1) + 1 * (-1) + 1 * (-2) = 6$.

Há vários alinhamentos possíveis para duas seqüências. Cada alinhamento terá sua pontuação. O **alinhamento ótimo** é o que tem a maior pontuação. A **similaridade** entre as duas seqüências é o nome que se dá a pontuação máxima.

Agora que já foi definido alguns conceitos importantes e já foi apresentado um exemplo, será apresentado o algoritmo básico. Para ilustrar o funcionamento dele, considere o problema de obter o alinhamento ótimo para o par de seqüências: AAAC e AGC.

A idéia básica está em analisar todas as possibilidades de alinhamento para a última coluna e descobrir qual delas é o alinhamento ótimo. No exemplo temos três possibilidades para a última coluna: alinhar C com C, C com buraco, ou buraco com C; já que não é possível alinhar buraco com buraco. Veja a figura a seguir.

Figura 11. Possibilidades de alinhamento da última coluna.

Depois de encontrar todas as possibilidades; calcula-se a pontuação para cada uma delas, e faz-se o mesmo para o restante das seqüências. Veja pelo esquema a seguir que este raciocínio nos leva a um método recursivo.

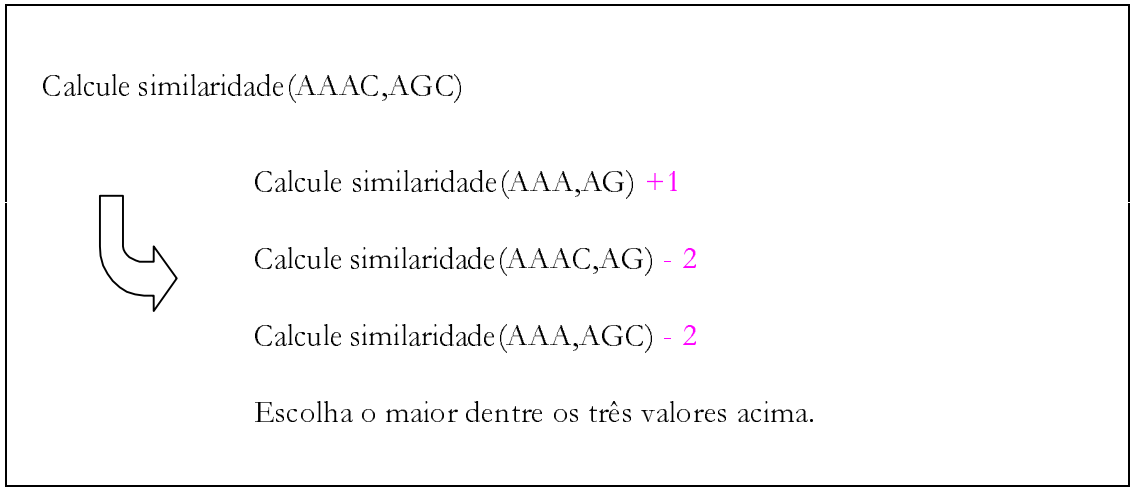


Figura 12. Esquema do método recursivo do algoritmo.

O problema deste método é que, se aplicado diretamente, vai gerar um número exponencial de chamadas recursivas. Contudo muitas destas chamadas são redundantes. Veja isto na figura a seguir que mostra uma árvore que representa estas chamadas recursivas.

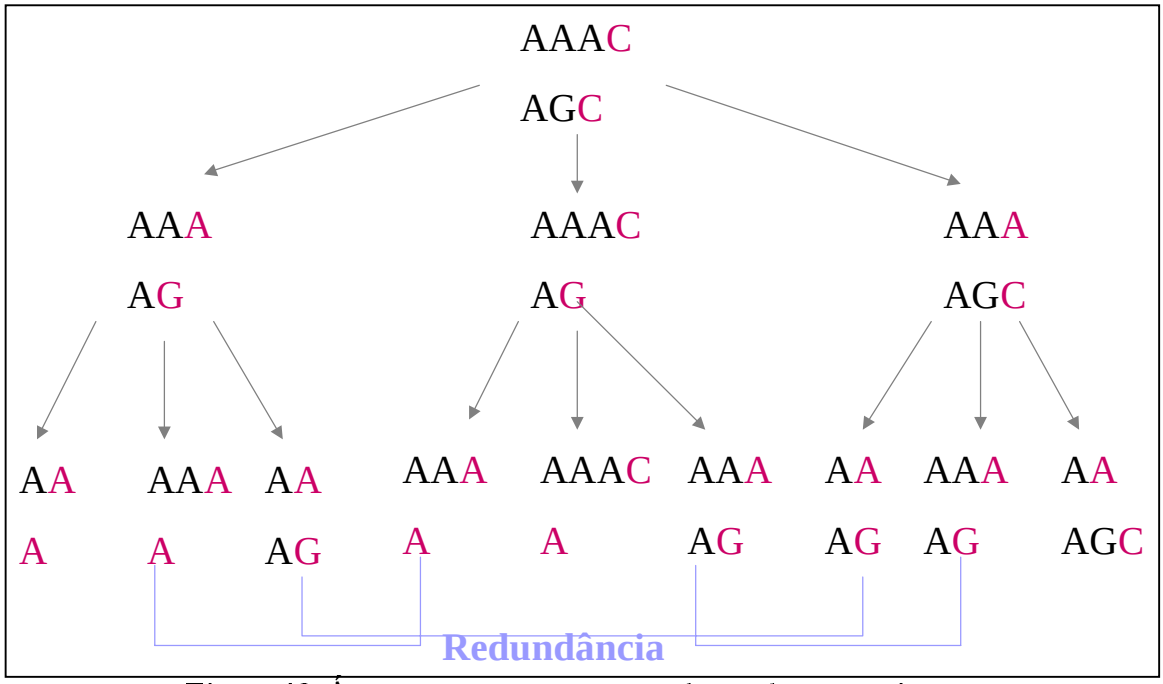


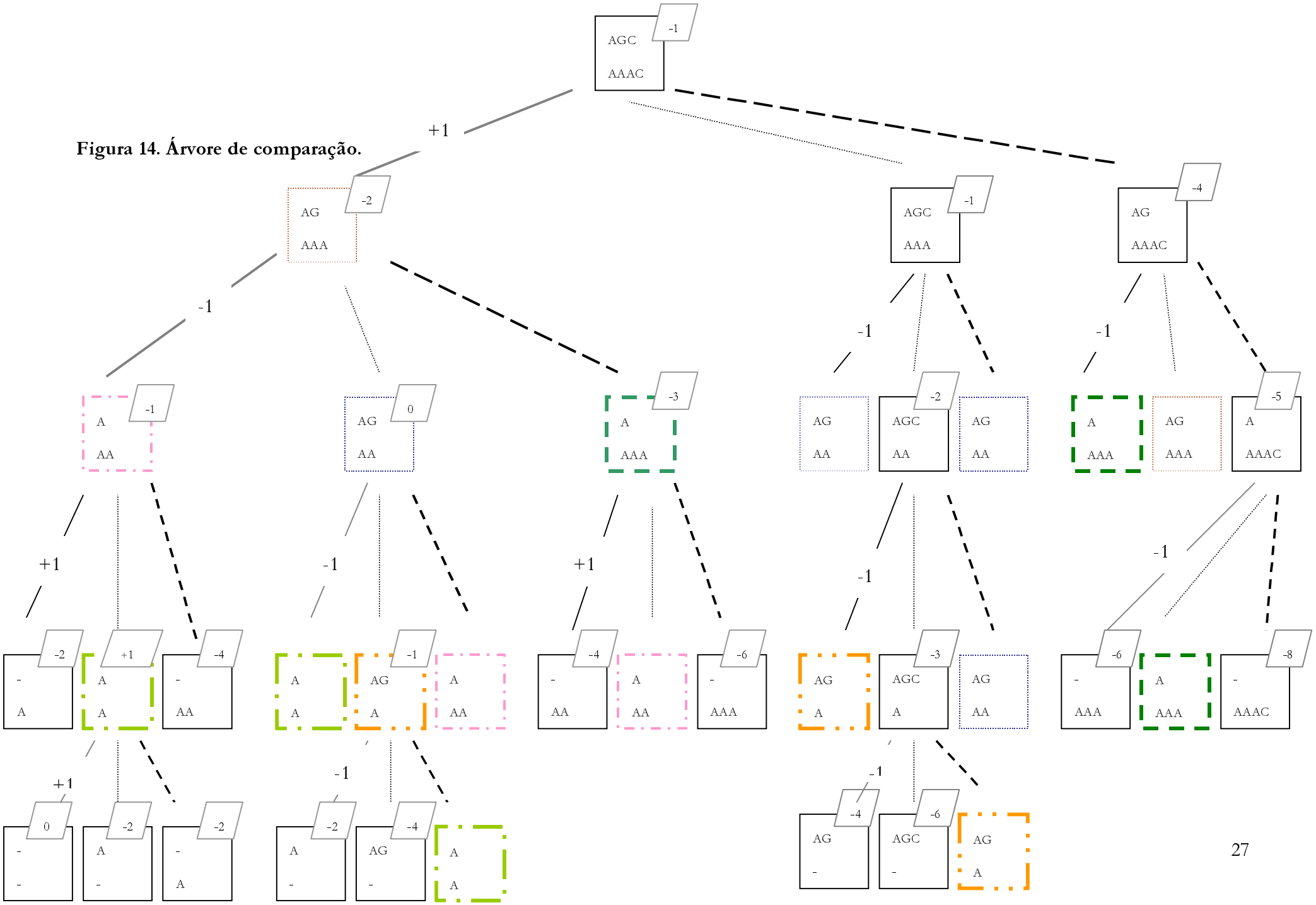
Figura 13. Árvore que representa as chamadas recursivas.

Não há necessidade de calcular mais de uma vez uma comparação, desde que os resultados sejam guardados de maneira que possam ser consultados rapidamente.

Programação Dinâmica é exatamente isso: observar que, embora um método gere um número muito grande de chamadas recursivas, o número total de instâncias de entrada é limitado, de modo que o cálculo pode ser feito uma só vez e armazenado para consultas posteriores, se necessário. Em geral, uma matriz é usada para guardar resultados parciais [Meidanis, 94].

A seguir está a árvore que apresenta todas as chamadas recursivas que devem ser feitas para se resolver o problema do exemplo. Note que as chamadas redundantes não estão presentes.

Figura 14. Árvore de comparação.



A árvore da figura está esquematizada seguindo os seguintes parâmetros:

- Cada nó da árvore corresponde à comparação de duas seqüências.
- Cada nó tem três filhos. Cada filho corresponde a uma possibilidade para a última coluna. Analisando da esquerda para a direita, temos os seguintes casos:
  - O primeiro filho corresponde à possibilidade do alinhamento dos dois caracteres da última coluna. Se os dois caracteres forem iguais então a aresta indica a pontuação +1, caso contrário indica -1. Esta aresta é apresentada de forma contínua e na cor cinza.
  - O segundo filho corresponde à possibilidade do alinhamento do caracter da última coluna da primeira seqüência com um buraco na segunda seqüência. Este alinhamento tem sempre pontuação -2, e por isso não foi indicado na figura. Esta aresta é apresentada de forma pontilhada, fina e na cor preta.
  - O terceiro filho corresponde à possibilidade do alinhamento de um buraco na primeira seqüência e o caracter da última coluna da segunda seqüência. Este alinhamento tem sempre pontuação -2, e por isso não foi indicado na figura. Esta aresta é apresentada de forma pontilhada, grossa e na cor preta.
- Cada nó tem também a pontuação maior de suas possibilidades exibida em um paralelogramo na cor cinza.
- Alguns nós estão com estilos diferentes do comum, que possui linha contínua e preta. Isto acontece porque eles aparecem mais de uma vez na árvore, o que indica que serão chamados mais de uma vez na recursão, ou seja, eles representam as redundâncias. Nestes casos, a análise será feita somente no primeiro nó que aparece da esquerda para a direita.
- Os nós que possuem uma das seqüências vazia (ou as duas) indicam a base da recursão. A pontuação de uma base é igual a 2k, sendo k o tamanho da seqüência não vazia.

O valor do alinhamento ótimo para as seqüências originais só é obtido no final da recursão. Para calculá-lo é necessário chegar até às folhas e retornar da recursão. Portanto os valores exibidos nos paralelogramos só são calculados no retorno da recursão.

Para ilustrar este fato, vamos pensar na primeira chamada da recursão, indicada pelo nó que possui as seqüências originais: AGC e AAAC. Para calcular o valor do alinhamento ótimo delas é necessário calcular o alinhamento ótimo de seus três filhos: AG e AAA, AGC e AAA, AG e AAAC. Da mesma forma, para calcular o alinhamento ótimo de AG e AAA é necessário calcular o alinhamento ótimo de seus três filhos, e assim por diante. Este raciocínio é aplicado até que um nó base é atingido, como por exemplo, o nó com as seqüências A e buraco (-). Neste caso a pontuação é 2, porque k é igual a 1 (tamanho da seqüência A).

O retorno da recursão de um determinado nó só pode ser feito quando todos os seus três filhos souberem sua pontuação ótima. Neste momento é necessário calcular a soma da pontuação de cada filho com o valor da aresta que ele se liga ao pai. Existirão então três valores calculados, um para cada filho. A pontuação ótima do nó pai é o maior destes valores. Por exemplo, imagine que a recursão está quase no fim e as pontuações dos alinhamentos dos três filhos de AGC e AAAC já foram calculadas. Sabe-se que a pontuação de AG e AAA é -2, de AGC e AAA é -1, e de AG e AAAC é -4. É necessário calcular as somas das pontuações com os valores das arestas. Para o primeiro filho, a soma será (+1) + (-2) = -1; para o segundo filho, a soma será (-1) + (-2) = -3; para o terceiro filho a soma será (-4) + (-2) = -6. O maior valor é -1, e portanto a pontuação ótima de AGC e AAAC é -1.

