Biologia Computacional - 2004/2 09/11/04

Aula 1: Casamento Inexato, Alinhamento de Sequências e Programação Dinâmica

Instrutor: Berilhes Borges Garcia Escriba: André C. M. Costa

DRAFT

1 Pesquisando Banco de Dados de Sequências

Descobertas biológicas baseadas na similaridade de sequências tornaram-se uma rotina.
A primeira de tais descobertas foi a conexão entre oncogenes e proteínas regulando o crescimento. *Simian Sarcoma* é um retrovírus que causa câncer em macacos. Este oncogene chamada *v-sis* foi sequenciado em 1983 (Doolittle et. al.). Comparando a sequência de aminoácidos codificada pelo *v-sis* contra sequências de proteínas anteriormente publicadas, Doolittle observou uma similaridade significativa com um fator de crescimento chamado PDGF.

2 Casamento Inexato, Alinhamento de Sequências e Programação Dinâmica

Casamento inexato ou aproximado e comparações de sequências são ferramentas centrais em biologia molecular computacional. Porque:

- Erros em dados moleculares
- Permitem tentar entender a evolução mutacional das sequências.

Igualdade inexata é uma noção muito rígida de similaridade.
Comparação e casamento inexato utilizam alinhamento de strings (cadeias) para reconhecer suas similaridades.
Computar alinhamentos envolve:

- Considerar subsequências (ao invés de substrings contíguas), e
- Resolver problemas de otimização (maximizar a similaridade de sequências), frequentemente isto é feito por meio de programação dinâmica.

3 O Problema da Distância de Edição

O problema mais clássico, em casamento inexato, resolvido por programação dinâmica é o problema da distância de edição.
Uma forma rotineira de formalizar a diferença de distância entre duas cadeias S_1 e S_2 é determinar o número de operações de edição de caracteres unícios necessários para transformar S_1 em S_2.

Uma transcrição de edição é uma sequência de operações de I (inserir o próximo caracter de S_2), D (deletar), R (substituir o próximo caracter de S_2), e M (casar) que transforma S_1 em S_2.
Example: Uma transcrção para transformar "vintner" para "writers"

```
R I M D M D M M I
v   i n t n e r
w r i   t   e r s
```

⊠

A distância de edição entre as cadeias S_1 e S_2 é o número mínimo de operações I, D e R em qualquer transcrição que transforma S_1 em S_2. É também chamada de distância de Levenshtein.
Um transcrição com o menor número de operações I, D, e R é uma transcrição ótima.

3.1 Problema da Distância de Edição

Compute a distância de edição e uma transcrição ótima para duas cadeias S_1 e S_2.
Note que a função de S_1 e S_2 são simétricas, uma vez que uma deleção em uma string corresponde a uma inserção em outra.

4 Alinhamento de Strings (Cadeias)

Um alinhamento é uma forma alternativa (à transcrição de edição) de indexar as diferenças e similaridades entre strings.
Um alinhamento (global) de S_1 e S_2 é obtido inserindo-se espaços nas cadeias, e então colocando-se um sob o outro de modo que cada caracter ou espaço esteja emparelhado a um caracter único ou espaço da outra cadeia.

- "global" → cadeias inteiras participam no alinhamento.
- "alinhamento local" → regiões com alta similaridade.

Example: Um alinhamento global de "vintner" e "writers".

```
v _ i n t n e r _
w r i _ t _ e r s
```

⊠

5 Computando a distância de Edição

A distância de edição de duas strings $S_1[1 \dots n]$ e $S_2[1 \dots m]$ pode ser computado aplicando-se programação dinâmica.

Definição 1. *$D(i, j)$ é uma distância de edição dos prefixos $S_1[1 \dots i]$ e $S_2[1 \dots j]$.*

Note que $D(n, m)$ é a distância de edição de S_1 e S_2.

