For written notes on this lecture, please read chapter 10 of *The Practical Bioinformatician*

CS2220: Introduction to Computational Biology

Lecture 6: Essence of Sequence Comparison

Limsoon Wong

6 March 2009

Plan

- **Dynamic Programming**
- **String Comparison**

- **Sequence Alignment**
  - Pairwise Alignment
    - **Needleman-Wunsch global alignment algorithm**
    - **Smith-Waterman local alignment algorithm**
  - Multiple Alignment

- **Popular tools**
  - FASTA, BLAST, Pattern Hunter

What is Dynamic Programming

The Knapsack Problem

- **Each item that can go into the knapsack has a size and a benefit**
- **The knapsack has a certain capacity**
- **What should go into the knapsack so as to maximize the total benefit?**

Formulation of a Solution

Source: http://mat.gsia.cmu.edu/classes/dynamic/node6.html

- **Intuitively, to fill a *w* pound knapsack, we must end off by adding some item. If we add item *j*, we end up with a knapsack *k'* of size $w - w_j$ to fill …**

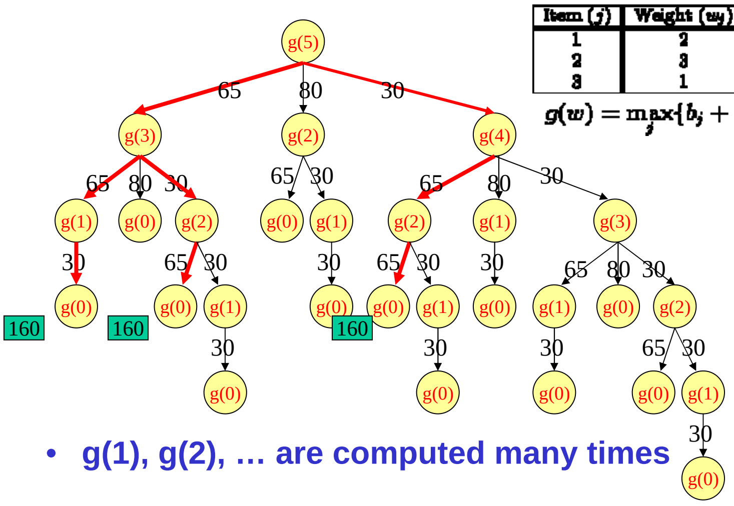
Why is g(w) optimal?

$$g(w) = \max_j \{b_j + g(w - w_j)\}$$

- **Where**
  - w_j and b_j be weight and benefit for item *j*
  - $g(w)$ be max benefit that can be gained from a *w*-pound knapsack

---

An Example: Direct Recursive Evaluation

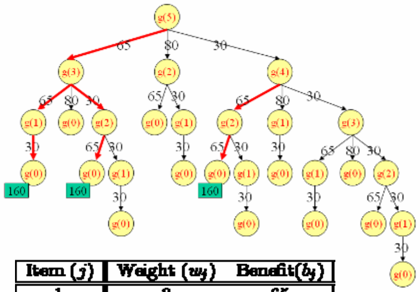


Item (*j*)	Weight (w_j)	Benefit(b_j)
1	2	65
2	3	80
3	1	30

$$g(w) = \max_j \{b_j + g(w - w_j)\}$$

- **g(1), g(2), … are computed many times**

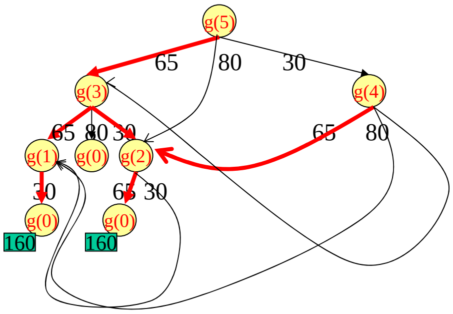
“Memoize” to avoid recomputation



Item (j)	Weight (w_j)	Benefit(b_j)
1	2	65
2	3	80
3	1	30

$$g(w) = \max_j \{b_j + g(w - w_j)\}$$

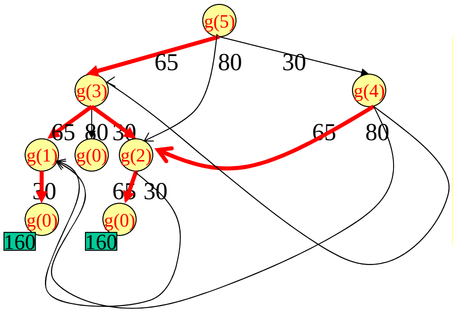
```
int s[]; s[0] := 0;
g’(w) = if s[w] is defined
    then return s[w];
    else {
        s[w] := max$_j${b$_j$ + g’(w – w$_j$)};
        return s[w]; }
```



Remove Recursion: Dynamic Programming

```
int s[]; s[0] := 0;
g’(w) = if s[w] is defined
    then return s[w];
    else {
        s[w] := max$_j${b$_j$ + g’(w – w$_j$)};
        return s[w]; }
```

```
int s[]; s[0] := 0; s[1] := 30;
s[2] := 65; s[3] = 95;
for i := 4 .. w do
    s[i] := max$_j${b$_j$ + s[i – w$_j$]};
return s[w];
```



g(0) = 0

g(1) = 30, item 3

g(2) = max{65 + g(0) =65, 30 + g(1) = 60} = 65, item 1

g(3) = max{65 + g(1) = 95, 80 + g(0) = 80, 30 + g(2) = 95} = 95, item 1/3

g(4) = max{65 + g(2) = 130, 80 + g(1) = 110, 30 + g(3) = 125} = 130, item 1

g(5) = max{65 + g(3) = 160, 80 + g(2) = 145, 30 + g(4) = 160} = 160, item 1/3

Characteristics of Dynamic Programming

Source: http://mat.gsia.cmu.edu/classes/dynamic/node4.html

- Problem can be divided into *stages* with a *decision* required at each stage

Exercise: What is a stage in the Knapsack problem?

