Encontro França-Brasil de Bioinformática
Universidade Estadual de Santa Cruz (UESC)
Ilhéus-BA - Brasil

Alinhamento de Sequências e Genômica Comparativa

Maria Emília M. T. Walter
Departamento de Ciência da Computação
Marcelo M. Brígido
Departamento de Biologia Celular
Universidade de Brasília

Ilhéus, 09 de novembro de 2010

Conteúdo e Metodologia

- Aulas Teóricas e Práticas
  - Teoria (11:30h-12:00h): Introdução a métodos de comparação e alinhamentos de sequências biológicas
  - Teoria (16:00h-16:40h): Métodos computacionais para identificar genes parálogos e genes ortólogos: Inparanoid (16:00h-16:15h), OrthoMCL (16:15h-16:30h), 3GC e n3GC (16:30h-16:40h)
  - Prática (16:40h-18:00h): métodos computacionais: Inparanoid (30 minutos:16:40h -17:10h), OrthoMCL (40 minutos: 17:10h – 17:50h); 3GC e n3GC (10 minutos: 17:50h – 18:00h).

Comparação de sequências

(Setubal/Meidanis: Introduction to Computational Molecular Biology)

- Operação básica mais importante em Bioinformática
- Consiste em encontrar:
  - quais partes das sequências são “parecidas” e quais partes são “diferentes”
- Similaridade de duas sequências:
  - Medida de quanto estas sequências são “parecidas”
- Alinhamento de duas sequências:
  - forma de colocar uma sequência acima da outra para explicitar a correspondência entre dois caracteres destas sequências

Exemplos

- Temos uma sequência a ser comparada com milhares de outras sequências
  - Pergunta: existem sequências contendo subcadeias *(substrings)* similares à primeira sequência?
- Temos duas sequências sobre o mesmo alfabeto
  - Pergunta: existem duas *substrings*, uma de cada sequência, que são similares?

Comparação de sequências

- Problema de comparar duas sequências:
  - encontrar o melhor alinhamento entre duas sequências
  - três casos:
    - inteiras: comparação global
      (algoritmo de Needleman-Wunsch)
    - *substrings* das duas sequências: comparação local
      (algoritmo de Smith-Waterman)
    - alinhar prefixos e sufixos: comparação semi-global

Comparação global

- Exemplo: tomar duas sequências

  GACGGATTAG

  GATCGGAATAG

- alinhamento:

  G A – C G G A T T A G

  G A T C G G A A T A G

algoritmo: tomar duas sequências e determinar o melhor alinhamento entre elas

Alinhamento entre duas sequências

- Definições:
  - alinhamentos podem conter espaços
  - **alinhamento**: inserção de espaços nas sequências para que ambas fiquem de mesmo tamanho
  - explicitar o alinhamento: colocamos uma sequência com espaços, acima da outra, fazendo corresponder caracteres/espaços da primeira com caracteres/ espaços da segunda sequência
  - não podemos alinhar espaço/espaço
  - podem aparecer espaços nas extremidades das sequências

Escore

- Dado um alinhamento entre duas sequências, associamos um escore a ele:
  - cada coluna recebe um valor:
    - +1: casamento *(match)*, caracteres idênticos
    - -1: não casamento (*mismatch)*, caracteres diferentes
    - -2: espaço *(gap)*, representado por -
  - escore total: soma dos valores associados à cada coluna
  - melhor alinhamento: máximo escore
  - **similaridade** das sequências s e t: sim(s,t)

Exemplo

- alinhamento anterior:

  G A – C G G A T T A G

  G A T C G G A A T A G

  +1 +1 -2 +1 +1 +1 +1 -1 +1 +1 +1

  escore: $9_x(+1) + 1_x(-2) + 1_x(-1) = 6$

- escolha de +1, -1, -2: comumente usado, mas pode ser modificado

Algoritmo

- gerar todos os possíveis alinhamentos, e escolher o melhor: tempo exponencial
- técnica: programação dinâmica
  - dadas duas sequências s e t, construímos soluções determinando todas as similaridades entre prefixos de s e t
  - início: computamos prefixos menores
  - depois: usamos valores previamente calculados para resolver o problema para prefixos maiores

Algoritmo

- |s| = m, m+1 prefixos (incluindo ε: *substring* vazia)
- |t| = n, n+1 prefixos (incluindo ε)
- criamos uma matriz (m+1)x(n+1) tal que
  - (i,j) = sim(s[1..i],t[1..j])
  - inicialização: múltiplos das penalidades de *gaps* (-2), razão: único alinhamento possível com uma das sequências vazia é adicionar -
  - exemplo: s = AAAC alinhamento: A A A C

    t = ε - - - -

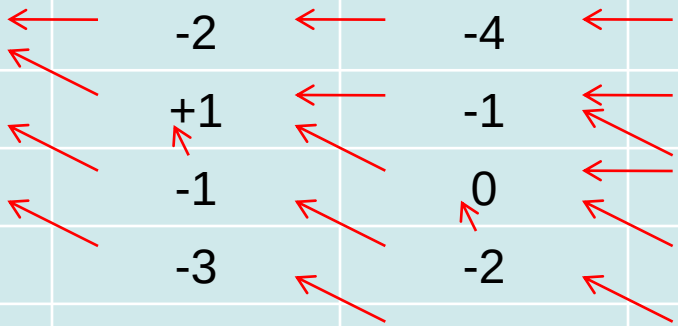
    -2-2-2-2 = -8

escore do alinhamento: -2k,

k é o tamanho da sequência não-vazia

Matriz

	ε 0	A 1	G 2	C 3	sequência t
ε	0	-2	-4	-6	
A 0	-2	+1	-1	-3	
A 1	-4	-1	0	-2	
A 2	-6	-3	-2	-1	
C 3	-8	-5	-4	-1	
sequência s					