Programação dinâmica computa $D(n, m)$ computando $D(i, j)$ para todo $i \leq n$ e $j \leq m$

6 Componentes da Programação Dinâmica

Programação Dinâmica (para o problema da distância de edição), possui três componentes essenciais:

- Relação de Recorrência → Como $D(i, j)$ é determinado dos valores $D(i', j')$ com i' e j' menores?
- Computação Tabular → Como memorizar valores computados, de modo a evitar computação repetidas?
- Traceback → Como encontrar (como reconstruir) a solução ótima?

6.1 A Relação de Recorrência

Como determinar os valores $D(i, j)$?

Base: $D(i, 0)$: como editar $S_1[1 \dots i]$ para $S_2[1 \dots 0] = e$?
Com i deleções → $D(i, 0) = i$.
Similarmente, $D(0, j) = j \leftarrow$ insira $S_2[1], S_2[2], \dots, S_2[j]$.

Caso indutivo para $i, j > 0$:

$$D(i,j) = min \begin{cases} D(i-1,j)+1 & \text{(D)} \\ D(i,j-1)+1 & \text{(I)} \\ D(i-1,j-1)+t(i,j) & \text{(M R)} \end{cases}$$

onde $t(i, j) = 0$ se $S_1[i] = S_2[j]$ e 1 caso contrário.

O que isto significa?
Uma transcrição que transforma $S_1[1 \dots i]$ para $S_2[1 \dots j]$ faz uma de três coisas:

- transforma $S_1[1 \dots i-1]$ para $S_2[1 \dots j]$ e deleta $S_1[j]$ (D)
- transforma $S_1[1 \dots i]$ para $S_2[1 \dots j-1]$ e insere $S_2[j]$, ou (I)
- transforma $S_1[1 \dots i]$ para $S_2[1 \dots j-1]$ e então casa ou substitui $S_1[i]$ por $S_2[j]$. (M R)

Uma transcrição ótima é aquela que possui o menor valor de acordo com estas três possibilidades.

Note que podem existir mais de uma transcrição ótima, uma vez que mais de um dos três casos pode ter o mesmo valor.

6.2 Computação Tabular da Distância de Edição

É fácil implementar a recorrência para $D(i, j)$ como um procedimento recursivo.

O problema é que um procedimento recursivo para $D(i, j)$ computa os mesmos valores um número exponencial de vezes. Contudo, existem somente $(n+1) \times (m+1)$ valores diferentes de $D(i, j)(0 \leq i \leq n, 0 \leq j \leq m)$.

Solução 1. *Computá-los em uam ordem conveniente (bottom-up), e armazená-los em um vetor de modo que cada valor é computado uma única vez.*

6.2.1 Computação Bottom-Up

Preencha uma tabela $D(i, j)$, onde $i = 0, \ldots, n$ e $j = 0, \ldots, m$, em uma ordem crescente de pares (i, j).
Primeiro inicialize a coluna 0 e linha 0 de acordo com os casos bases da recorrência.

```
for i := 0 to n do D(i,0) = i;
for j := 0 to m do D(0,j) = j;
```

Tabela $D(i, j)$ depois da inicialização da linha 0 e coluna 0.

$D(i,j)$	S_2		**w**	**r**	**i**	**t**	**e**	**r**	**s**
S_1		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1							
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

Então compute as células restantes.

6.2.2 Computando as Células Restantes

Células internas $D(i, j)(i, j > 0)$ podem ser computadas em qualquer ordem, desde que os três valores requeridos pela recorrência já tenham sido computados.

$D(i-1, j-1) \searrow$	$\downarrow D(i-1, j)$
$D(i, j-1) \rightarrow$	$D(i, j)$

Por exemplo, considerando primeiro a linha

```
for i := 1 to n do
   for j := 1 to m do
      compute D(i, j) de acordo com a recorrência
```

Exemplo de uma computação primeiro em linha

$D(i,j)$	S_2		**w**	**r**	**i**	**t**	**e**	**r**	**s**
S_1		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4				
n	5	5							
e	6	6							
r	7	7							

6.2.3 Complexidade da Tabulação

Cada uma das $\Theta(nm)$ células é preenchida em tempo constante.