Como foi dito anteriormente o método de programação dinâmica será usado para se resolver este problema e evitar que se calcule todas as redundâncias. Para tanto é preciso que se tenha uma estrutura de dados que armazene valores para possíveis consultas posteriores. Neste caso será usada uma matriz. Ela guardará os valores essenciais da árvore apresentada anteriormente.

A estrutura da matriz será apresentada junto com uma comparação com a estrutura da árvore.

A segunda linha da matriz indica a primeira seqüência s1 (no exemplo é a AGC). A segunda coluna indica a segunda seqüência s2 (no exemplo é a AAAC). A seqüência s1 tem tamanho genérico m, e por isso ela será representada por s1[1…m]. Analogamente, a seqüência s2 tem tamanho n e é representada por s2[1…n]. Quando o prefixo de comprimento i da seqüência s1 for considerado, será usado s1[1..i]. Analogamente, será usado s2[1…j]. Os valores de i e de j são exibidos na primeira linha e primeira coluna da matriz.

Cada célula da matriz representa a comparação entre dois prefixos de s1 e s2. Na célula estará apresentado o valor da pontuação da comparação entre as duas subseqüências. Veja na figura a seguir um esquema da matriz para o exemplo. As pontuações não foram colocadas ainda. Esta figura tem o objetivo de indicar qual comparação cada célula representa. Em cada célula são exibidos os prefixos que estão sendo comparados de duas formas: com os caracteres que os representam, e genericamente, usando s1[1..i] e s2[1…j].

	i	**0**	**1**	**2**	**3**
j	**s1[1…m]** **s2[1…n]**	**-**	**A**	**G**	**C**
0	**-**	-|- s1[0]|s2[0]	A| - s1[1]|s2[0]	AG| - s1[1..2]|s2[0]	AGC| - s1[1..3]|s2[0]
1	**A**	-|A s1[0]|s2[1]	A| A s1[1]| s2[1]	AG| A s1[1..2]|s2[1]	AGC| A s1[1..3]|s2[1]
2	**A**	-|AA s1[0]|s2[1..2]	A| AA s1[1]|s2[1..2]	AG| AA s1[1..2]|s2[1..2]	AGC| AA s1[1..3]|s2[1..2]
3	**A**	-|AAA s1[0]|s2[1..3]	A| AAA s1[1]|s2[1..3]	AG| AAA s1[1..2]|s2[1..3]	AGC| AAA s1[1..3]|s2[1..3]
4	**C**	-|AAAC s1[0]|s2[1..4]	A| AAAAC s1[1]|s2[1..4]	AG| AAAAC s1[1..2]|s2[1..4]	AGC| AAAAC s1[1..3]|s2[1..4]

Figura 15. Matriz indicando em cada célula as subseqüências que estão sendo comparadas.

Note que a terceira coluna e a terceira linha possuem células em que pelo menos um dos prefixos é vazio, e por isso estas células correspondem à base da recursão.

Para ilustrar uma analogia entre a árvore e a matriz será apresentado o caso dos prefixos A e AAA. Veja a seguir a figura.

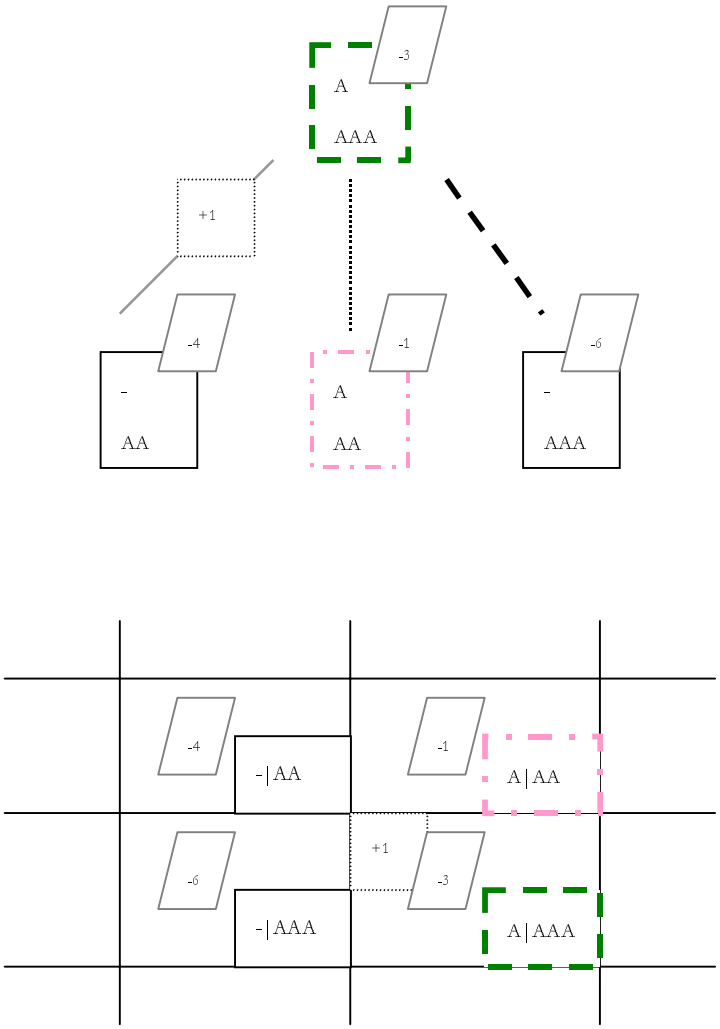


Figura 16. Esquema comparativo da árvore e da matriz.

Para calcular a pontuação de A|AAA é necessário calcular as pontuações de -|AA, A|AA, e -|AAA. Na árvore estas três comparações estão representadas como filhos de A|AAA; já na matriz elas são células vizinhas de A|AAA encontradas a esquerda, a diagonal esquerda, e acima.

A pontuação de A|AAA é calculada da seguinte forma na matriz:

- Soma-se a pontuação (-1) de A|AA a (-2). Isto porque o alinhamento da última coluna que foi feito em A|AAA para se obter A|AA foi A|-, cuja pontuação é -2.
- Soma-se a pontuação (-6) de -|AAA a (-2). Isto porque o alinhamento da última coluna que foi feito em A|AAA para se obter -|AAA foi -|A, cuja pontuação é -2.
- Soma-se a pontuação (-4) de -|AA a (+1). Isto porque o alinhamento da última coluna que foi feito em A|AAA para se obter -|AA foi A|A, cuja pontuação é +1.

– A maior das somas acima será a pontuação de A|AAA.

No caso teríamos os valores -3, -8 e -3. Sendo assim a pontuação de A|AAA é -3.

Veja a seguir um esquema genérico para o cálculo da pontuação de qualquer célula da matriz sendo comparado a árvore.

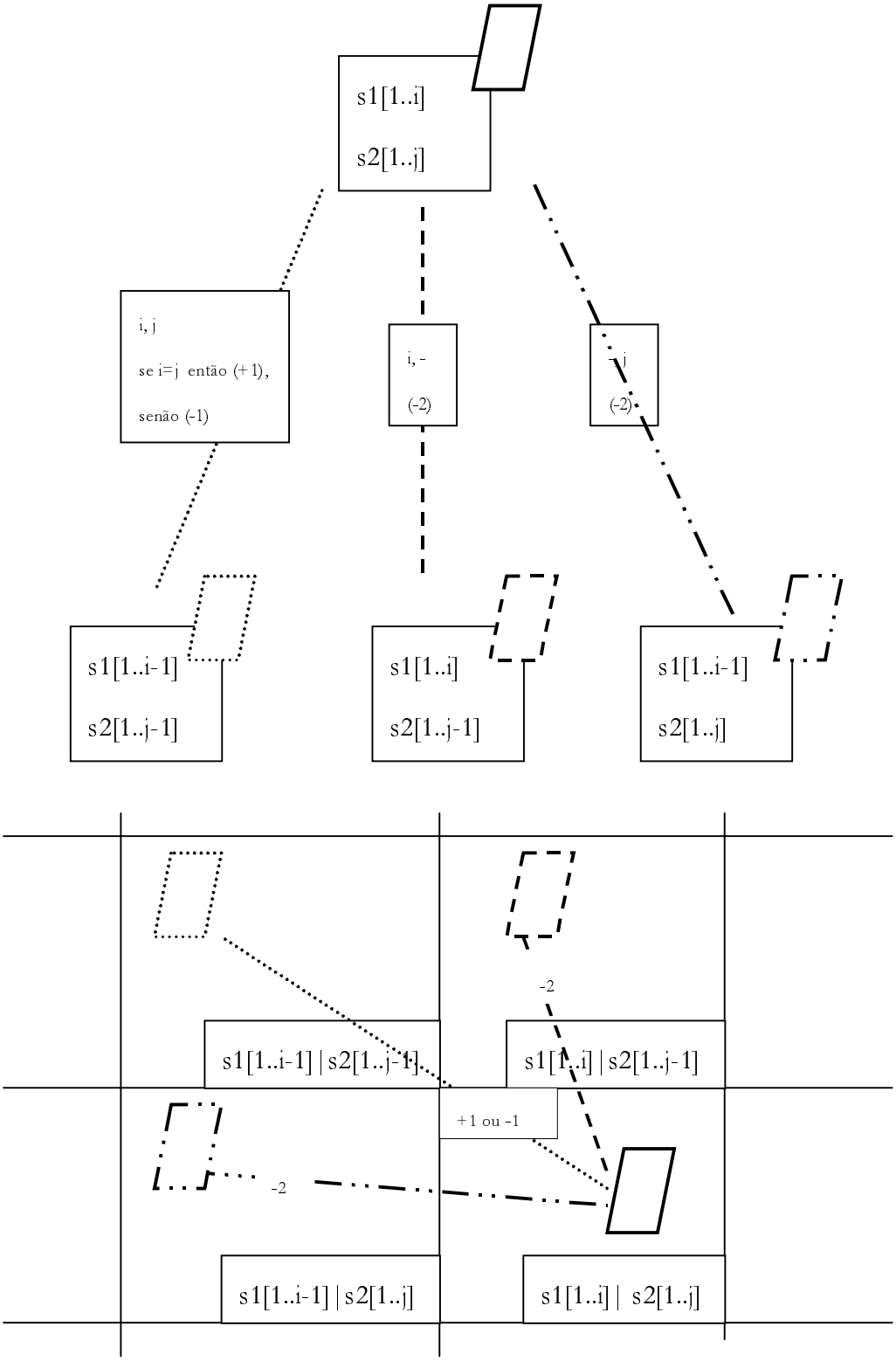


Figura 17. Esquema de pontuação da árvore e da matriz.

A pontuação de qualquer célula da matriz depende da pontuação de suas vizinhas assim como a pontuação de um nó da árvore depende de seus filhos. As células vizinhas à esquerda e acima da célula que está sendo calculada a pontuação devem ter suas pontuações somadas a (-2), assim como o segundo e o terceiro filho do nó está sendo calculado a pontuação. A célula vizinha na diagonal esquerda corresponde ao primeiro filho na árvore, e por isso deve ter sua pontuação somada a (-1) ou a (+1).

Simplificadamente, o seguinte algoritmo mostraria o que deve ser feito para cada célula:

```
/* se (i = j) então p(i,j) = 1, senão p(i,j) = -1 */

pontuação_célula(i,j) = MAX(      pontuação_célula(i,j-1) -2,
                                  pontuação_célula(i-1,j-1) + p(i,j)
                                  pontuação_célula(i-1,j) -2)
```

Figura 18. Algoritmo simplificado para o cálculo da pontuação de uma célula da matriz.

A seguir está a matriz completa do exemplo com todas as pontuações apresentadas nos centros das células. Exceto nas células que representam as bases da recursão, há também o número +1 ou -1 (p(i, j)). Não é necessário apresentar o número -2 porque ele está fixo em todos os casos que deve aparecer.

	i	0	1	2	3
j	s1[1…m] s2[1…n]	-	A	G	C
0	-	**0** -|-	**-2** A|-	**-4** AG|-	**-6** AGC|-
1	A	**-2** -|A	+1 **+1** A|A	-1 **-1** AG|A	-1 **-3** AGC|A
2	A	**-4** -|AA	+1 **-1** A|AA	-1 **0** AG|AA	-1 **-2** AGC|AA
3	A	**-6** -|AAA	+1 **-3** A|AAA	-1 **-2** AG|AAA	-1 **-1** AGC|AAA
4	C	**-8** -|AAAC	-1 **-5** A|AAAAC	-1 **-4** AG|AAAAC	+1 **-1** AGC|AAAAC

Figura 19. Matriz do exemplo AGC e AAAC.

A figura a seguir apresenta o algortimo básico usado para preencher a matriz dada. Os cálculos devem ser efetuados de modo que os valores de células anteriores necessários para determinar o valor de uma célula estejam disponíveis no momento certo.

Este algoritmo calcula a similaridade entre s1[1..m] e s2[1..n], dependendo de um parâmetro g que indica o peso de um buraco (geralmente g<0) e de uma função p para pares de caracteres. Este parâmetro g e esta função p já estavam sendo utilizados até agora, só que ainda não tinha sido definidos formalmente. Foi usado g = -2, p(a,a) = 1 e p(a,b) = -1 se a ‡ b nos cálculos da matriz do exemplo.

```
Sim(s1,s2)

      /*supondo |s1| = m, |s2| = n*/

      for i • 0 to m do
            a[i,0] • i*g
      for j • 0 to n do
            a[0,j] • j*g
      for i • 1 to m do
            for j • 1 to n do
                  a[i,j] = max(   a[i,j-1] + g,
                                  a[i-1,j-1] + p(i,j)
                                  a[i-1,j] + g)
      return a[m,n]
```

Figura 20. Algoritmo básico de programação dinâmica para comparação de seqüências.

Com as pontuações de todas as células calculadas é possível calcular o alinhamento ótimo. Para tanto é preciso percorrer as células da matriz. Cada passo de uma célula para outra corresponde a uma aresta da árvore e a uma coluna do alinhamento. Genericamente, estando na célula a[i,j], é possível ir para as células a[i-1,j], a[i-1,j-1] e a[i,j-1]. Se o passo for de a[i,j] para a[i,j-1], significa que está sendo considerado o alinhamento de um buraco em s2 e um caracter em s1. Se o passo for de a[i,j] para a[i-1,j], significa que está sendo considerado o alinhamento de um buraco em s1 e um caracter em s2. Se o passo for de a[i,j] para a[i-1,j-1], significa que está sendo considerado o alinhamento de um caracter em s2 e um caracter em s1. Veja o esquema na figura a seguir.