- Each stage has a # of *states* associated
- Decision at one stage transforms one state into a state in the next stage

Exercise: What is a state in the Knapsack problem?

- Given current state, the optimal decision for each remaining states does not depend on next states or decisions

E.g., g(2) doesnt depends on g(3)

- There is a recursive relationship that identifies the optimal decision for stage *j+1*, given stage *j* has already been solved
- The initial stages must be solvable by themselves

E.g., g(0) = 0

Sequence Alignment

Motivations for Sequence Comparison

- DNA is blue print for living organisms

⇒ Evolution is related to changes in DNA

⇒ By comparing DNA seqs we can infer evolutionary relationships betw seqs w/o knowledge of the evolutionary events themselves

- Foundation for inferring function, active site, and key mutations

---

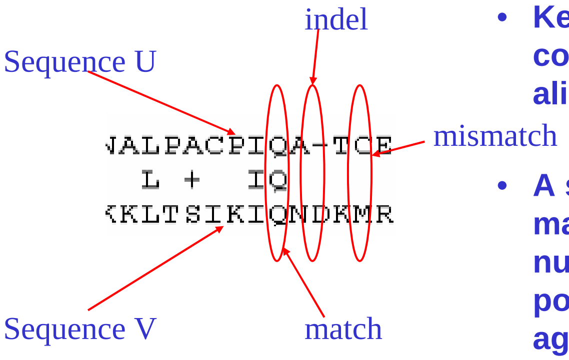
Earliest Research in Seq Comparison

Source: Ken Sung

- Doolittle et al. (*Science*, July 1983) searched for platelet-derived growth factor (PDGF) in his own DB. He found that PDGF is similar to v-sis oncogene

```
PDGF-2  1       SLGSLTIAEPAMIAECKTREEVFCICRRL?DR?? 34
p28sis 61 LARGKRSLGSLSVAEPAMIAECKTRTEVFEISRRLIDRTN 100
```

Sequence Alignment



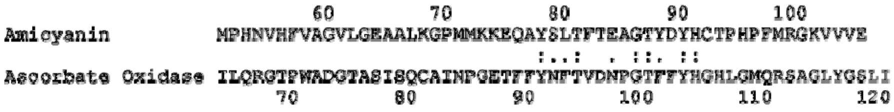
- **Key aspect of seq comparison is seq alignment**
- **A seq alignment maximizes the number of positions that are in agreement in two sequences**

Sequence Alignment: Poor Example

- **Poor seq alignment shows few matched positions**

⇒ The two proteins are not likely to be homologous

Alignment by FASTA of the sequences of amicyanin and domain 1 of ascorbate oxidase



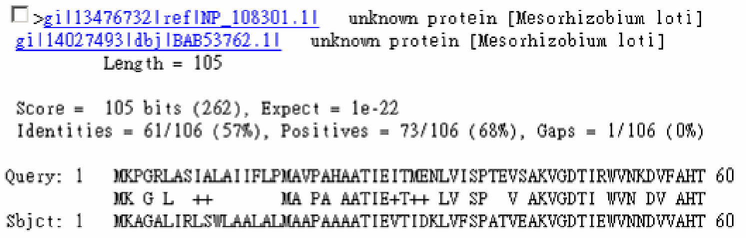
```
                          60        70        80        90        100
Amicyanin          MPHNVHFVAGVLGEAALKGPMMKKEQAYSLTFTEAGTYDYHCTPHPFMRGKVVVE
                                                :..:   . ::. ::
Ascorbate Oxidase  ILQRGTPWADGTASISQCAINPGETFFYNFTVDNPGTFFYHGHLGMQRSAGLYGSLI
                       70        80        90        100       110       120
```

No obvious match between
Amicyanin and Ascorbate Oxidase

Sequence Alignment: Good Example

- **Good alignment usually has clusters of extensive matched positions**

⇒ The two proteins are likely to be homologous



```
☐ >gi|13476732|ref|NP_108301.1|   unknown protein [Mesorhizobium loti]
gi|14027493|dbj|BAB53762.1|   unknown protein [Mesorhizobium loti]
          Length = 105

 Score =  105 bits (262), Expect = 1e-22
 Identities = 61/106 (57%), Positives = 73/106 (68%), Gaps = 1/106 (0%)

Query: 1   MKPGRLASIALAIIFLPMAVPAHAATIEITMENLVISPTEVSAKVGDTIRWVNKDVFAHT 60
            MK G L  ++       MA PA AATIE+T++ LV SP  V AKVGDTI WVN DV AHT
Sbjct: 1   MKAGALIRLSWLAALALMAAPAAAATIEVTIDKLVFSPATVEAKVGDTIEWVNNDVVAHT 60
```

good match between
Amicyanin and unknown M. loti protein

Alignment:

Simple-Minded Probability & Score

Let p, q, r be respectively the probability of a match, a mismatch, and an indel. Then the probability of an alignment $A = (X, Y)$ is

$$prob(A) = p^m \cdot q^n \cdot r^h$$

where

$$\begin{aligned} m &= |\{i \mid x'_i = y'_i \neq -\}| \\ n &= |\{i \mid x'_i \neq y'_i, x'_i \neq -, y'_i \neq -\}| \\ h &= |\{i \mid x'_i = -, y'_i \neq -\} \cup \{i \mid x'_i \neq -, y'_i = -\}| \end{aligned}$$

