Algoritmos para Rearranjo de Seqüências de DNA

Aluno: Thiago Teixeira Santos
Orientador: Prof. Dr. Carlos Eduardo Ferreira
DCC-IME/USP

1 Novembro 1998

1 Introdução

As recentes pesquisas sobre os genomas de diversas formas de vida suscitaram diversos problemas computacionais. Devido ao grande tamanho dos genomas, mapeá-los torna-se uma tarefa difícil em que o uso de programas de computador preparados para a *comparação de seqüências* e para o *casamento de padrões* em cadeias de DNA se torna indispensável.

Este projeto tem por objetivo o estudo de *algoritmos combinatórios para rearranjo de seqüências de DNA*. Um destes problemas é o de alinhamento entre duas seqüências, para o qual desenvolvemos um programa que analisa duas cadeias, buscando alinhá-las da forma que se obtenha a maior semelhança possível entre elas.

Por exemplo, no estudo do código genético de duas espéies de animais, o biólogo verifica a semelhança entre fragmentos de suas cadeias de DNA. Esta tarefa é árdua (ou mesmo impossível) para ser feita manualmente pelo pesquisador que poderá fazer uso de tais programas.

2 Tópicos Estudados

2.1 O Problema

Considere as seguintes seqüências de DNA: GACGGAATAG e GATCGGAATAG. Não se pode deixar de notar a extrema semelhança entre elas, que fica mais evidente se as colocamos alinhadas como se segue:

CA_CGGATTAG
GATCGGAATAG

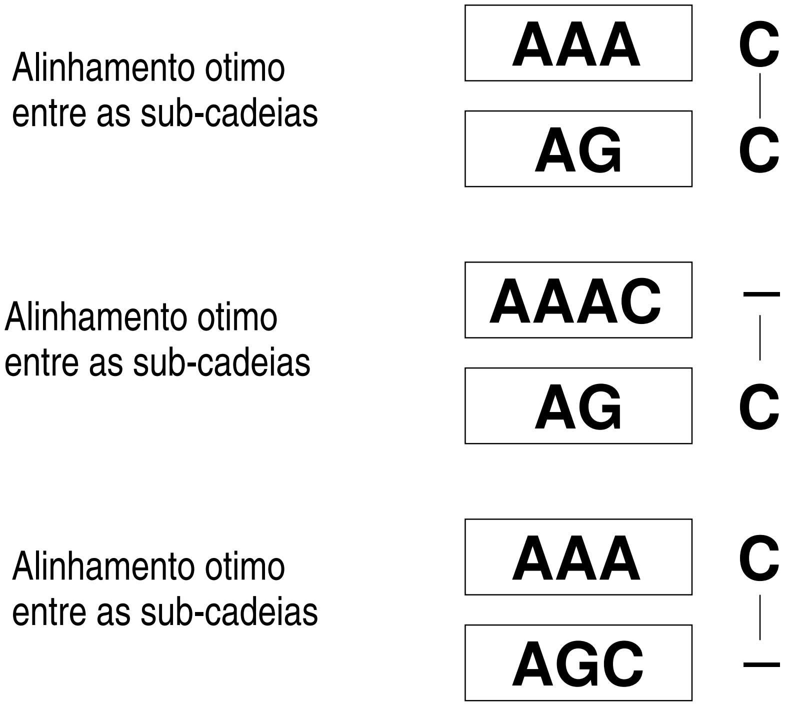
As únicas diferenças são um "T" a mais na segunda seqüência e uma troca de "A" por "T" na quarta posição da direita para a esquerda. Note que foi necessário um *buraco* na primeira cadeia para que as bases iguais antes e depois do buraco se alinhassem nas duas seqüências. Logo, as cadeias a serem alinhadas não precisam ser necessariamente do mesmo tamanho pois podemos incluir buracos em qualquer das seqüências no alinhamento.