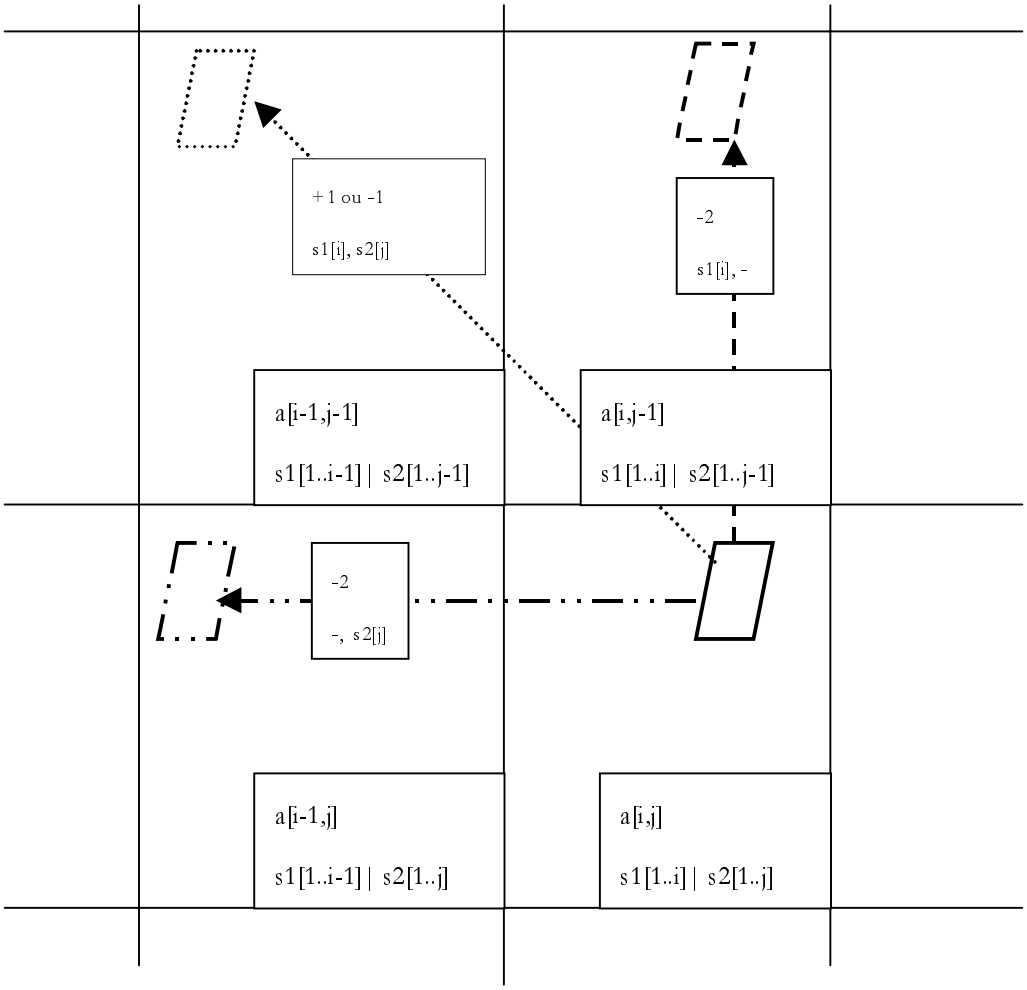


Figura 21. Esquema do cálculo do alinhamento.

Seguindo este raciocínio, o alinhamento ótimo pode ser construído de trás para frente a partir da matriz calculada anteriormente pelo algoritmo básico. Estando em uma determinada célula, descobre-se como foi obtido a pontuação dela. Para isso, basta fazer as somas das pontuações das células vizinhas com p(i,j) ou g (dependendo do caso), como já foi descrito, e encontrar qual das somas é igual à pontuação procurada. Descobrindo a célula vizinha responsável pela pontuação da célula que está sendo tratada, descobre-se também o alinhamento de uma determinada coluna. O próximo passo é fazer a mesma análise descrita para esta célula vizinha. Desta forma a matriz vai sendo percorrida.

É possível que mais de uma célula vizinha possa ser a responsável pela pontuação da célula que está sendo tratada. Neste caso, qualquer uma delas é válida, dependendo do algoritmo implementado. Fica a critério do programador determinar as prioridades.

Aplicando esta idéia, é possível analisar a matriz do exemplo e descobrir o alinhamento ótimo de AGC e AAAC.

Siga as setas na matriz e os passos a seguir:

- A análise se inicia na última célula da matriz, que corresponde às seqüências AGC e AAAC. Testa-se como a pontuação desta célula é -1. A soma da vizinha acima seria (-1-2) = -3, da diagonal esquerda seria (-2+1) = -1, e da esquerda seria (-4-2) = -6. Então a pontuação é igual à soma da vizinha da diagonal esquerda, que corresponde ao alinhamento da última coluna de AGC|AAAC, que é C|C (lembrando que os outros casos tratados eram C|- e -|C). Passamos a analisar a vizinha da diagonal esquerda.

- A análise continua na célula das seqüências AG|AAA. Testa-se como a pontuação desta célula é -2. A soma da vizinha acima seria (0-2) = -2, da diagonal esquerda seria (-1-1) = -2, e da esquerda seria (-3-2) = -5. Então a pontuação é igual à soma da vizinha da diagonal esquerda, que corresponde ao alinhamento da última coluna de AG|AAA, que é G|A (lembrando que os outros casos tratados eram G|- e -|A). Passamos a analisar a vizinha da diagonal esquerda.

- A análise continua na célula das seqüências A|AA. Testa-se como a pontuação desta célula é -1. A soma da vizinha acima seria (+1-2) = -1, da diagonal esquerda seria (-2+1) = -1, e da esquerda seria (-4-2) = -6. Então a pontuação é igual à soma da vizinha da diagonal esquerda, que corresponde ao alinhamento da última coluna de A|AA, que é A|A. Mas é igual também à soma da vizinha acima, que corresponde ao alinhamento de A|-. Neste momento qualquer um dos casos poderia ser seguido. Suponha que o algoritmo implementado dê prioridade para a análise da vizinha da diagonal esquerda.

- A análise continua na célula das seqüências -|A. Esta é uma célula base da recursão. Testa-se como a pontuação desta célula é -2. Neste caso só existe a célula vizinha de cima, e a soma dela deve ser igual à -2. Como esperado, a matriz mostra que a soma é (0-2) = -2, e, portanto, retorna-se a base da recursão, que é o alinhamento -|A.

- Voltando da recursão, podemos construir os vetores alin1[] e alin2[], que possuem o alinhamento de cada seqüência. Neste caso será:

alin1	-	A	G	C
alin2	A	A	A	C

Figura 22. Vetor com resultado do alinhamento.

	i	**0**	**1**	**2**	**3**
j	**s1[1...m]** **s2[1...n]**	**-**	**A**	**G**	**C**
0	**-**	**0** -|-	**-2** A| -	**-4** AG| -	**-6** AGC| -
1	**A**	**-2** -|A	+1 **+1** A|A A| A	-1 **-1** AG| A	-1 **-3** AGC| A
2	**A**	**-4** -|AA	+1 **-1** A| AA	-1 **0** G|A G| AA	-1 **-2** AGC| AA
3	**A**	**-6** -|AAA	+1 **-3** A| AAA	-1 **-2** AG| AA	-1 **-1** C|C AGC| AAA
4	**C**	**-8** -|AAAC	-1 **-5** A| AAAAC	-1 **-4** AG| AAAAC	+1 **-1** AGC| AAAAC

Figura 23. Cálculo do alinhamento ótimo pela matriz a.

Pela árvore também se chegaria à mesma conclusão. Veja a figura da árvore simplificada a seguir.

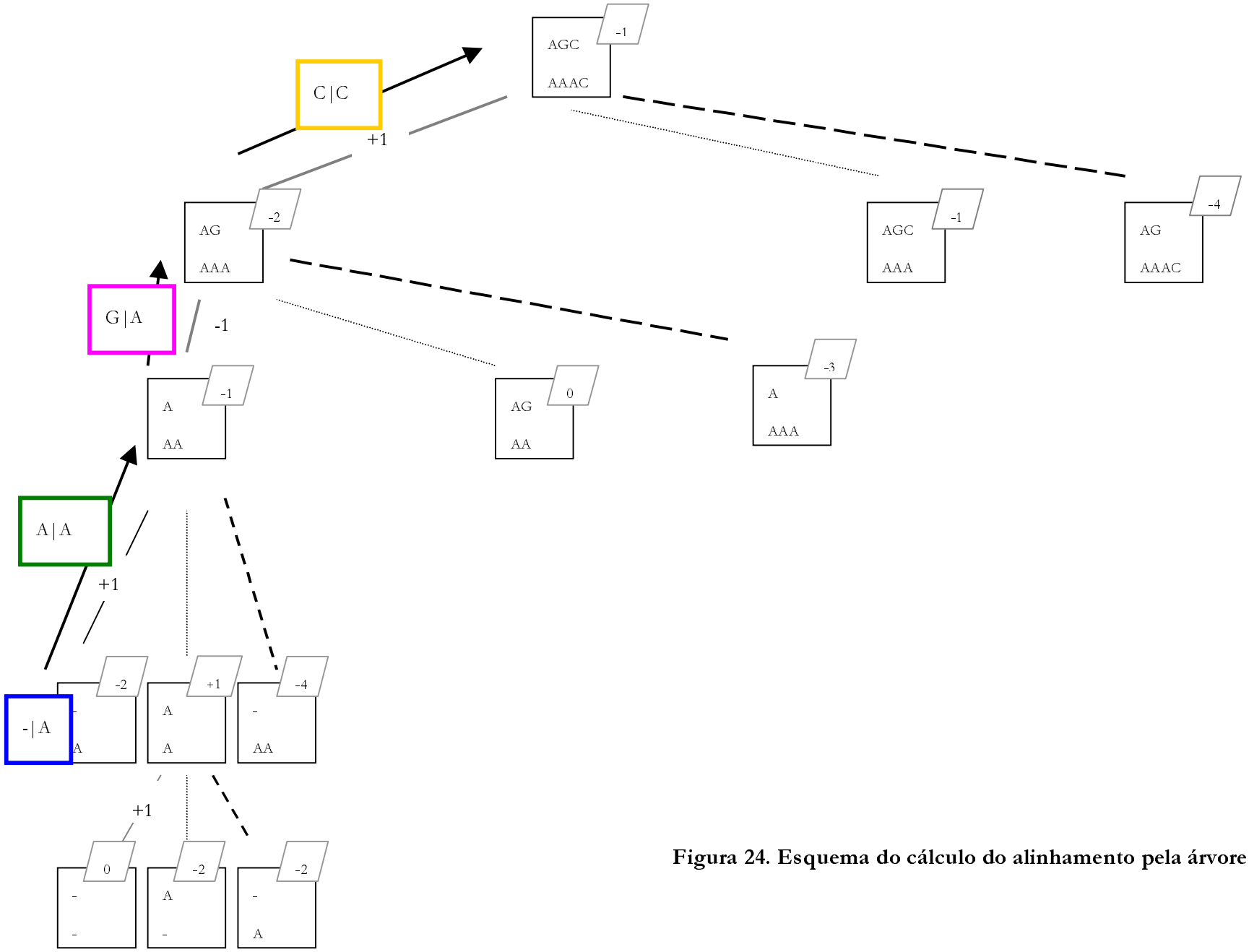


Figura 24. Esquema do cálculo do alinhamento pela árvore.

O algoritmo do alinhamento ótimo está na figura a seguir. Deve ser dada a matriz e as seqüências s1 e s2. A chamada é feita por Alin(m,n,t), sendo m o tamanho de s1, n o tamanho de s2, onde t é o comprimento do alinhamento que será retornado pelo algoritmo. A saída será um par de vetores alin1 e alin2 que conterão em suas posições de 1 a t os caracteres alinhados. Estes vetores são globais.

```
Alin(i,j,t)
      if (i=0) and (j=0) then
            t •0
      else if (i>0) and a[i,j] = a[i-1,j] + g then
            Alin(i-1,j,t)
                  t• t+1
            alin1[t] • s1[i]
            alin2[t] • '-'
      else if (i>0) and (j>0)  and a[i,j] = a[i-1,j-1]+ p(i,j) then
            Alin(i-1,j-1,t)
                  t• t+1
            alin1[t] • s1[i]
            alin2[t] • s2[j]
      else      /* tem que ser j>0 e a[i,j] = a[i,j-1]+ g */
            Alin(i,j-1,t)
                  t• t+1
            alin1[t] • '-'
            alin2[t] • s2[j]
```

Figura 25. Algoritmo de cálculo de alinhamento a partir da matriz *a*.

Complexidade

O algoritmo básico que preenche a matriz *a* (apresentado anteriormente na figura 20) com os valores das pontuações possui quatro malhas tipo for. As duas primeiras gastam tempo O(m) e O(n), respectivamente. As duas últimas malhas são encaixadas e preenchem a matriz. Gasta-se essencialmente O(mn) unidades de tempo nesta parte e esta é a parte dominante da complexidade temporal. O espaço gasto também é proporcional ao tamanho da matriz. Logo o algoritmo básico possui complexidade de tempo e de espaço O(mn).

O algoritmo que calcula o alinhamento a partir da matriz gasta tempo O(t), onde t é o tamanho do alinhamento retornado. Neste algoritmo, há essencialmente uma chamada recursiva por coluna de alinhamento, e cada chamada consome tempo O(1).

Economia de Espaço

A complexidade do algoritmo básico é quadrática, o que dificulta sua aplicação para seqüências muito longas. Com relação ao espaço de memória é possível reduzir as necessidades do algoritmo consideravelmente, de modo que passe a utilizar apenas espaço proporcional à soma dos tamanhos das seqüências de entrada.