- **Define score S(A) by simple log likelihood as**
  - S(A) = log(prob(A)) - [m log(s) + h log(s)], with log(p/s) = 1
- **Then S(A) = #matches - μ #mismatches - δ #indels**

Exercise: Derive μ and δ

Global Pairwise Alignment:

Problem Definition

- **Given sequences *U* and *V* of lengths *n* and *m*, then number of possible alignments is given by**
  - $f(n, m) = f(n-1,m) + f(n-1,m-1) + f(n,m-1)$
  - $f(n,n) \sim (1 + \sqrt{2})^{2n+1} n^{-1/2}$

Exercise: Explain the recurrence above

- **The problem of finding a global pairwise alignment is to find an alignment *A* so that *S(A)* is max among exponential number of possible alternatives**

Global Pairwise Alignment:

Dynamic Programming Solution

- **Define an indel-similarity matrix *s(.,.)*; e.g.,**
  - $s(x,x) = 2$
  - $s(x,y) = -\mu$, if $x \neq y$
- **Then**

Let U and V be two sequences of length n and m. Then their global pairwise alignment can be extracted from the dynamic programming computation of $S_{n,m}$, where

$$S_{i,j} = \max \left\{ \begin{array}{l} S_{i-1,j-1} + s(u'_i, v'_j) \\ S_{i-1,j} - \delta \\ S_{i,j-1} - \delta \end{array} \right\}$$

Exercise: What is the effect of a large δ ?

This is the basic idea of the Needleman-Wunsch algorithm

Needleman-Wunsch Algorithm (I)

Source: Ken Sung

- **Consider two strings S[1..n] and T[1..m]**
- **Let V(i, j) be score of opt alignment betw S[1..i] and T[1..j]**

- **Basis:**
  - V(0, 0) = 0
  - V(0, j) = V(0, j –1) – δ
    - **Insert j times**
  - V(i, 0) = V(i – 1, 0) – δ
    - **Delete i times**

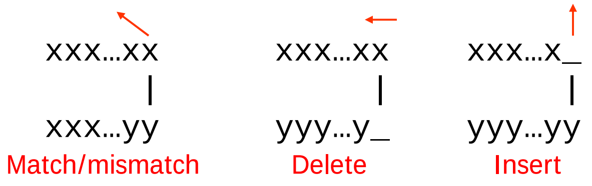
Needleman-Wunsch Algorithm (II)

Source: Ken Sung

- **Recurrence: For i>0, j>0**

$$V(i,j)=\max\begin{cases} V(i-1,j-1)+s(S[i],T[j]) & \textbf{Match / mismatch} \\ V(i-1,j)-\delta & \textbf{Delete} \\ V(i,j-1)-\delta & \textbf{Insert} \end{cases}$$

- **In the alignment, the last pair must be either match/mismatch, delete, insert**



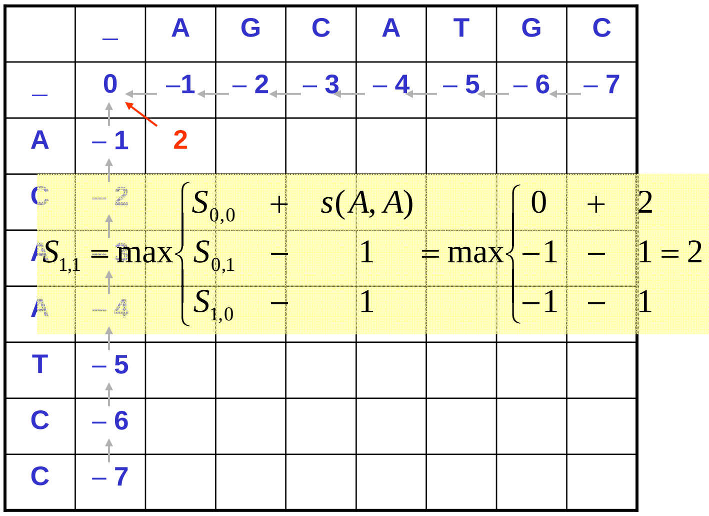
Example (I)

Source: Ken Sung

	_	A	G	C	A	T	G	C
_	0	–1	– 2	– 3	– 4	– 5	– 6	– 7
A	– 1							
C	– 2							
A	– 3							
A	– 4							
T	– 5							
C	– 6							
C	– 7							