Teorema 2. *A distância de edição $D(n,m)$ das strings $S_1[1\ldots n]$ e $S_2[1\ldots m]$ pode ser computada em tempo $O(nm)$.*

Uma transcrição de edição ótima pode ser computada dentro do mesmo limite de tempo.
Para isto, armazene em cada célula (i,j) têS ponteiros, e ajuste-os para apontar para a(s) célula(s) que fornece(m) o valor para $D(i,j)$.

- ponteiros não são necsaaários, mas são úteis para explicação.
- ponteiros na linha 0 apontam para células na esquerda, e ponteiros na coluna 0 para a célula acima.

$D(i,j)$	S_2		**w**	**r**	**i**	**t**	**e**	**r**	**s**
S_1		0	1	2	3	4	5	6	7
	0	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7
v	1	↑ 1	↖ 1	↖← 2	↖← 3	↖← 4	↖← 5	↖← 6	↖← 7
i	2	↑ 2	↖↑ 2	↖ 2	↖ 2	← 3	← 4	← 5	← 6
n	3	↑ 3	↖↑ 3	↖↑ 3	↖↑ 3	↖ 3	↖← 4	↖← 5	↖← 6
t	4	↑ 4	↖↑ 4	↖↑ 4	↖↑ 4	↖ 3	↖← 4	↖← 5	↖← 6
n	5	↑ 5	↖↑ 5	↖↑ 5	↖↑ 5	↑ 4	↖ 4	↖← 5	↖← 6
e	6	↑ 6	↖↑ 6	↖↑ 6	↖↑ 6	↑ 5	↖ 4	↖← 5	↖← 6
r	7	↑ 7	↖↑ 7	↖ 6	↖←↑ 7	↑ 6	↑ 5	↖ 4	← 5

7 Encontrando Transcrição e Alinhamentos Ótimos

Uma transcrição ótima pode ser encontrada seguindo os ponteiros da célula (n,m) para a célula $(0,0)$.

- ponteiro '←' de (i,j) para $(i,j-1)-$ inserção de $S_2[j]$

- ponteiro '↑' de (i, j) para $(i-1, j)-$ delerção de $S_1[i]$
- ponteiro '$\nwarrow$' de (i, j) para $(i-1, j-1)-$ casamento/substituição dependendo se $S_1[i]$ e $S_2[j]$ casam ou não.

Alternativamente, o caminho especifica um alinhamento ótimo, onde

1. ponteiro '←' − espaço em S_1 alinhado à $S_2[j]$,
2. ponteiro '↑' − espaço em S_2 alinhado à $S_1[j]$, e
3. ponteiro '$\nwarrow$' − espaço em $S_1[i]$ e $S_2[j]$ estão alinhados.

7.1 Encontrando Alinhamentos Ótimos

Example: Alinhamentos ótimos especificados pelos ponteiros da tabela precedente.

1. S_1' : _ v i n t n e r _
   S_2' : w r i _ t _ e r s
2. S_1' : v _ i n t n e r _
   S_2' : w r i _ t _ e r s
3. S_1' : v i n t n e r _
   S_2' : w r i t _ e r s

⊠

7.2 Complexidade de Trilhar uma Transcrição Ótima

Teorema 3. *Depois de computar a tabela de programação dinâmica com ponteiros, uma transcrição ótima pode ser encontrada em tempo $O(n+m)$.*

Proof. Começando na célula (n, m), um caminho para $(0, 0)$ pode ser encontrado escolhendo-se qualquer ponteiro que sai desta célula. Adicionalmente cada célula $(i, j) \neq (0, 0)$ possui pelo menos um ponteiro que sai dela para uma das seguintes células: $(i-1, j), (i, j-1), (i-1, j-1)$. No máximo $n+m$ ponteiros necessitam ser seguidos. □

Os requisitos de espaço e tempo ($\Theta(nm)$) da solução de programação dinâmica pode ser melhorada com a ajuda de técnicas avançadas.