Note que o preenchimento da matriz pode ser feito em espaço linear. Como cada linha depende apenas da anterior, é possível efetuar os cálculos mantendo apenas um vetor de tamanho n, que guardará parte da linha que está sendo construída e parte da linha anterior.

Veja no caso do exemplo. A primeira coluna é calculada normalmente e fica guardada no vetor como na figura a seguir.

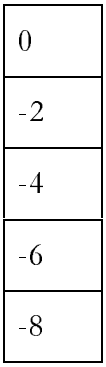


Figura 26. Vetor inicial com a primeira coluna da matriz.

A segunda coluna depende desta primeira. Utilizando o mesmo vetor, é possível calcular o valor da primeira célula da segunda coluna (que não depende de nenhum valor da primeira coluna) e substituí-lo no vetor. Veja na figura a seguir.

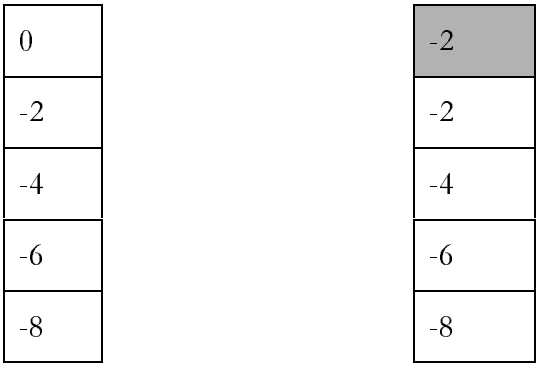


Figura 27. Vetor antigo e novo com a primeira célula substituída.

O valor da segunda célula da segunda coluna pode ser calculado. Ele depende do valor novo e antigo da primeira célula, e do valor da segunda célula. Veja na figura seguinte.

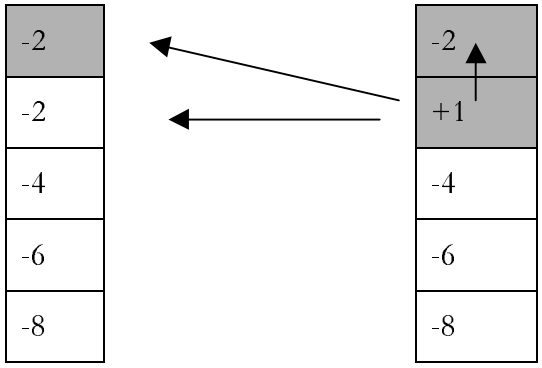


Figura 28. Vetor antigo e novo com a segunda célula substituída.

Seguindo o mesmo raciocínio calcula-se o valor da terceira célula.

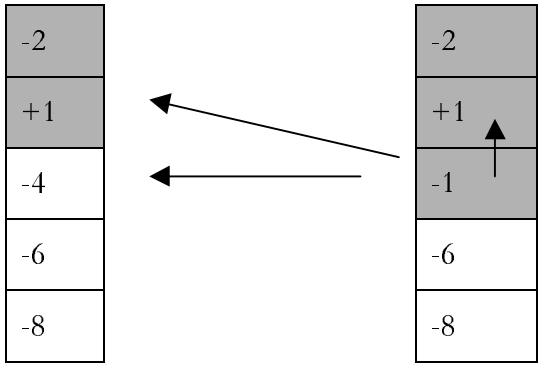


Figura 29. Vetor antigo e novo com a terceira célula substituída.

Esta idéia vai sendo aplicada até que o vetor todo fica preenchido por valores da segunda coluna, e o mesmo pode ser feito para se descobrir os valores da terceira coluna e das outras também.

O algoritmo que implementa estas idéias está a seguir.

```
BestScore(s1, s2, a)
      m •|s1|
      n •|s2|
for j • 0 to n do
                        a[j] • j *g
      for i • 1 to m do
            velho  • a[0]
            a[0] • i * g
            for j • 1 to n do
                  temp • a[j]
                  a[j] • max (  a[j] + g,
                                velho +  pont(s1[i],s2[j]),
                                a[j-1] + g)
                  velho • temp
```

Figura 30. Algoritmo para achar a pontuação de um alinhamento ótimo usando espaço linear.

Contudo, o algoritmo para obter um alinhamento ótimo visto anteriormente depende da matriz toda. Para remover esta dificuldade, a estratégia empregada é a de divisão e conquista, isto é, reduzir o problema dado a dois subproblemas menores e depois juntar as soluções destes subproblemas. Para obter mais informações sobre este algoritmo consulte [Meidanis, 94].

3.2. Extensões

Existem várias extensões do algoritmo básico que foi apresentado anteriormente. Estas extensões foram feitas para que o algoritmo se tornasse mais adequado para certas tarefas de comparação de seqüências. Será descrito aqui nesta seção três variações do algoritmo básico: a primeira se destina a situações onde não é desejado penalizar buracos nas extremidades das seqüências; a segunda penaliza a *abertura* de uma série de buracos mais do que o *aumento* da mesma; e a última trata de alinhamentos locais.

3.2.1. Buracos nas Extremidades

Entende-se por buracos nas extremidades aqueles que aparecem antes do primeiro ou depois do último caractere de uma seqüência.

No exemplo seguinte o buraco da primeira seqüência não está na extremidade, enquanto todos os outros buracos da segunda seqüência estão.

```
CAGCA - CTTGGATTCTCGG

- - - CAGCGTGG - - - - - - - -
```

Figura 31. Exemplos de buracos nas extremidades de seqüências.

Neste exemplo os comprimentos das seqüências são muito diferentes: uma é de tamanho 8, e a outra é de 18. Numa situação como esta haverá muitos buracos em qualquer alinhamento. Mas, se os buracos extremos forem desconsiderados, o alinhamento fica bastante bom, com 6 igualdades, 1 desigualdade, e 1 buraco.

Note que o alinhamento apresentado na figura anterior não é o melhor alinhamento para estas seqüências. O melhor está na figura a seguir:

```
CAGCA CTTGGATTCTCGG

CAGC-  - - - -G -T- - - -GG
```

Figura 32. Exemplo do melhor alinhamento das seqüências.

Apesar deste alinhamento ser ótimo (tem pontuação melhor que o outro) e de ter casado todos os caracteres da seqüência inferior, ele não é tão interessante do ponto de vista de encontrar semelhanças entre seqüências. A seqüência inferior foi simplesmente mutilada arbitrariamente por buracos.

Por este motivo existe uma variação do algoritmo básico que despreza buracos nas extremidades. Para uma descrição dele, consulte [Meidanis, 94].

3.2.2. Penalidades Para Buracos

Acredita-se que, em se tratando de mutações, a criação de uma série de buracos consecutivos é mais provável do que a criação do mesmo número de buracos isolados. Isto porque a série de buracos pode ser derivada de uma única mutação que removeu um trecho consecutivo de resíduos, enquanto que buracos isolados são advindos provavelmente de mutações distintas. Sendo mais comum a ocorrência de um único evento do que a ocorrência simultânea de vários eventos, conclui-se que n buracos aglutinados são mais comuns que n buracos separados.

Por este motivo há uma função sub-aditiva para penalizar uma série de buracos e uma variação do algoritmo básico que faz distinção entre buracos aglutinados e isolados. Esta variação foi proposta pela primeira vez por Needleman e Wunsch [Needleman, 70].

3.2.3. Comparações Locais

Até nesta seção foram descritos algoritmos que tratam de **alinhamento global**. Isto significa que o alinhamento é calculado levando em consideração o comprimento total das seqüências A e B que estão sendo comparadas. O algoritmo que será apresentado nesta seção trata de **alinhamento local**. Neste caso é localizado o melhor alinhamento entre sub-regiões de A e B. Pode existir um grande número de alinhamentos locais distintos. Veja na figura a seguir um esquema do alinhamento local e global. Esta figura foi obtida de [Barton, 96].

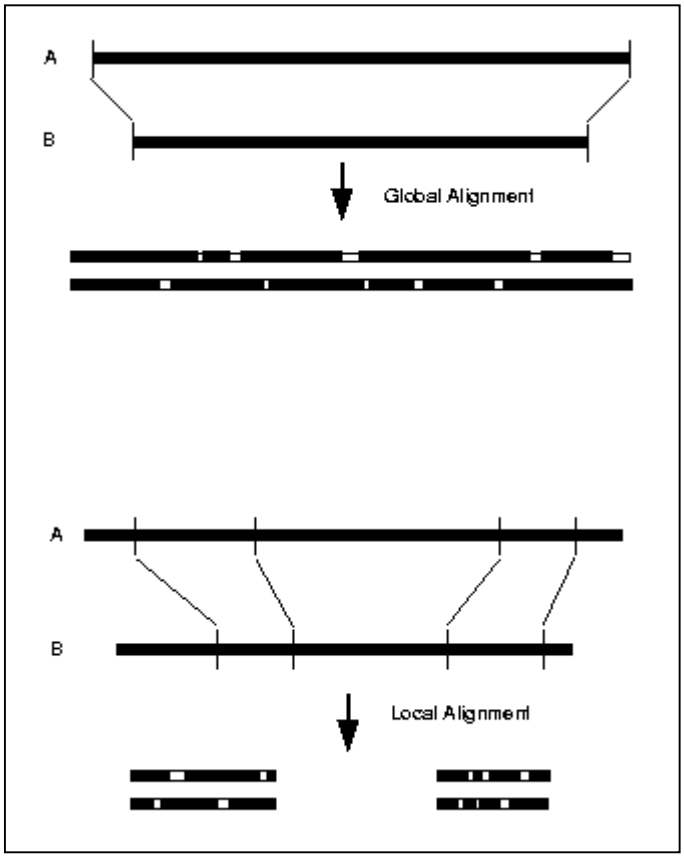


Figura 33. Alinhamento global e local [Barton, 96].

Outra definição possível para alinhamento local de [Meidanis, 94] é a seguinte: "um **alinhamento local** entre s1 e s2 é um alinhamento entre um fator de s1 e um fator de

s2". Quando se trata de alinhamento local, geralmente é desejável o alinhamento ótimo e outros com pontuação próxima do ótimo.

Existe uma variação do algoritmo básico para tratar alinhamentos locais. Ele usa a mesma matriz de pontuações. A inicialização da primeira linha e primeira coluna se faz com zeros. Não haverá elementos negativos nas células, pois no pior caso há sempre o alinhamento entre os sufixos vazios de s1[1..i] e s2[1..j], que dá uma pontuação igual a zero. Para o cálculo das outras células usa-se a seguinte fórmula:

```
a[i,j] = max(       0,
                    a[i,j-1] + g,
                    a[i-1,j-1] + p(i,j)
                    a[i-1,j] + g)
```

Figura 34. Fórmula para cálculo das células da matriz no algoritmo de alinhamentos locais.

Esta fórmula é a mesma usada no algoritmo básico, exceto pelo 0.

Quando a matriz estiver preenchida, a maior pontuação da matriz indica o valor do alinhamento local ótimo. O restante do alinhamento é obtido voltando-se por células que geraram os máximos até que se encontre uma célula de valor zero.

Após encontrar o alinhamento ótimo, podem ser também retornados outros alinhamentos com pontuação alta. Procura-se a célula com maior valor que ainda não tenha sido usada em nenhum alinhamento já obtido. Em seguida, aplica-se o procedimento de volta a partir desta célula até uma posição com valor zero, construindo um novo alinhamento.

4. Algoritmos de Comparação de Biosseqüências em Bases de Dados

Durante os anos 80, Lipman, Pearson e Wilbur descreveram em detalhes heurísticas usadas em seus programas para buscas em bases de biosseqüências [Wilbur, 83][Lipman, 85] [Pearson, 88]. O primeiro programa a surgir foi o FASTP, que faz buscas com proteínas. A seguir apareceu uma versão para seqüências de nucleotídeos, FASTN. Posteriormente ambos foram juntados num único programa chamado FASTA. Estes programas efetuam comparações locais e retornam apenas um alinhamento local - o considerado melhor. Mais tarde, programas que também obtém vários alinhamentos locais (LFASTA, PLFASTA) foram incorporados a família. Um sumário destes programas encontra-se nas referências [Pearson, 90]. Um estudo extenso sobre a **sensibilidade** (capacidade de detectar homologias remotas) e **seletividade** (capacidade de detectar falsas homologias) de FASTA foi empreendido por Pearson [Pearson, 91].

Durante a década de 90 surgiram os programas BLAST (Basic Local Alignment Search Tool). Como na família FAST, o BLAST possui versões para proteínas (BLASTP) e ácidos nucléicos (BLASTN).

O algoritmo BLAST foi desenvolvido por Altschul, Gish, Miller, Myers e Lipman [Altschul, 90]. A motivação para o desenvolvimento de BLAST foi a necessidade de aumentar a velocidade do FASTA.

4.1. FASTP

Esta seção tem como objetivo principal descrever o algoritmo FASTP. Ele foi o primeiro algoritmo da família FAST desenvolvido e é muito importante para o entendimento dos outros algoritmos que surgiram depois, como o FASTN, FASTA, LFASTA, TFASTA, e PLFASTA.

4.1.1. Introdução

Freqüentemente a única informação disponível sobre uma proteína é a sua seqüência de aminoácidos. Esta informação tem se tornado mais importante por causa da disponibilidade dos bancos de dados de seqüências de proteínas, e também pelas descobertas crescentes de relacionamentos entre uma nova seqüência de proteína e várias classes de proteínas nestes bancos de dados [Lipman, 85].

Como já foi visto anteriormente, a maioria dos algoritmos de similaridade de seqüência de proteína e DNA comparam cada nucleotídeo ou aminoácido de uma seqüência com todos os resíduos da segunda seqüência. Com estes algoritmos, a comparação de seqüências de 200 aminoácidos com uma biblioteca de proteínas com 500.000 resíduos necessita de aproximadamente 10^8 comparações [Lipman, 85].