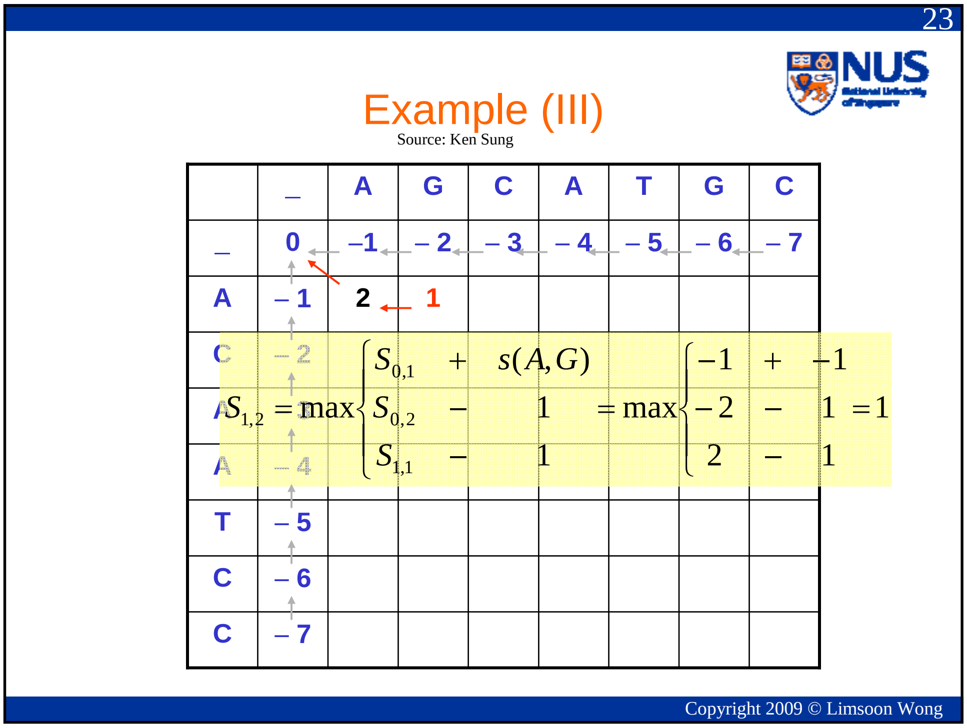
Example (II)

Source: Ken Sung



	_	A	G	C	A	T	G	C
_	0	–1	– 2	– 3	– 4	– 5	– 6	– 7
A	– 1	2						
C	– 2							
A	– 3							
A	– 4							
T	– 5							
C	– 6							
C	– 7							

$$S_{1,1} = \max\begin{cases} S_{0,0} & + & s(A,A) \\ S_{0,1} & - & 1 \\ S_{1,0} & - & 1 \end{cases} = \max\begin{cases} 0 & + & 2 \\ -1 & - & 1 = 2 \\ -1 & - & 1 \end{cases}$$

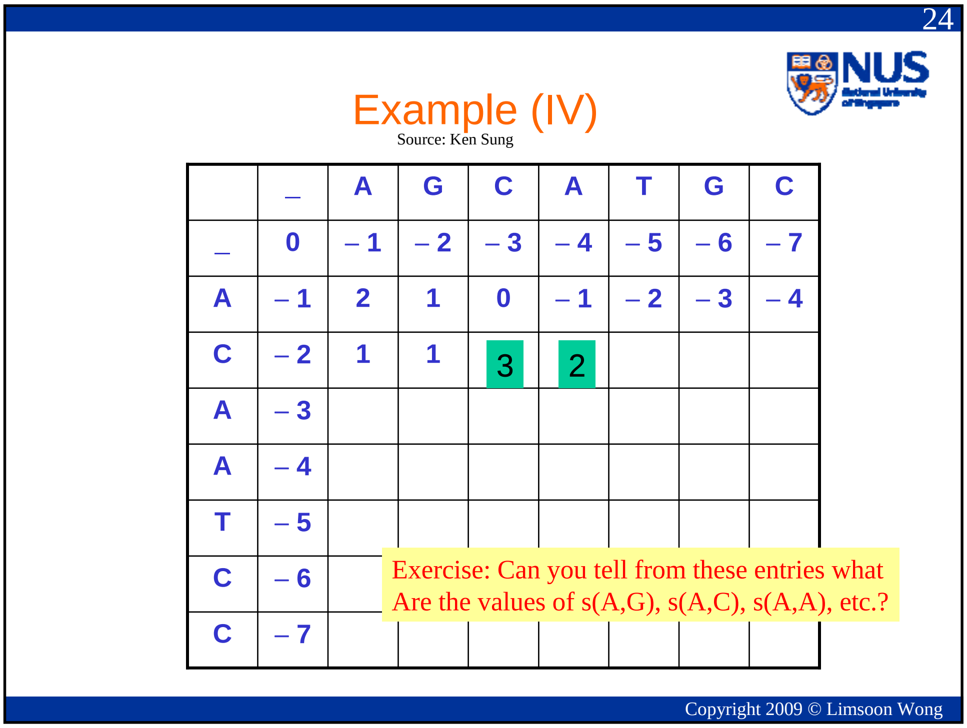


Example (III)

Source: Ken Sung

	_	A	G	C	A	T	G	C
_	0	–1	– 2	– 3	– 4	– 5	– 6	– 7
A	– 1	2	1					
C	– 2							
A	– 3							
A	– 4							
T	– 5							
C	– 6							
C	– 7							

$$S_{1,2} = \max\begin{cases} S_{0,1} & + & s(A,G) \\ S_{0,2} & - & 1 \\ S_{1,1} & - & 1 \end{cases} = \max\begin{cases} -1 & + & -1 \\ -2 & - & 1 \\ 2 & - & 1 \end{cases} = 1$$



Example (IV)

Source: Ken Sung

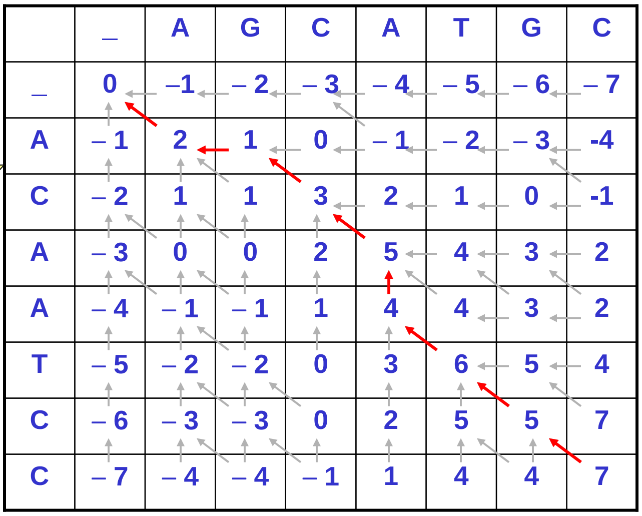
	_	A	G	C	A	T	G	C
_	0	– 1	– 2	– 3	– 4	– 5	– 6	– 7
A	– 1	2	1	0	– 1	– 2	– 3	– 4
C	– 2	1	1	3	2			
A	– 3							
A	– 4							
T	– 5							
C	– 6							
C	– 7							

Exercise: Can you tell from these entries what
Are the values of s(A,G), s(A,C), s(A,A), etc.?