A cada coluna do alinhamento entre duas seqüências, podemos atribuir uma *pontuação* como segue. Cada coluna do alinhamento receberá um certo número de pontos e a pontuação total do alinhamento será a soma dos pontos atribuídos às colunas. Se a coluna tiver duas bases iguais receberá 1 ponto. Se a coluna tiver duas bases diferentes recebera -1 ponto. Por fim, se a coluna possuir uma base e um buraco receberá -2 pontos. O problema será então encontrar o alinhamento que tiver pontuaçãoo máxima. Esta pontuação será chamada de *similaridade* entre as duas seqüências. Em geral, pode haver mais de um alinhamento com pontuação máxima. O alinhamento do exemplo acima ficaria:

$$9 * 1 + 1 * (-1) + 1 * (-2) = 6$$

2.2 O Algoritmo Básico

Vamos agora descrever o algoritmo que calcula o alinhamento ótimo. Tomemos como exemplo as seqüências "AAAC" e "AGC". Ao alinharmos a última coluna, teremos três possibilidades:



Este esquema nos dá um algoritmo recursivo para o cálculo do alinhamento ótimo. Chamando de $Sim(s_1, s_2)$ a similaridade entre s_1 e s_2 e $Max(i, j, k)$ o maior valor dentre i, j e k, temos:

$$Sim(AAAC, AGC) = Max(Sim(AAA, AG)+1, Sim(AAAC, AG)-2, Sim(AAA, AGC) - 2)$$

Há um grande inconveniente neste método: ele gera um número exponencial de chamadas recursivas. Teremos um total de 2^{m+n} chamadas onde m e n são os tamanhos das seqüências. Mas com o uso da *programação dinâmica* podemos impedir tal quantidade de chamadas. Esta técnica nos mostra que várias chamadas são iguais (por exemplo, $Sim(AAA, A)$ será solicitada mais de uma vez, gerando sempre o mesmo resultado!). A melhor maneira de comparar as seqüências é fazer uso da seguinte matriz:

		A	G	C
	0	-2	-4	-6
A	-2	1	-1	-3
A	-4	-1	0	-2
A	-6	-3	-2	-2
C	-8	-5	-4	-1

Convencionamos $s[a..b]$ como a subseqüência de s que vai do caracter na a-ésima posição ao caracter na b-ésina posição. Nesta matriz, o valor contido na linha i coluna j é a similaridade entre o prefixo $s_1[1..i]$ e o prefixo $s_2[1..j]$. Logo, sendo m e n os tamanhos das cadeias em estudo, a posição (m, n) da matriz contém o valor da similaridade de s_1 e s_2.

As setas indicam a origem do valor daquela posição. As setas horizontais indicam que o caracter da j-ésima posição de s_2 foi pareado com um buraco inserido em s_1 (penalização de -2). As setas verticais indicam que o caracter da i-ésima posição de s_1 foi pareado com um buraco em s_2 (penalização de -2). Já a seta diagonal indica que pareamos o caracter da i-ésima posição de s_1 com o da j-ésima posição de s_2 (1 se $s_1 = s_2$ e -1 se forem diferentes). Assim, as "setas" indicam, a partir da posição (m, n) da matriz, como devemos alinhar s_1 e s_2.

2.3 Melhorias do Algoritmo Básico

Há algumas peculiaridades negativas no algoritmo básico, onde são necessárias melhorias.

2.3.1 Buracos nas Pontas

Uma seqüência de DNA pode ser continuação da outra, como por exemplo:

ATCGTCCCACACGTAG**ACCCTGC**
e
ACCCTGCCGGTCTATTGCAAC

Mas o algoritmo básico poderia não nos mostrar este fato. Por penalizar todos os buracos com -2 pontos, este alinhamento (uma cadeia “encaixando” na extremidade da outra) não teria a melhor pontuação.