Antes de 1985, identificar novas proteínas através de procura em bancos de dados era muito difícil. Os programas de computadores necessitavam de várias horas em um microcomputador. Um dos programas mais rigorosos de comparação de seqüências de aminoácidos desta época, SEQHP [Goad, 82], necessitava mais de 8 horas para comparar uma proteína de 200 resíduos com uma biblioteca de 500.000 resíduos de proteínas do *National Biomedical Research Foundation* (NBRF) em um computador VAX 11/750 [Lipman, 85].

Wilbur e Lipman [Wilbur, 83] descreveram um algoritmo em 1983 que permitiu uma procura rápida em bancos de dados de proteínas e nucleotídeos focando-se somente em um grupo de identidades entre as duas seqüências, o que requer, conseqüentemente, menos comparações em uma procura.

Lipman e Pearson [Lipman, 85] descreveram uma modificação deste algoritmo em 1985 que melhorou sua sensibilidade e eficiência. Este algoritmo foi chamado de FASTP. Nele uma seqüência de 200 resíduos podia ser comparada a biblioteca do NBRF em 2 a 5 minutos em um VAX 11/750.

O FASTP foi modificado para rodar em uma variedade de computadores incluindo o IBM PC. Ele foi escrito na linguagem de programação C, originalmente em um VAX 11/780 com sistema operacional Unix. Posteriormente foi feita uma substituição para o sistema operacional VAX/VMS e um microcomputador IBM PC [Lipman, 85].

A memória necessária para o programa é modesta, mas é necessário espaço em disco para armazenar a biblioteca de proteínas [Lipman, 85].

4.1.2. Descrição do Algoritmo

Simplificadamente é possível dizer que o FASTP possui 3 passos. Estes passos serão descritos a seguir.

Passo 1. Identificar regiões compartilhadas pelas seqüências com mais alta densidade de identidades (*k-tupla* = 1) ou pares de identidades (*k-tupla* = 2).

O primeiro passo usa uma técnica rápida, similar a descrita por [Wilbur, 83], para descobrir identidades entre as seqüências. FASTP obteve maior velocidade e seletividade neste primeiro passo usando uma **tabela de busca** para localizar todas as identidades ou grupos de identidades entre as duas seqüências de aminoácidos [Pearson, 90].

Uma tabela de busca é um método rápido de se descobrir a posição de um resíduo em uma seqüência. Uma forma de encontrar o "A" em uma seqüência "NDAPL" é comparar o "A" com cada resíduo da seqüência. Uma maneira mais rápida (se muitos resíduos tiverem que ser comparados) é fazer uma tabela de todos os resíduos possíveis (23 no caso de proteínas) tal que a representação computacional para o resíduo é a mesma que sua posição na tabela. O valor que é colocado na tabela indica onde o resíduo está presente na seqüência, se ele estiver presente. Para este exemplo, a tabela tem o valor 1 na posição 3 ("N" é o terceiro aminoácido da tabela e o primeiro da seqüência), 2 na posição 4 ("D" é o quarto aminoácido da tabela e o segundo da seqüência), 3 na posição 1, 4 na 15, 5 na 11, e as outras 18 posições são zero. A presença e a posição do "A" na seqüência pode ser determinada em um único passo olhando-se na primeira posição da tabela [Pearson, 90].

Veja a seguir a tabela de busca para a seqüência NDAPL.

	Aminoácido	**Posição do aminoácido na seqüência**
1	A	3
2	R	0
3	N	1
4	D	2
5	C	0
6	Q	0
7	E	0
8	G	0
9	H	0
10	I	0

11	L	5
12	K	0
13	M	0
14	F	0
15	P	4
16	S	0
17	T	0
18	W	0
19	Y	0
20	V	0
21	B	0
22	Z	0
23	X	0

Figura 35. Tabela de busca da seqüência NDAPL.

O parâmetro *k-tupla* determina quantas identidades consecutivas são necessárias em um emparelhamento de resíduos. O valor 2 de *k-tupla* é freqüentemente utilizado para comparação de seqüências de proteínas, o que significa que o programa examina somente as porções das duas seqüências que têm no mínimo dois resíduos adjacentes idênticos em ambas seqüências. Procuras mais sensíveis podem ser feitas com *k-tupla* igual a 1 [Pearson, 90].

O parâmetro *k-tupla* está relacionado com o tamanho da tabela de busca. Se *k-tupla* é igual a 1, então a tabela de busca deve ter 23 linhas, uma para cada aminoácido. Se *k-tupla* é igual a 2, então a tabela deve ter 23^2, uma para cada par de aminoácidos. Analogamente se *k-tupla* é igual a 3, a tabela deve ter 23^3, uma para cada trio de aminoácidos. Este raciocínio é aplicado a qualquer valor de *k-tupla*.

Além da tabela de busca, é usado o **método da diagonal** para encontrar todas as regiões de similaridade entre as duas seqüências, contando os emparelhamentos de comprimento *k-tupla* e penalizando as desigualdades. Este método identifica as regiões da diagonal que tem densidades mais altas de emparelhamentos com comprimento *k-tupla*. O termo diagonal refere-se a linha diagonal que é vista em uma *dot matrix plot* [Maizel, 81] quando

uma seqüência é comparada com ela mesma, e ela caracteriza um alinhamento sem buracos entre duas seqüências. Por exemplo, o alinhamento do resíduo 8 de uma seqüência no eixo x e o resíduo 13 em outra seqüência no eixo y especifica uma diagonal, o alinhamento do resíduo 16 de uma com o 21 da outra cai na mesma diagonal, como 27 e 32, e assim por diante [Pearson, 90]. Note que em todos os casos do exemplo a diferença entre a posição do resíduo da segunda seqüência com o resíduo primeira da seqüência é 5 ((13-8 = 5), (21-16 = 5), (32-27 = 5)).

A figura a seguir mostra uma fórmula usada por FASTP para identificar porções da diagonal com densidade mais alta de identidades. Cada porção é chamada de **região local de similaridade**, ou, simplesmente, região.

```
Para cada grupo k-tupla de resíduos na seqüência da biblioteca{

      A. Para cada posição na seqüência de consulta com resíduos idênticos{

         1. Calcule a diagonal baseada na posição corrente das seqüências
            da biblioteca e de consulta.
            (diagonal = posição_biblioteca - posição_consulta)

         2. Verifique se alguma posição inicial foi encontrada nesta
            diagonal.

         3. Se nenhuma região foi encontrada nesta diagonal antes, salve a
            posição corrente como o início da região com pontuação inicial
            16 (para k-tupla = 2 em FASTP).

         4. Senão{

                  a. Se já existia uma região nesta diagonal, calcule a
                     distância do fim da região anterior até o
                     emparelhamento k-tupla corrente.

                  b. Se a distância é maior que a pontuação da região
                     anterior, salve a região anterior e inicie uma nova
                     região.

                  c. Senão{

                        i.    Salve a pontuação da região anterior.

                        ii.   Estenda a região anterior até o
                              emparelhamento k-tupla corrente.

                        iii.  Diminua a pontuação da região anterior pela
                              distância que a região foi estendida, e
                              aumente-a de 16 pelo novo emparelhamento k-
                              tupla.
                  }
            }
      }
}

Em cada momento que se salva uma região é verificado se a pontuação da
região é melhor que a menor pontuação das 10 melhores regiões que já foram
salvas anteriormente. Se for melhor, substitui-se a região salva com menor
pontuação pela nova região e descobre-se a nova região com menor pontuação
na lista de regiões atualizadas.
```

Figura 36. Fórmula do primeiro passo do algoritmo FASTP [Pearson, 90].

Para *k-tupla* igual a 2, após a execução desta fórmula, um grupo de emparelhamentos de aminoácidos separados por menos que 16 resíduos seriam combinados dentro de uma região, e os emparelhamentos separados por mais de 30 resíduos da região anterior iniciaria uma nova região [Pearson, 90].

Para ilustrar o funcionamento desta fórmula será apresentado o exemplo da comparação das seqüências FLWRTWS e SWKTWT. Suponha que *k-tupla* seja igual a 1, e que a

seqüência FLWRTWS é uma seqüência nova que deve ser comparada a um conjunto de seqüências de um banco de dados, sendo SWKTWT uma das seqüências pertencentes a este conjunto. A figura a seguir apresenta esta situação.

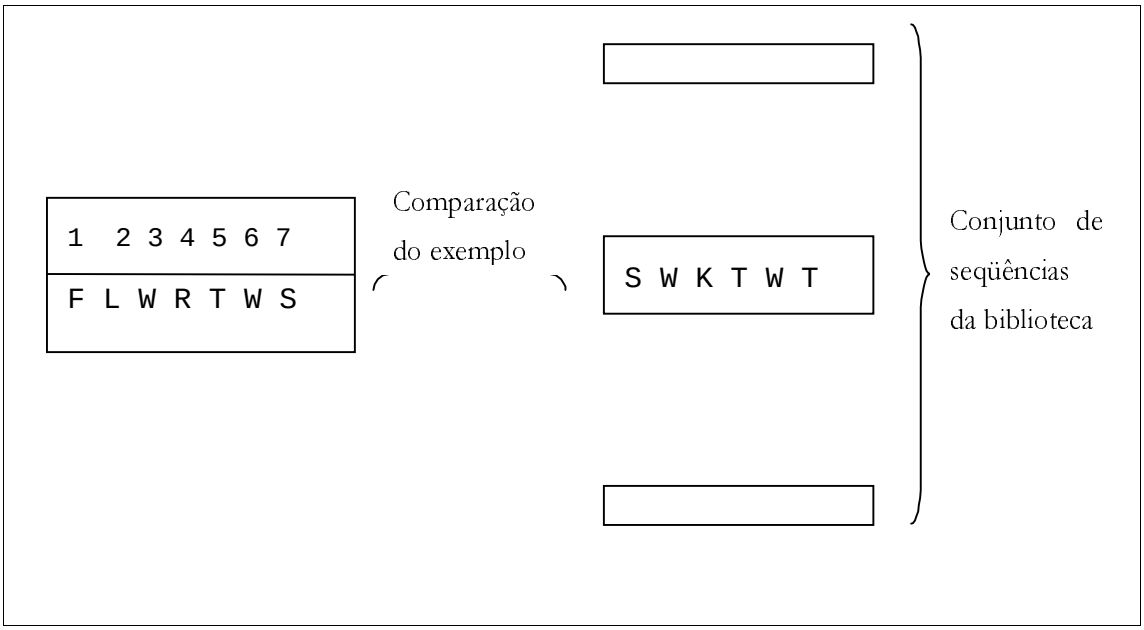


Figura 37. Exemplo de comparação das seqüências FLWRTWS e SWKTWT.

Será necessário analisar cada grupo *k-tupla* de resíduos da seqüência da biblioteca. Como o exemplo trata de *k-tupla* igual a 1, os resíduos S, W, K, T, W, T serão analisados individualmente.

Antes de iniciar a execução da fórmula deve ser construída a tabela de busca da seqüência FLWRTWS. A seguir está uma figura com um esquema simplificado dela.

Aminoácidos	Posições na seqüência FLWRTWS
…	0
R	4
…	0
L	2
…	0
F	1

…	0
S	7
T	5
W	3,6
…	0

Figura 38. Esquema simplificado da tabela de busca da seqüência FLWRTWS.

O primeiro resíduo a ser analisado é o S, que se localiza na posição 1 da seqüência da biblioteca. Ele deve ser comparado a cada posição que possuir resíduos idênticos da seqüência nova FLWRTWS. Pela tabela de busca S está na posição 7 da seqüência nova. A diagonal neste caso é -6 (1-7). Nenhuma região foi encontrada com esta diagonal anteriormente e por isso esta posição é marcada como o início de uma nova região com pontuação 16. Este valor de pontuação foi escolhido apenas como exemplo. Na verdade o algoritmo utiliza o valor 16 quando *k-tupla* é igual a 2, que não é o caso aqui (o valor de *k-tupla* usado aqui é 1). Veja a figura a seguir.

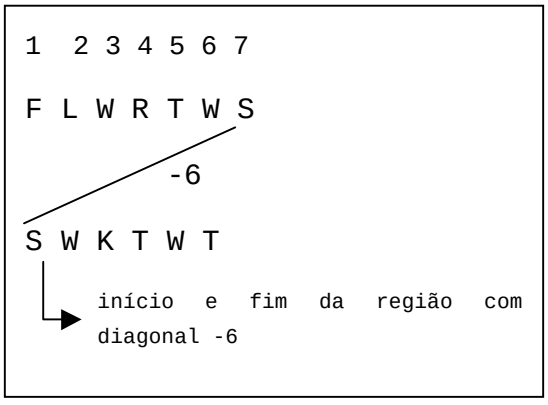


Figura 39. Diagonal -6 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

O segundo resíduo a ser analisado é o W, que se localiza na posição 2 da seqüência da biblioteca. Ele deve ser comparado a cada posição que possuir resíduos idênticos da seqüência nova FLWRTWS. Pela tabela de busca W está nas posições 3 e 6 da seqüência nova. As diagonais nestes casos são -1 (2-3) e -4 (2-6). Nenhuma região foi encontrada com estas diagonais anteriormente e por isso estas posições são marcadas como inícios de novas regiões com pontuação 16.

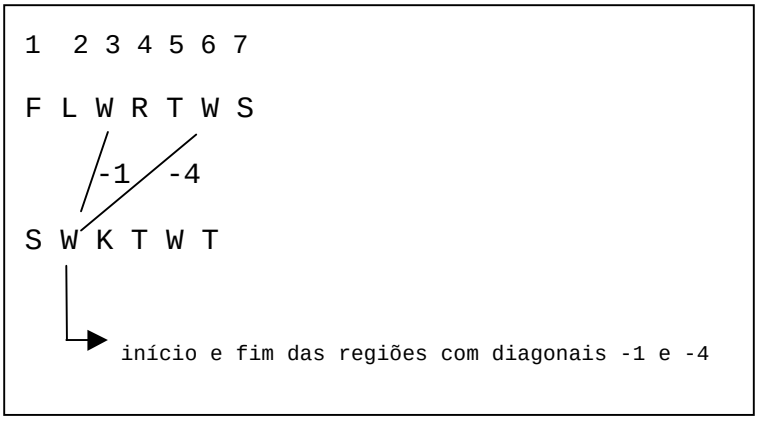


Figura 40. Diagonais -1 e -4 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

O terceiro resíduo a ser analisado é o K, que se localiza na posição 3 da seqüência da biblioteca. Como não há nenhuma posição da seqüência nova idêntica a este resíduo, ele é ignorado.