Example (V)

Source: Ken Sung

What is the alignment corresponding to this?



	_	A	G	C	A	T	G	C
_	0	–1	– 2	– 3	– 4	– 5	– 6	– 7
A	– 1	2	1	0	– 1	– 2	– 3	-4
C	– 2	1	1	3	2	1	0	-1
A	– 3	0	0	2	5	4	3	2
A	– 4	– 1	– 1	1	4	4	3	2
T	– 5	– 2	– 2	0	3	6	5	4
C	– 6	– 3	– 3	0	2	5	5	7
C	– 7	– 4	– 4	– 1	1	4	4	7

Pseudo Codes

Source: Ken Sung

```
Create the table V[0..n,0..m] and P[1..n,1..m];
V[0,0] = 0;
For j=1 to m, set V[0,j] := v[0,j – 1] – δ ;
For i=1 to n, set V[i,0] := V[i – 1,0] – δ ;
For j=1 to m {
   For i = 1 to n {
        set V[i,j] := V[i,j – 1] – δ ;
        set P[i,j] := (0, – 1);
        if V[i,j] < V[i – 1,j] – δ then
                set V[i,j] := V[i – 1,j] – δ ;
                set P[i,j] := (– 1, 0);
        if (V[i,j] < V[i – 1, j – 1] + s(S[i],T[j])) then
                set V[i,j] := V[i – 1, j – 1] + s(S[i],T[j]);
                set P[i,j] := (– 1, – 1);
   }
}
Backtracking P[n,m] to P[0,0] to find optimal alignment;
```

Analysis

Source: Ken Sung

- **We need to fill in all entries in the n×m matrix**
- **Each entry can be computed in O(1) time**

⇒Time complexity = O(nm)

⇒Space complexity = O(nm)

Exercise: Write down the memoized version of Needleman-Wunsch. What is its time/space complexity?

Problem on Speed

Source: Ken Sung

- **Aho, Hirschberg, Ullman 1976**
  - If we can only compare whether two symbols are equal or not, the string alignment problem can be solved in Ω(nm) time
- **Hirschberg 1978**
  - If symbols are ordered and can be compared, the string alignment problem can be solved in Ω(n log n) time
- **Masek and Paterson 1980**
  - Based on Four-Russian's paradigm, the string alignment problem can be solved in O(nm/log2 n) time
- **Let d be the total number of inserts and deletes. Thus $0 \leq d \leq n+m$. If d is smaller than n+m, can we get a better algorithm? Yes!**

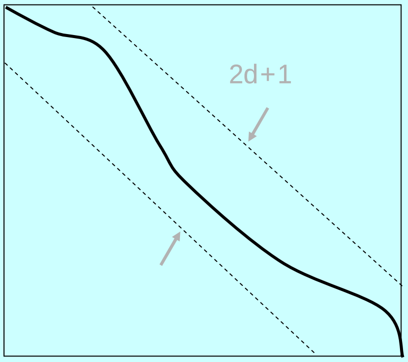
O(dn)-Time Algorithm

Source: Ken Sung

- **The alignment should be inside the 2d+1 band**

⇒No need to fill-in the lower and upper triangle

⇒Time complexity: O(dn)

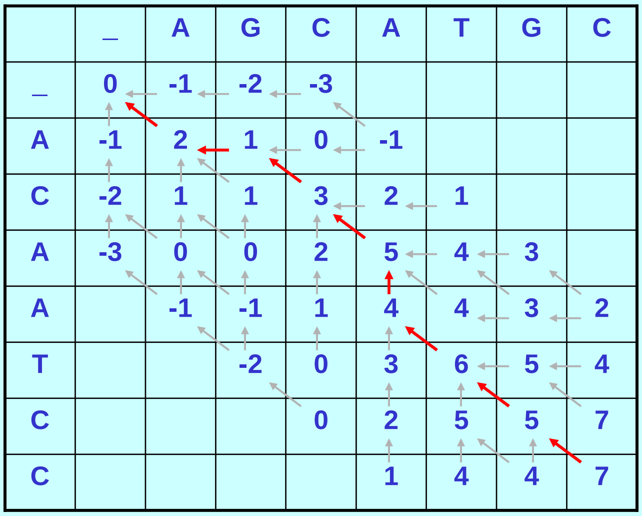


Example

- **d=3**

  A_CAATCC

  AGCA_TGC



	_	A	G	C	A	T	G	C
_	0	-1	-2	-3				
A	-1	2	1	0	-1			
C	-2	1	1	3	2	1		
A	-3	0	0	2	5	4	3	
A		-1	-1	1	4	4	3	2
T			-2	0	3	6	5	4
C				0	2	5	5	7
C					1	4	4	7

Recursive Equation for O(dn)-Time Algo

$$v(i, j, d) = \max \begin{cases} v(i-1, j-1, d) + s(S[i], S[j]) & \\ v(i-1, j, d-1) - \delta & \text{if } d > 0 \\ v(i, j-1, d-1) - \delta & \text{if } d > 0 \end{cases}$$

Exercise: Write down the base cases, the memoized version, and the non-recursive version.