Se levarmos este fato em consideração, poderíamos modificar o algoritmo para *não* cobrar por buracos nas extremidades. No exemplo acima, o alinhamento ótimo ficaria:

ATCGTCCCACACGTAG**ACCCTGC**_____________
_____________**ACCCTGC**CGGTCTATTGCAAC

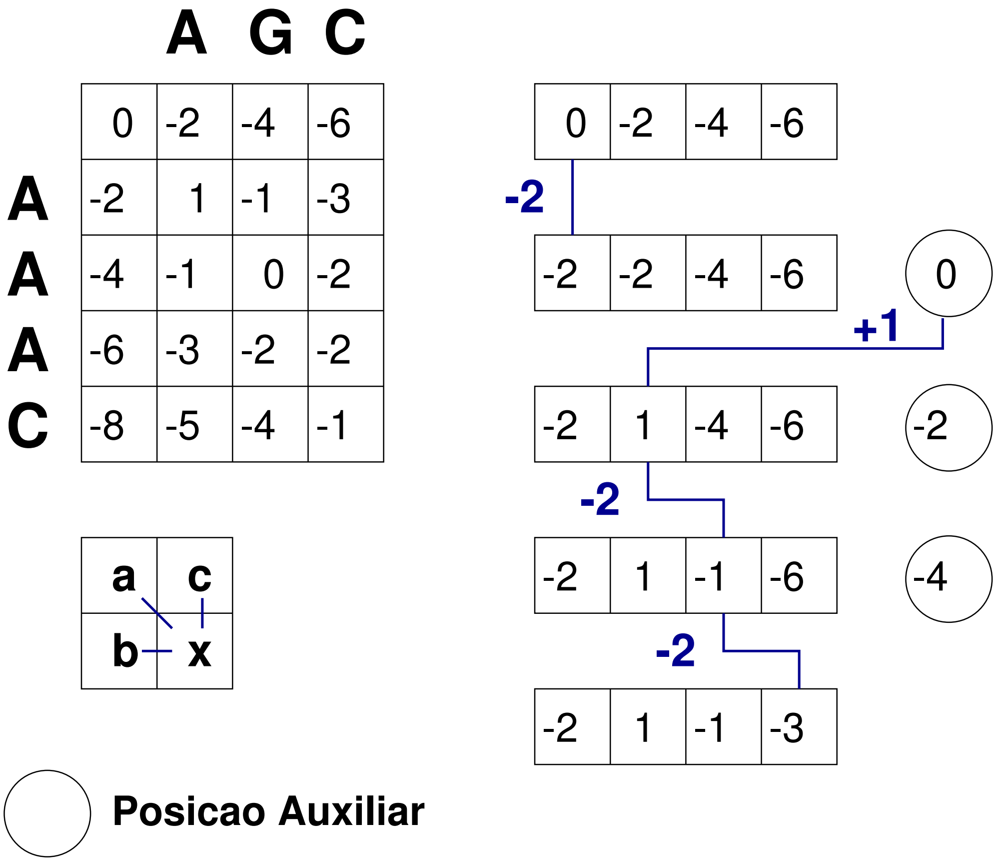
Para não cobrarmos por buracos no *início* das seqüêcias, basta inicializarmos a primeira linha e a primeira coluna da matriz com zeros. Uma vez que a posição $(i, 0)$ da matriz nos dá o valor do alinhamento ótimo entre $s_1[1..i]$ com buracos anteriores ao primeiro caracter de s_2, ao inicializarmos a *coluna* 0 com zeros não estaremos cobrando por buracos na extremidade da s_2. Raciocínio análogo para a *linha* 0 em relação a buracos na extremidade de s_1.

Para não cobrarmos por buracos no *final* das seqüêcias, basta tomarmos *o maior valor dentre a última linha e a última coluna* como a similaridade e construirmos o alinhamento ótimo a partir dele. Sendo m o comprimento de s_1, a posição (m, j) conterá o valor do alinhamento ótimo entre $s_1[1..m]$ (o que equivale a s_1 por inteira) e $s_2[1..j]$. Sendo n comprimento de s_2 e $j < n$, teremos comparado s_1 a um prefixo de s_2. Adotando (m, j) como a similaridade entre as duas cadeias, não levaremos em conta a comparação de s_1 com os caracteres finais de s_2. Podemos colocar 1 buraco ao final de s_1 para cada caracter de s_2 não comparado. Desta forma, não estaremos penalizando os buracos ao final de s_1. Raciocínio análogo para a coluna n da matriz.