O quarto resíduo a ser analisado é o T, que se localiza na posição 4 da seqüência da biblioteca. Ele deve ser comparado a cada posição que possuir resíduos idênticos da seqüência nova FLWRTWS. Pela tabela de busca T está na posição 5 da seqüência nova. A diagonal neste caso é -1 (4-5).

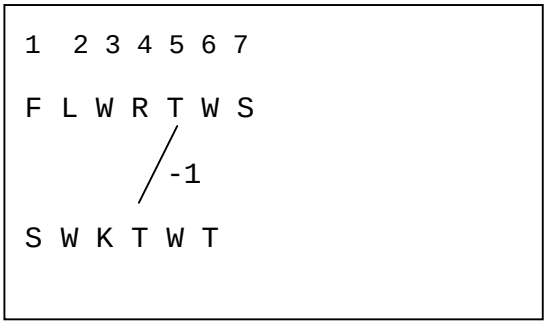


Figura 41. Diagonal -1 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

Uma região já foi encontrada com esta diagonal anteriormente. Neste caso deve ser calculada a distância da região anterior até esta posição. No caso esta distância equivale a 2, e não é maior que a pontuação da região anterior, que é 16. Veja a figura a seguir.

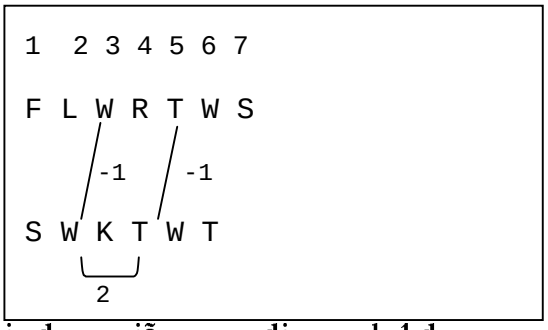


Figura 42. Distância das regiões com diagonal -1 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

Neste caso a região anterior será estendida até o emparelhamento encontrado.

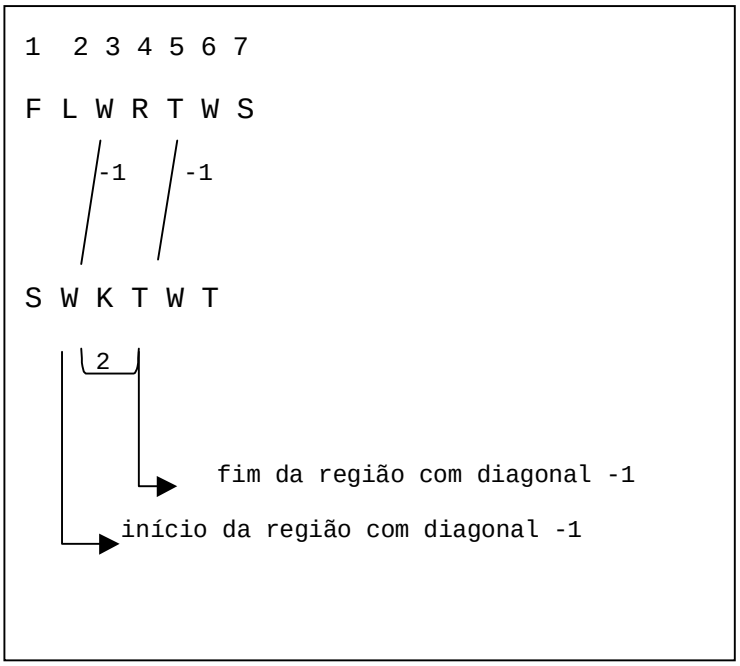


Figura 43. Estendendo a região com diagonal -1 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

A pontuação da região é diminuída da distância, 2, e é acrescida de 16. Neste caso a pontuação fica igual a 30 (16-2+16).

O quinto resíduo a ser analisado é o W, que se localiza na posição 5 da seqüência da biblioteca. Ele deve ser comparado a cada posição que possuir resíduos idênticos da seqüência nova FLWRTWS. Pela tabela de busca W está nas posições 3 e 6 da seqüência nova. As diagonais nestes casos são 2 (5-3) e -1 (5-6). Não foi encontrada anteriormente

nenhuma região com diagonal 2, e por isso esta posição marca o início de uma nova região com pontuação 16. Veja a figura a seguir.

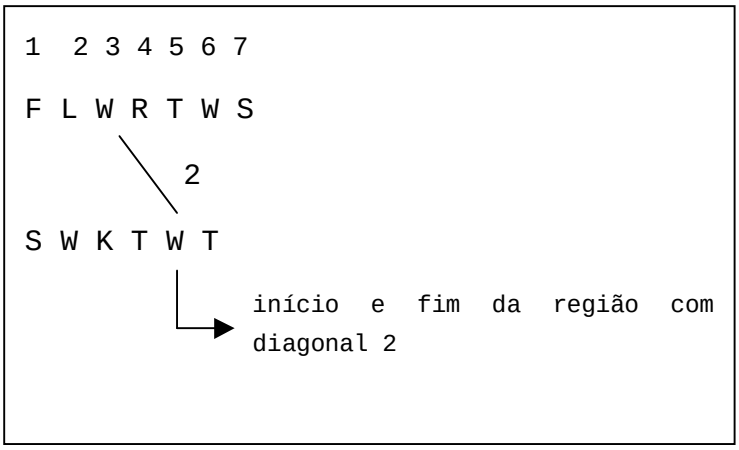


Figura 44. Diagonal 2 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

Com diagonal -1 já foi encontrada região anteriormente. A distância da região anterior até este emparelhamento equivale a 1 e não é maior que a pontuação da região anterior, que é 30. Neste caso a região anterior é estendida até este emparelhamento e a pontuação da nova região é atualizada para 45 (30-1+16). Veja o esquema a seguir.

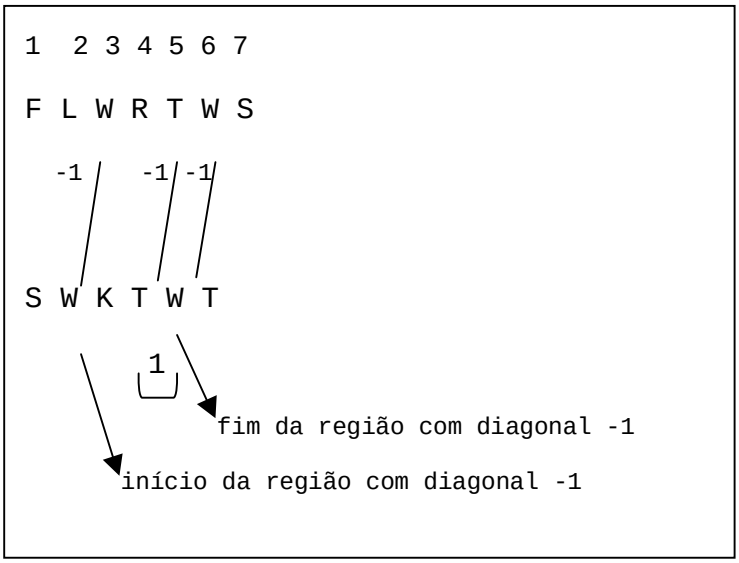


Figura 45. Estendendo a região com distância -1 do exemplo da comparação das seqüências FLWRTWS e SWKTWT.

O sexto e último resíduo a ser analisado é o T, que se localiza na posição 6 da seqüência da biblioteca. Ele deve ser comparado a cada posição que possuir resíduos idênticos da seqüência nova FLWRTWS. Pela tabela de busca T está na posição 5 da seqüência nova. A diagonal neste caso é +1 (6-5). Nenhuma região foi encontrada com esta diagonal anteriormente e por isso esta posição é marcada como o início de uma nova região com pontuação 16.

Neste exemplo simples foram encontradas 5 regiões. As regiões com diagonais -6, -4, 2, e +1 com pontuação igual a 16, e a região com diagonal -1 e pontuação 45. Veja o esquema a seguir.

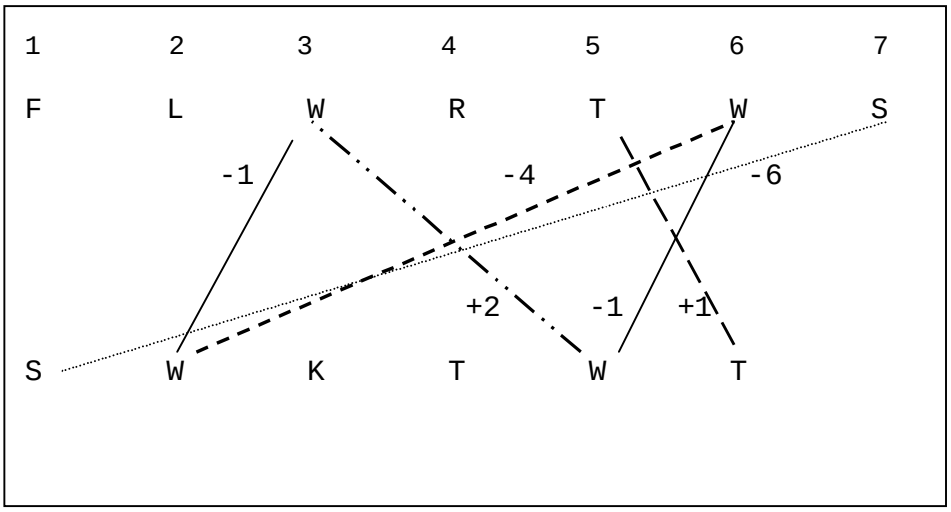


Figura 46. Esquema das diagonais encontradas no exemplo da comparação das seqüências FLWRTWS e SWKTWT.

O gráfico a seguir representa os emparelhamentos encontrados no exemplo. A seqüência FLWRTWS se localiza no eixo y, enquanto a seqüência SWKTWT se localiza no eixo x. Logo as posições 1, 2, 3, 4, 5, 6, e 7 do eixo y representam F, L, W, R, T, W, e S respectivamente. Analogamente as posições 1, 2, 3, 4, 5, e 6 do eixo x representam S, W, K, T, W, e T respectivamente. Os pontos no gráfico representam os emparelhamentos. Veja por exemplo que o resíduo W da posição 2 de SWKTWT se emparelha com os resíduos das posições 3 e 6 de FLWRTWS. Note também a diagonal que é formada pelos pontos (2,3), (4,5), e (5,6). Esta é a diagonal -1 que corresponde à região com pontuação mais alta do exemplo.

Esta foi a descrição do passo 1 do algoritmo. Ele gerará como resultado as dez regiões com mais alta pontuação. Note que o exemplo dado, por ser muito simples, só gerou como resultado cinco regiões.

Passo 2. Examinar as dez regiões de mais alta densidade de identidades usando uma matriz de pontuações. Cada região desta é um alinhamento parcial sem introdução de buracos.

No caso de proteínas, após a descoberta das dez regiões de mais alta densidade de identidades no primeiro passo, é comum que estas regiões sejam examinadas no segundo passo utilizando a matriz PAM250. A matriz PAM (do inglês *Percent of Accepted Mutation*) estabelece um peso para cada casamento entre aminoácidos baseando-se em estatísticas de mutações envolvendo proteínas reconhecidamente homólogas. Gonnet [Gonnet, 92] descreve em detalhe a derivação destas matrizes e seu uso em comparação de proteínas, acrescentando que este é o método mais bem fundamentado para alinhar polipeptídeos [Meidanis, 94]. Os seguintes fatos são levados em consideração na matriz PAM250:

- Os aminoácidos idênticos alinhados raros recebem pontuação mais alta do que os aminoácidos idênticos alinhados mais comuns;
- As substituições que ocorrem mais freqüentemente na evolução recebem pontuações positivas, enquanto as substituições mais improváveis recebem pontuações negativas.

Nesta etapa do algoritmo não são consideradas as inserções e remoções. É tirado proveito do fato de que as substituições ocorrem com mais freqüência do que as inserções e remoções. Isto faz com que o tempo de computação e os requisitos de memória sejam bem reduzidos.

Continuando o exemplo da comparação das duas seqüências FLWRTWS e SWKTWT, é possível calcular para cada região retornada no primeiro passo a pontuação de acordo com a matriz PAM250. No caso da diagonal -1 devem ser considerados os alinhamentos dos resíduos W|W, K|R, T|T, e W|W. Pela PAM250 as pontuações destes alinhamentos são 17, 3, 3, e 17, respectivamente. Logo a pontuação total é 40 (17+3+3+17). No caso da diagonal -6 deve ser considerado o alinhamento S|S, que pela matriz tem pontuação 2. As diagonais -4 e 2 correspondem ao alinhamento de W|W,

que tem pontuação 17. Por último, a diagonal +1 corresponde ao alinhamento de T|T, cuja pontuação pela matriz é 3.

Se o exemplo fosse maior todas as dez regiões teriam que ser consideradas. Como este é um exemplo simples apenas as cinco regiões existentes foram consideradas.

A pontuação mais alta das dez regiões baseada na matriz PAM250 é usada como a pontuação de similaridade entre as duas seqüências. Esta é a chamada pontuação inicial ou ***init1***.

Durante uma procura em um banco de dados, a pontuação inicial é calculada entre a seqüência nova (a seqüência de consulta) e cada uma das seqüências do banco de dados. As pontuações iniciais são apresentadas em um histograma e uma média e um padrão de desvio são calculados. As seqüências são então classificadas pelas suas pontuações iniciais.

Passo 3. Uma pontuação otimizada, que permite remoções e inserções, é calculada para as similaridades maiores da classificação [Lipman,85]. O alinhamento otimizado usa uma modificação do primeiro método descrito por Needleman e Wunsch [Needleman, 70] com refinamentos subsequentes [Goad, 82], [Sellers, 74], [Sankoff, 72], [Smith, 81]. Esta comparação final considera todos os alinhamentos possíveis da seqüência de consulta e da biblioteca que estão dentro de uma banda com largura de 32 resíduos centralizada em torno da região inicial com mais alta pontuação [Pearson, 88][Pearson, 90].