Global Pairwise Alignment:

More Realistic Handling of Indels

- **In Nature, indels of several adjacent letters are not the sum of single indels, but the result of one event**
- **So reformulate as follows:**

Let $g(k)$ be the indel weight for an indel of k letters. Typically, $g(k) \leq k \cdot g(1)$. Let U and V be two sequences of length n and m. Then their global pairwise alignment can be extracted from the dynamic programming computation of $S_{n,m}$, where

$$S_{0,0} = 0, \quad S_{0,j} = -g(j), \quad S_{i,0} = -g(i)$$

$$S_{i,j} = \max \left\{ \begin{array}{l} S_{i-1,j-1} + s(u'_i, v'_j) \\ \max_{1 \leq k \leq j} \{S_{i,j-k} - g(k)\} \\ \max_{1 \leq k \leq i} \{S_{i-k,j} - g(k)\} \end{array} \right\}$$

Gap Penalty

Source: Ken Sung

- **g(q): $\mathbb{N} \rightarrow \Re$ is the penalty of a gap of length q**
- **Note g() is subadditive, i.e, g(p+q) ≤ g(p) + g(q)**

- **If g(k) = $\alpha + \beta k$, the gap penalty is called affine**
  - A penalty (α) for initiating the gap
  - A penalty (β) for the length of the gap

N-W Algorithm w/ General Gap Penalty (I)

Source: Ken Sung

- **Global alignment of S[1..n] and T[1..m]:**
  - Denote V(i, j) be the score for global alignment between S[1..i] and T[1..j]
  - Base cases:
    - **V(0, 0) = 0**
    - **V(0, j) = g(j)**
    - **V(i, 0) = g(i)**

N-W Algorithm w/ General Gap Penalty (II)

Source: Ken Sung

- **Recurrence for i>0 and j>0,**

$$V(i,j) = \max\begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \textbf{Match / mismatch} \\ \max\limits_{0 \le k \le j-1} \{V(i,k) + g(j-k)\} & \textbf{Insert T[k+1..j]} \\ \max\limits_{0 \le k \le i-1} \{V(k,j) + g(i-k)\} & \textbf{Delete S[k+1..i]} \end{cases}$$