2.3.2 Economia de Espaço em Memória

Seqüências de DNA são muito longas. Imaginemos a comparação de duas cadeias com 100000 bases cada. Nossa matriz teria $100000 * 100000 = 10000000000$ posições. Se cada posição ocupar 1 byte de memória teremos um gasto de 10000000000 bytes = 10 gigabytes!

Mas há uma forma de diminuir drasticamente este número obtendo *espaço linear*, proporcional a soma das seqüências de entrada. Imaginemos nossa matriz mxn substituída por uma matriz 1xn. Podemos realizar a mesma tarefa com esta matriz menor, basta repararmos que cada linha da matriz antiga é obtida imediatamente de sua linha antecessora.



Podemos notar que a pequena matriz de uma linha, que antes equivalia a primeira linha da matriz maior, depois de algumas iterações, tornou-se igual a segunda linha da grande matriz. Repetindo o mesmo processo, a matriz menor torna-se semelhante a última linha da maior de onde retira-se a similaridade. Notamos que para isso usamos a propriedade mostrada no diagrama: uma determinada posição X da matriz é obtida do máximo entre:

1. Posição A somada com a comparação entre os caracteres $s_1[i]$ e $s_2[j]$ (1 se igual e -1 para caracteres diferentes).

2. Posição B -2 (buraco em s_2).

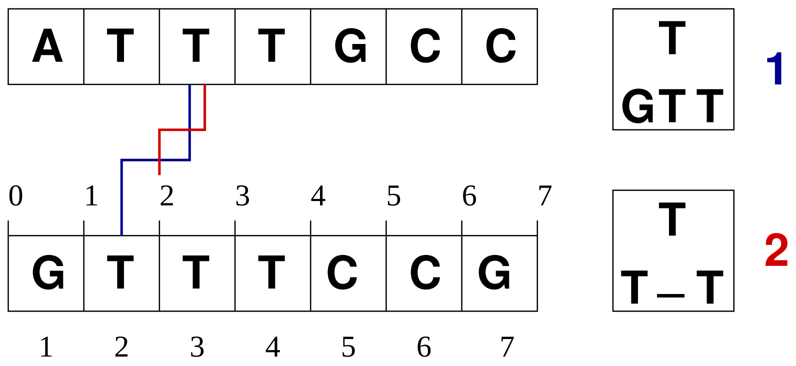
3. Posição C -2 (buraco em s_1).

A posição auxiliar guarda o caracter B, perdido na interação anterior.

2.3.3 Obtendo o Alinhamento Ótimo com Economia de Espaço

Acima, obtivemos a similaridade entre duas seqüências. Resta obtermos o alinhamento ótimo entre duas cadeias s_1 e s_2. Trata-se de uma tarefa mais complicada pois, no algoritmo antigo, tínhamos em mãos toda a matriz, algo que não ocorre agora.

Faremos uso de um algoritmo *recursivo* para resolver esta questão. Trata-se da estratégia da *divisão e conquista*, isto é, transformar um problema maior em dois subproblemas menores e, a seguir, juntar as soluções.