4.2. FASTN

O programa FASTP executa consultas em bibliotecas de proteínas. Depois de pouco tempo que o FASTP foi desenvolvido, William R. Pearson fez uma modificação no algoritmo que permitiu comparações de seqüências de DNA, e disponibilizou o programa, chamado FASTN.

Vale ressaltar aqui que, para comparações de DNA, o valor de *k-tupla* pode variar de 1 a 6, sendo recomendável de 4 a 6 [Pearson, 90].

4.3. FASTA

Esta seção tem como objetivo apresentar o algoritmo FASTA que foi desenvolvido após o FASTP e FASTN.

4.3.1. Introdução

O FASTA é uma versão melhorada do FASTP que combina as funções do FASTP e do FASTN e provê um algoritmo de comparação de seqüência mais sensível. O FASTA compara uma seqüência de proteína com outra seqüência de proteína ou com um banco de dados de proteína, ou compara uma seqüência de DNA com outra seqüência de DNA ou com um banco de dados de DNA.

4.3.2. Descrição do Algoritmo

O FASTA executa os passos 1 e 2 da mesma forma que o FASTP.

Revisando, no passo 1 são identificadas as 10 regiões compartilhadas pelas duas seqüências com mais alta densidade de identidades (*k-tupla* = 1) ou pares de identidades (*k-tupla* = 2), e no passo 2 são examinadas estas 10 regiões com uma matriz de pontuação (no caso de seqüências de proteínas pode ser usada a PAM250). Cada região desta é um alinhamento parcial sem buracos. Neste momento o FASTP determina a região inicial com maior pontuação encontrada, chamada de *init1*, e executa o passo 3.

O FASTA executa um passo a mais antes do passo 3 do FASTP. Para aumentar a pontuação, ele verifica quais regiões iniciais podem ser agrupadas em um único alinhamento [Pearson, 90]. Para que uma região inicial possa participar do agrupamento, ela deve ter pontuação maior que um valor *cutoff* (valor limite determinado empiricamente). Dadas as localizações das regiões iniciais, suas respectivas pontuações, e uma penalidade para o agrupamento (análogo à penalidade para buracos), o FASTA calcula um alinhamento ótimo entre a seqüência nova e as regiões iniciais (com pontuações maiores que *cutoff*) agrupadas da seqüência da biblioteca. Este alinhamento é calculado através de um algoritmo de programação dinâmica similar ao descrito por [Wilbur, 83]. A pontuação resultante deste alinhamento ótimo é conhecida como ***initn***.

Depois de uma pesquisa completa na biblioteca, o FASTA apresenta as pontuações iniciais *initn* de cada seqüência da biblioteca em um histograma, calcula a média de *initn* entre todas as seqüências da biblioteca, e determina o desvio padrão.

A pontuação inicial (*initn*) é usada para classificar as seqüências da biblioteca. Depois desta classificação, o FASTA executa seu último passo da comparação, que é igual ao terceiro passo do FASTP. Isto é, as seqüências da biblioteca com maiores pontuações (*initn*, no caso do FASTA, e *init1* no caso do FASTP) são alinhadas usando uma modificação do método de otimização NWS (*Needleman-Wunch-Sellers algorithm*) [Needleman, 70] [Sellers, 74] [Smith,81].

O esquema a seguir resume o funcionamento do FASTP e do FASTA.

Entrada: Seqüência nova

Objetivo: Comparar a seqüência nova com todas as seqüências do banco de dados.

Para cada seqüência do banco de dados faz-se:

Passo 1: Entrada: Seqüência nova e uma seqüência do banco de dados.

Idéia: Encontra-se regiões que possuam identidades entre as seqüências comparadas. Calcula-se pontuações para estas regiões.

Palavras-chave: *k-tupla*, tabela de busca e método da diagonal.

Saída: 10 regiões cujas pontuações são maiores.

Passo 2: Entrada: As 10 regiões do passo 1.

Idéia: Analisa-se cada uma das regiões com uma matriz de pontuação. No caso das seqüências de proteínas, pode se usar a PAM250.

Saída: Pontuação destas 10 regiões de acordo com a matriz de pontuação. Estas regiões são alinhamentos parciais sem buracos. A pontuação melhor é chamada de *init1*.

Passo 3: Executado apenas pelo FASTA.

Entrada: 10 regiões do passo 2.

Idéia: Descobre-se quais regiões iniciais vindas do passo 2 podem ser agrupadas. Calcula-se o alinhamento ótimo destas regiões agrupadas, levando em consideração os buracos que foram inseridos. A pontuação resultante deste alinhamento é chamada de *initn*.

Passo 4: Classifica-se as seqüências do banco de dados. FASTP faz isso levando em consideração *init1*, e FASTA, *initn*.

Calcula-se a média de *init1* e *initn*.

Calcula-se o desvio padrão de *init1* e *initn*.

Apresenta um histograma.

Calcula-se o alinhamento ótimo da seqüência nova com as seqüências do banco de dados que obtiveram melhor classificação através do algoritmo NWS. Esta comparação final considera todas as possibilidades de alinhamentos da seqüência de consulta e da seqüência do banco de dados que estão dentro de uma certa banda centralizada ao redor da região inicial com maior pontuação encontrada no passo 2.

Figura 47. Esquema resumido do funcionamento de FASTP e FASTA.

A seguir estão figuras obtidas em [Barton, 96] que ilustram o funcionamento dos passos do FASTA. A figura (a) mostra o passo 1, a (b) mostra o passo 2, a (c) mostra o passo 3, e a (d) mostra o passo 4.

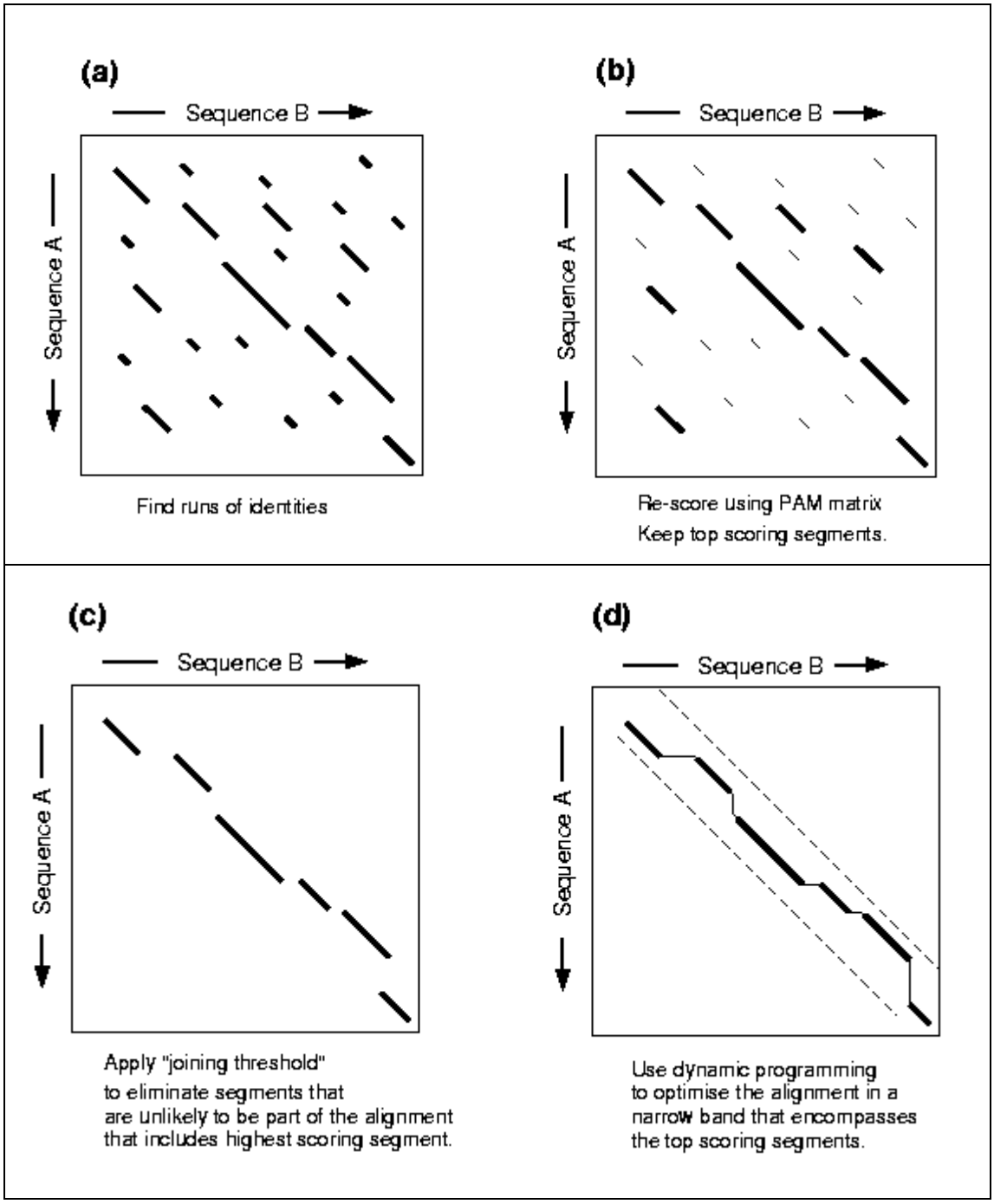


Figura 48. Passos 1, 2, 3 e 4 do FASTA [Barton, 96].

4.4. TFASTA

A famíla FAST inclui programas como o TFASTA, que permite a comparação de uma seqüência de proteína com uma seqüência banco de dados de seqüências de DNA. Para obter informações de TFASTA consulte [Pearson, 90].

4.5. LFASTA, PLFASTA

Os programas LFASTA e PLFASTA também fazem parte da família FAST. Enquanto o FASTA retorna um único alinhamento local, LFASTA mostra todas as regiões locais de similaridade compartilhadas entre as seqüências que estão sendo comparadas. PLFASTA é idêntico ao LFASTA, só que apresenta os alinhamentos em uma forma similar a *dot matrix plot* . Assim como os outros algoritmos da família, todos são variações do FASTP. Para obter mais informações sobre LFASTA e PLFASTA veja [Pearson, 90].

4.6. BLAST

Nesta seção será descrito o algoritmo BLAST (*Basic Local Alignment Search Tool)* [Altschul, 90][Altschul, 97]. O método detectará similaridades fracas de seqüências, mas que possuem significado biológico, e é mais rápido do que os outros algoritmos baseados em heurísticas existentes.

Os programas da família BLAST foram feitos para se comparar seqüências de consultas de DNA ou proteína com bancos de dados de DNA ou proteína em qualquer combinação.

4.6.1. Idéia Geral do Algoritmo

Como foi falado anteriormente, existem dois tipos básicos de medidas de similaridades de seqüências. Geralmente elas são classificadas como globais ou como locais. Algoritmos de similaridade global otimizam o alinhamento total das duas seqüências, o que pode incluir grandes trechos com baixa similaridade [Needleman, 70]. Algoritmos de similaridade local consultam apenas subseqüências. Uma simples comparação pode produzir vários alinhamentos de subseqüências distintos e regiões não conservativas que não contribuem para a medida da similaridade [Smith2, 81][Goad, 82][Sellers, 84].

Geralmente medidas de similaridades locais são preferidas em procuras em bancos de dados.

Ao compararmos duas proteínas poderão existir regiões não conservadas, ou seja regiões em que os resíduos presentes em uma não estão conservados na outra (isto é, não são os mesmos). Mas existem alguns casos em que mesmo com resíduos diferentes, a região é considerada conservativa. Isto acontece porque os aminoácidos são classificados em grupos de acordo com suas características. Existem aminoácidos polares, apolares, ácidos, etc.. Uma modificação que leve um aminoácido polar para outro polar é chamada de conservativa porque mantém o tipo do aminoácido, já a não-conservativa muda de tipo (apolar→polar, ácido→básico, etc.).

A família BLAST, como a FAST, utiliza o método de similaridade local.

Muitas medidas de similaridades, incluindo a do BLAST, começam com uma matriz de pontuações para todos os pares de resíduos. As identidades e as substituições conservativas tem pontuações positivas, enquanto as substituições não comuns têm pontuações negativas. Para comparações de seqüências de aminoácidos, geralmente é usada a matriz PAM-120 (uma variação de [Dayhoff, 78]), enquanto para comparações de seqüências de DNA as pontuações das identidades são iguais a +5, e as desigualdades -4; sendo possível, logicamente, a utilização de outras pontuações.

Um **segmento de seqüência** é um trecho de resíduos de qualquer tamanho. A pontuação de similaridade para dois segmentos alinhados do mesmo tamanho é a soma dos valores das similaridades para cada par de resíduos alinhados.

Um **par maximal de segmentos**, ou MSP de *Maximal Segment Pair,* é um par de segmentos, de tamanho idêntico de duas seqüências que possui a maior pontuação. O MSP pode ter qualquer tamanho, e sua pontuação, calculada pelo BLAST, provê uma medida de similaridade local para qualquer par de seqüências.

Um biólogo molecular, no entanto, pode estar interessado em todas as regiões conservativas compartilhadas entre duas proteínas, e não somente no par que tiver maior pontuação. Por esta razão, o BLAST pesquisa todos os pares de segmentos maximais locais com pontuações maiores que um valor limite.

Em uma procura em um banco de dados com muitas seqüências, geralmente somente uma pequena quantidade, se existir, será homóloga à seqüência de consulta. Por isso o cientista está interessado em identificar as seqüências cujas pontuações de MSP estão

acima de uma pontuação-limite S. Neste conjunto de seqüências identificadas estão incluídas tanto aquelas que compartilham alta similaridade quanto aquelas que tem pontuações duvidosas.