Analysis

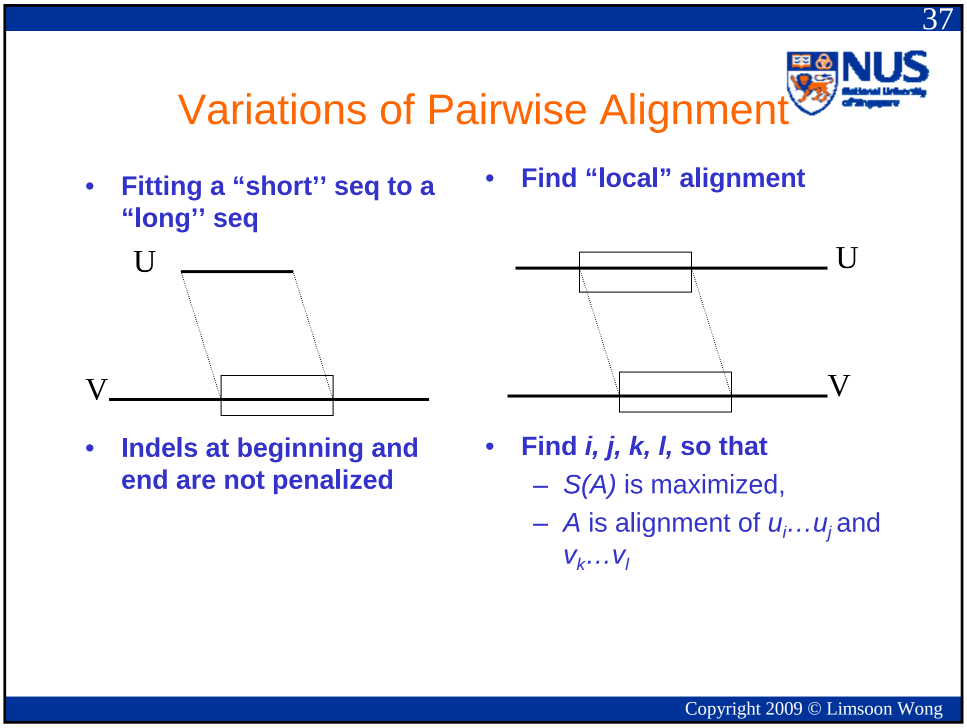
Source: Ken Sung

- **We need to fill in all entries in the n×m table**

- **Each entry can be computed in O(max{n, m}) time**

⇒ **Time complexity = O(nm max{n, m})**

⇒ **Space complexity = O(nm)**



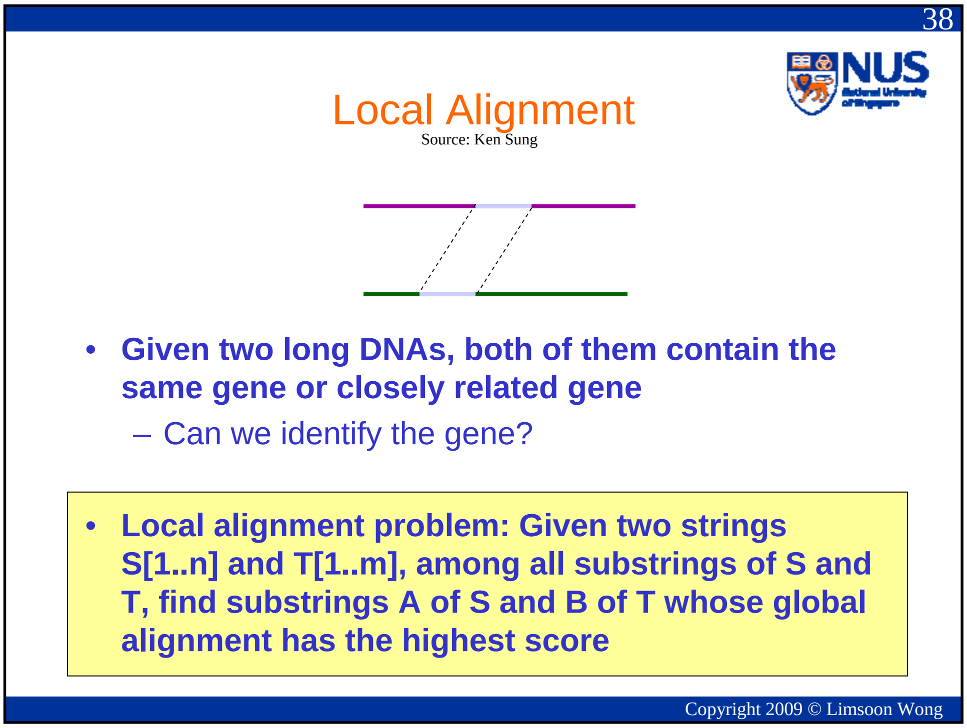
Variations of Pairwise Alignment

- **Fitting a “short’’ seq to a “long’’ seq**

- **Indels at beginning and end are not penalized**

- **Find “local” alignment**

- **Find *i, j, k, l,* so that**
  - $S(A)$ is maximized,
  - A is alignment of $u_i \dots u_j$ and $v_k \dots v_l$



Local Alignment

Source: Ken Sung

- **Given two long DNAs, both of them contain the same gene or closely related gene**
  - Can we identify the gene?

- **Local alignment problem: Given two strings S[1..n] and T[1..m], among all substrings of S and T, find substrings A of S and B of T whose global alignment has the highest score**

Brute-Force Solution

Source: Ken Sung

- **Algorithm:**
  - For every substring A of S, for every substring B of T, compute the global alignment of A and B
  - Return the pair (A, B) with the highest score
- **Time:**
  - There are n^2 choices of A and m^2 choices of B
  - Global alignment computable in $O(nm)$ time
  - In total, time complexity = $O(n^3m^3)$
- **Can we do better?**

Some Background

Source: Ken Sung

- **X is a suffix of S[1..n] if X=S[k..n] for some $k\geq 1$**
- **X is a prefix of S[1..n] if X=S[1..k] for some $k\leq n$**
- **E.g.**
  - Consider S[1..7] = ACCGATT
  - ACC is a prefix of S, GATT is a suffix of S
  - Empty string is both prefix and suffix of S

Which other string is both a prefix and suffix of S?

Dynamic Programming for Local Alignment Problem

Source: Ken Sung

- **Define V(i, j) be max score of global alignment of A and B over**
  - all suffixes A of S[1..i] and
  - all suffixes B of T[1..j]

- **Then, score of local alignment is**
  - $\max_{i,j} V(i,j)$

Smith-Waterman Algorithm

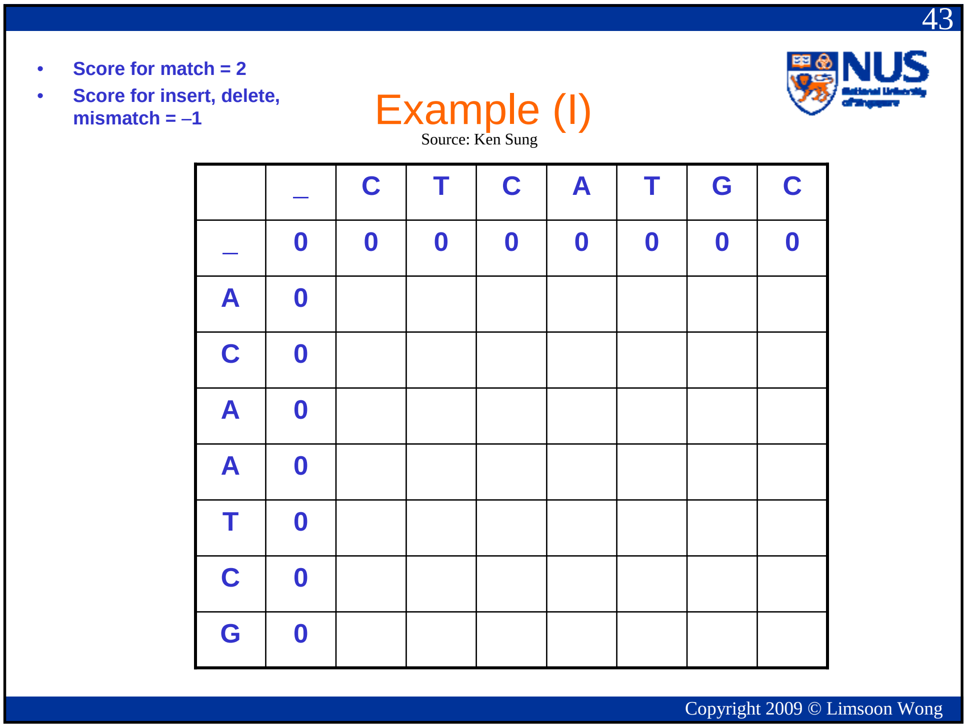
Source: Ken Sung

- **Basis:**

V(i, 0) = V(0, j) = 0

- **Recursion for i>0 and j>0:**

$$V(i,j) = \max \begin{cases} 0 & \textbf{Ignore initial segment} \\ V(i-1, j-1) + s(S[i], T[j]) & \textbf{Match / mismatch} \\ V(i-1, j) - \delta & \textbf{Delete} \\ V(i, j-1) - \delta & \textbf{Insert} \end{cases}$$