Note que a base T de s_1, na figura acima, pode ser alinhada de duas formas diferentes:

1. Ela pode ser alinhada com uma base j de s_2 ($1 \leq j \leq 7$, neste exemplo).

2. Ela pode ser alinhada com um buraco de s_2 ($0 \leq j \leq 7$, neste exemplo).

Para uma determinada base i de s_1, tendo decidido qual o caso em questão (1 ou 2) e determinada a posição j em s_2, o alinhamento ótimo total entre as duas seqüêcias pode ser obtido da seguinte forma:

1. $Alin.Ótimo(s_1[1..(i-1)], s_2[1..(j-1)]) + (s_1[i] <> s_2[j]) + Alin.Ótimo(s_1[(i+1)..m], s_2[(j+l)..n])$

2. $Alin.Ótimo(s_1[1..(i-1)], s_2[1..(j-1)]) + (s_1[i] <>' \_') + Alin.Ótimo(s_1[(i+1)..m], s_2[(j+l)..n])$

Só precisamos determinar o método com que a posição j (bem como as situações 1 e 2) é encontrada. Para uma dada base i de s_1, precisamos encontrar pontuações ótimas entre o prefixo $s_1[1..(i-1)]$ e os prefixos de s_2. Também precisamos encontrar as pontuações ótimas entre o sufixo $s_1[(i+1)..m]$ e os sufixos de s_2. Munidos destes valores, poderemos determinar qual posição j nos dá o melhor alinhamento entre as duas cadeias.

Para encontrarmos tais valores, basta usarmos o algoritmo que encontra a similaridade máxima entre duas seqüências (já visto anteriormente). Deste modo, a cada passo, tomamos i como sendo a metade da cadeia 1, analisamos qual seu melhor alinhamento e, em seguida, buscamos o alinhamento ótimo entre os prefixos determinado pela própria posição i e pela recém encontrada j, bem como seus sufixos. Observe o exemplo abaixo:

		G	T	T	T	C	C	G
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1	-3	-5	-7	-9	-11	-13
T	-4	-3	0	-2	-4	-6	-8	-10
T	-6	-5	-2	1	-1	-3	-5	-7

***s1:* ATTTGCC**

***s2:* GTTTCCG**

i: 4

		G	T	T	T	C	C	G
	0	-2	-4	-6	-8	-10	-12	-14
C	-2	-1	-1	-3	-5	-7	-9	-11
C	-4	-3	0	0	-2	-4	-6	-8
G	-6	-3	-2	-1	-1	-3	-5	-5

No exemplo acima, a última linha da tabela *Best Score* nos dá os alinhamentos ótimos entre o prefixo $s_1[1..3]$ com todos os prefixos de s_2 ($s_2[vazio]$, $s_2[1..1]$, $s_2[1..2]$, $\ldots s_2[1..7]$). Já a última linha da tabela *Best Score Reverso* nos dá os alinhamentos ótimos entre o sufixo $s_1[5..7]$ e todos os sufixos de s_2 ($s_2[1..7]$, $s_2[2..7]$, $\ldots s_2[vazio]$). O melhor alinhamento é dado pela posição $j = 4$:

$$prefixo[4] + sufixo[4] + Comp(s_l[i = 4] e s_2[j = 4])$$

ou seja

$$-1 + (-1) + 1 = -1$$

3 Atividades Realizadas

- Implementação dos algoritmos estudados na forma do aplicativo Análise de DNA, que pode ser encontrado na URL **http://www.ime.usp.br/ thsant**.
- Reuniões semanais com o orientador (Prof. Ferreira) para discussão do projeto, resolução de dúvidas e definição de metas.
- Estudo do algoritmo que visa à utilização linear de memória.
- Estudo do problema de pareamento sem penalização para a inserção de buracos nas extremidades das cadeias, combinado ao uso linear de memória.

4 Comentários e Conclusões

Nosso projeto é uma continuação do trabalho do aluno Dan Faccio, sob supervisão do Prof. Ferreira, realizado no ano de 1997.

Como objetivo maior temos a Montagem de Seqüências de DNA, uma continuação deste projeto que será ralizada no período 98/99.

Referências

[1] J. Meidanis, J. Setúbal. *Uma Introdução à Biologia Computacional*, IX Escola de Computação, Recife, 1994.

[2] C. E. Ferreira, C.C. Souza, Y. Wakabayashi. *Rearrangement of DNA Fragments: a branch-and-cut algorithm*. Relatório Técnico, Departamento de Ciência da Computação, IME-USP - Janeiro, 1997.