Resultados de [Karlin, 90] e [Karlin2, 90] nos permitem estimar a maior pontuação de MSP S na qual há chance de que similaridades estão provavelmente aparecendo. Para acelerar as buscas no banco de dados, o BLAST minimiza o tempo gasto em regiões da seqüência onde a similaridade com a seqüência de consulta tem pouca chance de exceder o valor limite.

A idéia geral do BLAST é a seguinte. Seja uma par de palavras um par de segmentos de comprimento fixo *w*. Veja na figura a seguir o esquema.

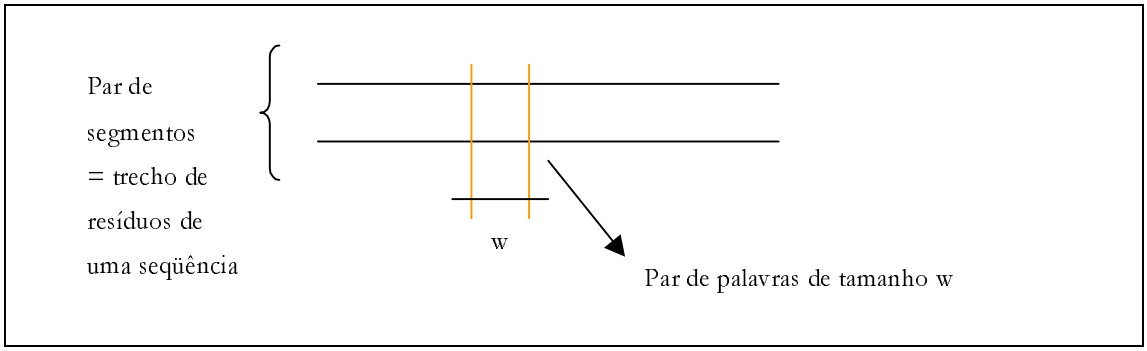


Figura 49. Par de segmentos e de palavras.

A principal estratégia do BLAST consiste em buscar pares de segmentos que contenham pares de palavras com pontuações de no mínimo T. Analisando a seqüência, é possível determinar se ela contém uma palavra de tamanho w que pode ser emparelhada com uma palavra da seqüência de consulta e produzir um par de palavras cuja pontuação é igual ou maior que um limite T. Qualquer acerto é estendido para determinar se ele está contido dentro de um par de segmentos no qual a pontuação é maior ou igual a S.

4.6.2. Passos do Algoritmo

Na implementação do BLAST, há alguns detalhes do algoritmo que variam de acordo com o banco de dados que está sendo procurado (se ele é de seqüências de proteínas ou de DNA). De qualquer forma, há três passos básicos independentes do banco de dados pesquisado, que são:

1. Compilar uma lista de palavras com alta pontuação;

2. Percorrer o banco de dados por acertos (*hits*); e

3. Estender os acertos.

A seguir será detalhado cada um destes passos. Como entrada principal do algoritmo tem-se uma seqüência de consulta, que é um segmento ou um trecho de resíduos. Esta seqüência nova deve ser comparada a todas as outras seqüências do banco de dados. Esta comparação utilizará uma matriz de pontuação, como os algoritmos da família FAST.

Passo 1. Para cada palavra de tamanho w da seqüência de consulta, determina-se quais palavras alinhadas com ela tem pontuação de no mínimo T. Desta maneira é gerada uma lista de palavras para cada palavra de tamanho w da seqüência de consulta.

Passo 2. Para cada palavra de cada lista construída no passo 1, percorre-se novamente o banco de dados até se encontrar os acertos relativos a tal palavra.

Passo 3. Para cada acerto encontrado, verifica se ele está dentro de um alinhamento cuja pontuação seja suficiente para ser notificada (pontuação deve ser maior ou igual a um valor S). Isto é feito estendendo o acerto em ambas direções, até que a pontuação de alinhamento atinja uma diferença de X abaixo da pontuação máxima já alcançada. Estes acertos estendidos são os chamados MSPs.

O passo 3 produz como saída os MSPs encontrados.

Estes passos estão esquematizados na figura a seguir, que foi retirada de [Barton, 96].

Quanto menor o valor de T, menor a possibilidade de perder MSPs com a pontuação S requisitada, no entanto valores baixos de T também aumentam o tamanho da lista de acertos geradas no passo 1 e conseqüentemente o tempo de execução e a memória necessária. Na prática o algoritmo deve se comprometer com valores de T e X para balancear as necessidades do processador e a sensibilidade [Barton, 96].

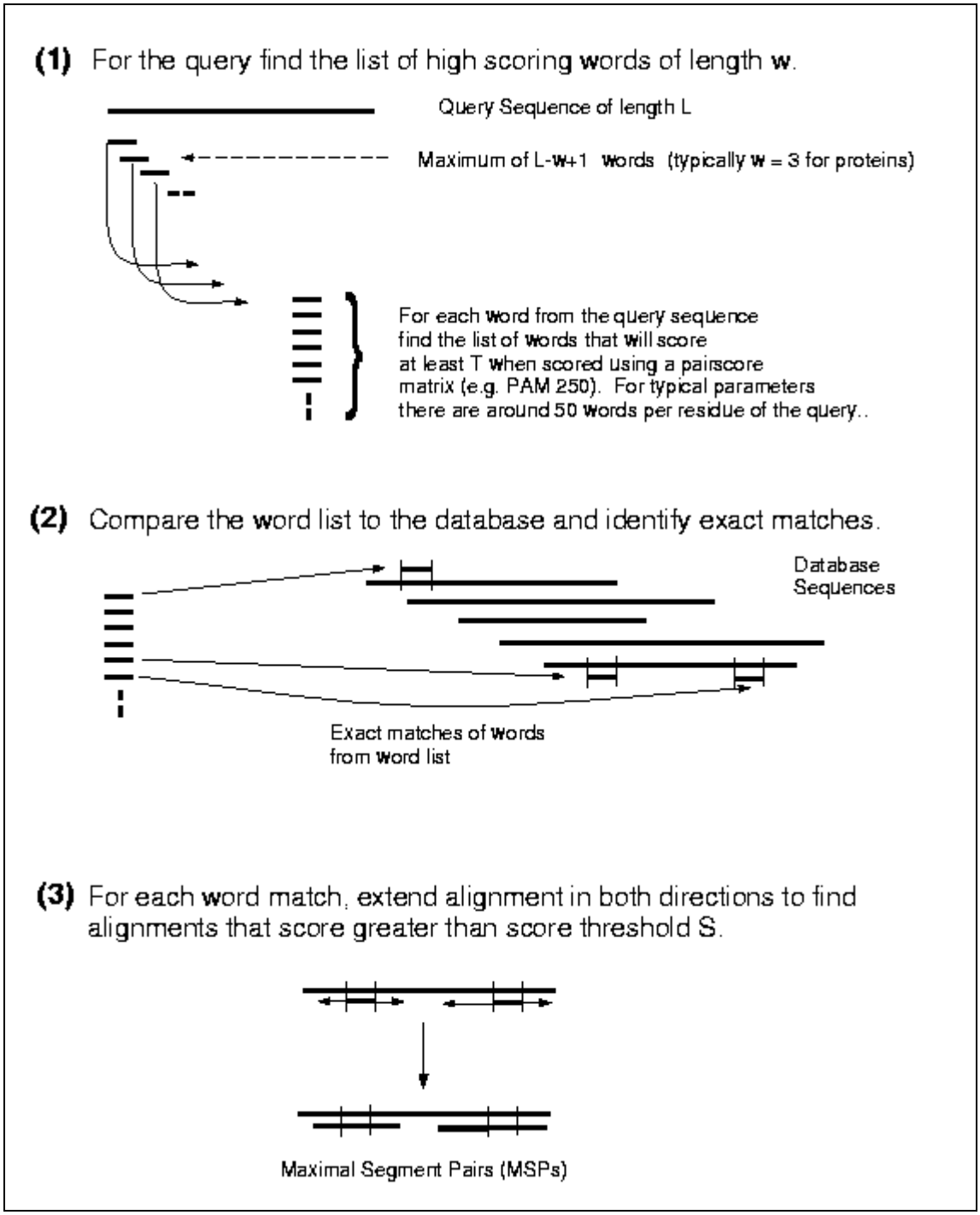


Figura 50. Esquema do algoritmo BLAST [Barton, 96].

5. Conclusão

O objetivo principal deste trabalho foi a apresentação de algoritmos de comparações de biosseqüências em bancos de dados.

No início, foi descrito o algoritmo baseado no método de programação dinâmica. Em seguida, foram apresentadas suas extensões para adequá-lo a algumas tarefas particulares de comparação de seqüências.

A complexidade de tempo destes algoritmos é quadrática, o que faz com que eles se tornem inviáveis para algumas aplicações, como a comparação de uma seqüência nova com todas as seqüências armazenadas em um banco de dados. Por este motivo, alguns métodos surgiram, como as famílias de algoritmos FAST e BLAST.

Os programas destas famílias estão baseados em heurísticas, que trouxeram uma grande melhora nos tempos de respostas das buscas em bases de biosseqüências.

Atualmente as famílias de algoritmos FAST e BLAST são as mais utilizadas pelos biólogos e, por esta razão, melhoras nestes algoritmos ou em estruturas de dados do banco de dados que facilitem ainda mais a execução destes algoritmos são muito importantes.

Agradecimentos

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro ao meu Mestrado.

Ao Dr. Antônio Basílio da FioCruz pela contribuição com seu conhecimento em Biologia.

6. Referências

[Altschul, 90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, e David J. Lipman. *"A basic local alignment search tool."* J. of Molecular Biology, 215: 403-410, 1990.

[Altschul, 97] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, e David J. Lipman. *"Gapped blast and psi-blast: a new generation of protein database search programs."* Nucleic Acids Research, 25(17):3389-3402, 1997.

[Barton, 96] Geoffrey J. Barton. *"Protein Sequence Alignment and Database Scanning."* Protein Structure Prediction - a Practical Approach. Editado por M. J. E. Sternberg, IRL Press at Oxford University Press, 1996, ISBN 0 19 963496 3. Uma versão preliminar está disponível em http://barton.ebi.ac.uk/ftp/preprints/review93.

[Dayhoff, 78] Dayhoff, M. O. Editor of *Atlas of Protein Sequence and Structure*, vol. 5, suppl. 3, Nat. Biomed. Res. Found., Washington, DC.

[Doolittle, 90] Russel F. Doolittle, editor. *"Molecular Evolution: Computer Analisys of Protein and Nucleic Acid Sequences."* Volume 183 de Methods in Enzymology. Academic Press, 1990.

[Goad, 82] W. Goad e M.Kanehisa. Nucleic Acids Research, 10, 247-263, 1982.

[Gonnet, 92] Gaston H. Gonnet. *"A Tutorial Introduction to Computational Biochemistry Using Darwin."* E.T.H., Zürich, 1992.

[Helio, 99] *"A mólecula da vida"*. http://www.aol.com.br/helio/biologia/biodna.htm.

[Karlin, 90] S. Karlin, e S.F. Altschul. Proc. Nat. Acad. Sci., U.S.A. 87, 2264-2268, 1990.

[Karlin2, 90] S.Karlin, A.Dembo, e T. Kawabata. *"Statistical Composition of High-scoring segments from Molecular Sequences."* Annals of Statistics, 18(2): 571-581, 1990.

[Lipman, 85] David J. Lipman e William R. Pearson. *"Rapid and sensitive protein similarity search."* Science, 227:1435-1441, 1985.

[Maizel, 81] J. Maizel e R. Lenk. Proc. Natl. Acad. Sci.. U.S.A. 78, 7665. 1981.

[Manber, 90] U.Manber e G.Myers. *"Suffix arrays: A new method for on-line string searches."* In Proc. First ACM-SIAM Symposium on Discrete Algorithms, 319-327, 1990.

[McCreight, 76] E.M.McCreight. *"A Space-economical Suffix Tree Construction Algorithm."* Journal of the ACM, 23:262-272, 1976.

[Meidanis, 94] João Meidanis e João Carlos Setúbal. *"Uma Introdução à Biologia Computacional"*. IX Escola de Computação. Recife, 1994.

[Needleman, 70] Saul B. Needleman e Christian D. Wunsch. *"A general method applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins."* J. of Molecular Biology, 48:443-453, 1970.

[Pearson, 88] William R. Pearson e David J. Lipman. *"Improved Tools for Biological Sequence Comparison."* Proceedings of the National Academy of Sciences of the U.S.A., 85: 2444-2448, 1988.

[Pearson, 90] William R. Pearson. *"Rapid and sensitive sequence comparison with FASTP and FASTA."* In Doolittle [Doolittle, 90], 63-98.

[Pearson, 91] William R. Pearson. *"Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA algorithms."* Genomics, 11:635-650, 1991.

[Roberts, 85] E.M.F. De Roberts, Jr.: *"Bases da Biologia Celular e Molecular"*. Editora Guanabara, 1985.

[Sankoff, 72] D. Sankoff. Proc. Natl. Acad. Sci.. U.S.A. 69, 4. 1972.

[Sellers, 74] P. Sellers. SIAM (Soc. Ind. Appl. Math.) J. Appl. Math. 26, 787. 1974.

[Sellers, 84] P. Sellers. Bull. Math. Biol. 46, 501-514, 1984.

[Smith, 81] T.F.Smith e M.S.Waterman. *"Identification of Common Molecular Subsequences."* J. of Molecular Biology, 147:195-197, 1981.

[Smith2, 81] T.F.Smith e M.S.Waterman. Advan. Appl. Math. 2, 482-489, 1981.

[Watson, 53] Watson, J. D., e Crick, F.H.C. *"Molecular Structure of Nucleic Acid: a Structure for Deoxyribose Nucleic Acid."* Nature, 171:737. 1953.

[Weiner, 73] Peter Weiner. *"Linear Pattern Matching Algorithm."* In Proc. 14th IEEE Symp. on Switching and Automata Theory, 1-11, 1973.

[Wilbur, 83] W.J. Wilbur e David J. Lipman. *"Rapid similarity searches of nucleic acid and protein data banks."* Proceedings of the National Academy of Sciences of the U.S.A., 80:726-730, 1983.