s = A A A C

t = A G C

Construção da matriz

- para construir (i,j), basta examinar três entradas:
  - (i-1,j): posição anterior na vertical
  - (i-1,j-1): posição anterior na diagonal
  - (i,j-1): posição anterior na horizontal
  - porque para obter o alinhamento entre s[1..i] e t[1..j]:
    - alinhar s[1..i] com t[1..j-1] e alinhar -/t[j] ou
    - alinhar s[1..i-1] com t[1..j-1] e alinhar s[i]/t[j] ou
    - alinhar s[1..i-1] com t[1..j] e alinhar s[i]/-
    - observação: não se pode alinhar -/-
  - $sim(s[1..i], t[1..j]) = \max(sim(s[1..i], t[1..j-1]) - 2,$
    $sim(s[1..i-1], t[1..j-1]) + p(i,j),$
    $sim(s[1..i-1], t[1..j]) - 2)$

$$p(i,j) = \begin{cases} +1 & s[i]=t[j] \\ -1 & s[i]\neq t[j] \end{cases}$$

Construção da matriz

- $sim(s[1..i], t[1..j]) = \max(sim(s[1..i], t[1..j-1]) - 2,$
  $sim(s[1..i-1], t[1..j-1]) + p(i,j), p(i,j) = \begin{cases} +1 & s[i]=t[j] \\ -1 & s[i]\neq t[j] \end{cases}$
  $sim(s[1..i-1], t[1..j]) - 2)$

Denotando a matriz por *a*:

$$a[i,j] = \max \begin{cases} a[i,j-1]-2 \\ a[i-1,j-1]+p(i,j) \\ a[i-1,j]-2 \end{cases}$$

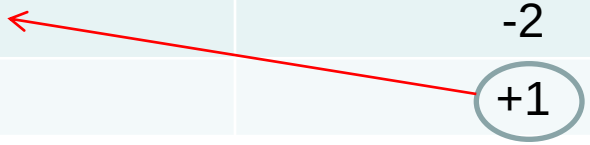
Calculamos por linha, da esquerda para a direita em cada linha

ou por coluna, de cima para baixo em cada coluna,

assegurando que a[i,j-1], a[i-1,j-1] e a[i-1,j] tenham sido calculadas quando a[i,j] tiver de ser calculada

Exemplo

	ε 0	A 1
ε	0	-2
A 0	-2	+1



a[1,1] = max (a[1,0]-2, a[0,0]+p(1,1), a[0,1-2]-2)
= max (-2 -2, 0 +1, -2 -2)
= max (-4, +1, -4)
= +1

Obter alinhamentos ótimos

- usar setas:
  - início: (m,n)
  - seguir as setas
  - final: (0,0)
  - cada seta indica uma coluna do alinhamento:
    - seta horizontal à esquerda ← -/t[j]
    - seta vertical para cima ↑ s[i]/-
    - seta diagonal ↖ s[i]/t[j]

Exemplos

- Escolhendo a seta no sentido anti-horário:
  - A A A C
    A G - C
    +1 -1 -2 +1 = -1 = a[m,n]
  - A A A C
    - A G C
    -2 +1 -1 +1 = -1 = a[m,n]
- não é necessário implementar a seta, basta testar se
  a[i,j] = max (a[i-1,j]-2, a[i-1,j-1]+p(i,j), a[i,j-1]-2)
  no sentido anti-horário
- complexidade do algoritmo:
  - construir a matriz: tempo: O(mn), quando |s|=|t|=n, então $O(n^2)$
    espaço: O(mn), quando |s|=|t|=n, então $O(n^2)$
  - construir o alinhamento: tempo e espaço: O(m+n), quando |s|=|t|=n, então O(n)

Comparação local

- alinhamento local entre s e t:
  - alinhamento entre *substring* de s e *substring* de t
- algoritmo: variante do algoritmo básico
- estrutura de dados: matriz $(m+1)x(n+1)$
- (i,j) = maior escore entre sufixo(s[1..i]) e sufixo(t[1..j])
- linha 1 e coluna 1: inicializadas com 0
  - sempre existe alinhamento entre sufixos vazios de s[1..i] e t[1..j] com escore 0, então (i,j) ≥ 0
- a[i,j] = max (a[i,j-1]-2,
  a[i-1,j-1]+p(i,j),
  a[i-1,j]-2,
  0)
- final: encontrar a maior entrada da matriz: escore de um alinhamento local ótimo, início de um alinhamento local ótimo

Matriz

	ε 0	A 1	G 2	C 3	sequência t
ε	0	0	0	0	
A 0	0	+1	0	0	
A 1	0	+1	0	0	
A 2	0	+1	0	0	
C 3	0	0	0	+1	
sequência s					

s = A A A C
t = A G C

alinhamentos:

C/C: s[4]/t[3] A/A: s[3]/t[1], s[2]/t[1], s[1]/t[1],