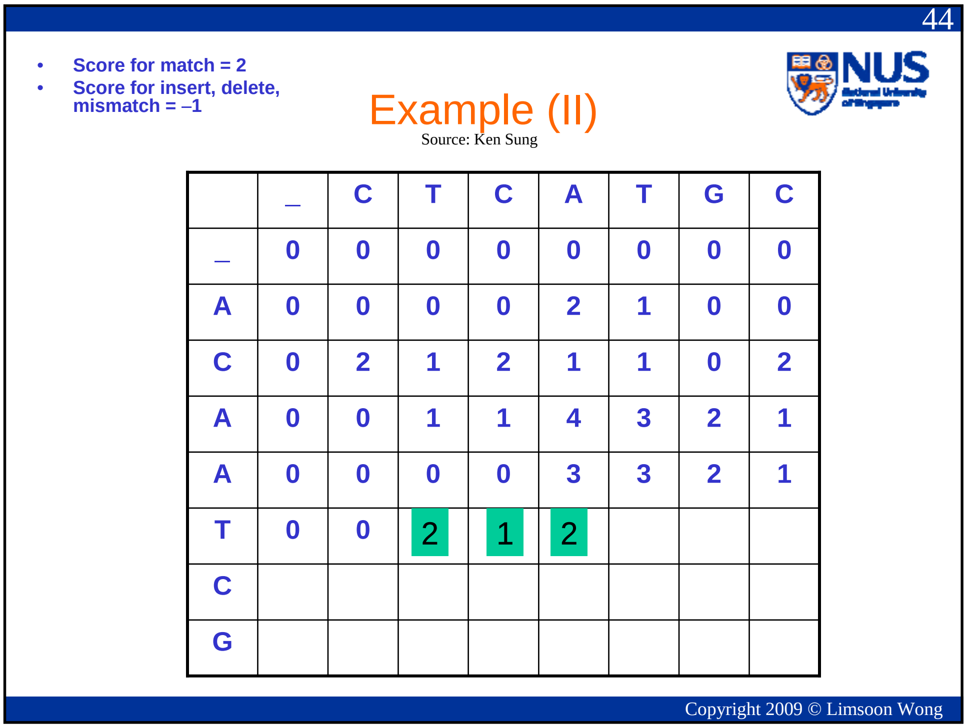


- Score for match = 2
- Score for insert, delete, mismatch = –1

Example (I)

Source: Ken Sung

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0							
C	0							
A	0							
A	0							
T	0							
C	0							
G	0							



- Score for match = 2
- Score for insert, delete, mismatch = –1

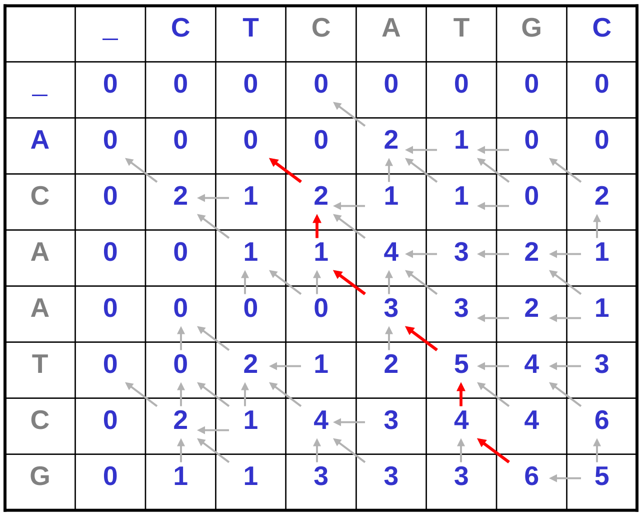
Example (II)

Source: Ken Sung

	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	0	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2			
C								
G								

Example (III)

Source: Ken Sung



	_	C	T	C	A	T	G	C
_	0	0	0	0	0	0	0	0
A	0	0	0	0	2	1	0	0
C	0	2	1	2	1	1	0	2
A	0	0	1	1	4	3	2	1
A	0	0	0	0	3	3	2	1
T	0	0	2	1	2	5	4	3
C	0	2	1	4	3	4	4	6
G	0	1	1	3	3	3	6	5

An optimal local alignment is

```
C_AT_G
CAATCG
```

What is the other optimal local alignment?

Analysis

Source: Ken Sung

- **Need to fill in all entries in the n×m matrix**
- **Each entries can be computed in O(1) time**
- **Finally, finding the entry with the max value**

⇒Time complexity = ??

⇒Space complexity = O(nm)

Exercise: What is the time complexity?

Multiple Sequence Alignment

What is a domain

- **A domain is a component of a protein that is self-stabilizing and folds independently of the rest of the protein chain**
  - Not unique to protein products of one gene; can appear in a variety of proteins
  - Play key role in the biological function of proteins
  - Can be "swapped" by genetic engineering betw one protein and another to make chimeras

- **May be composed of one, more than one, or not any structural motifs (often corresponding to active sites)**

Discovering Domain and Active Sites

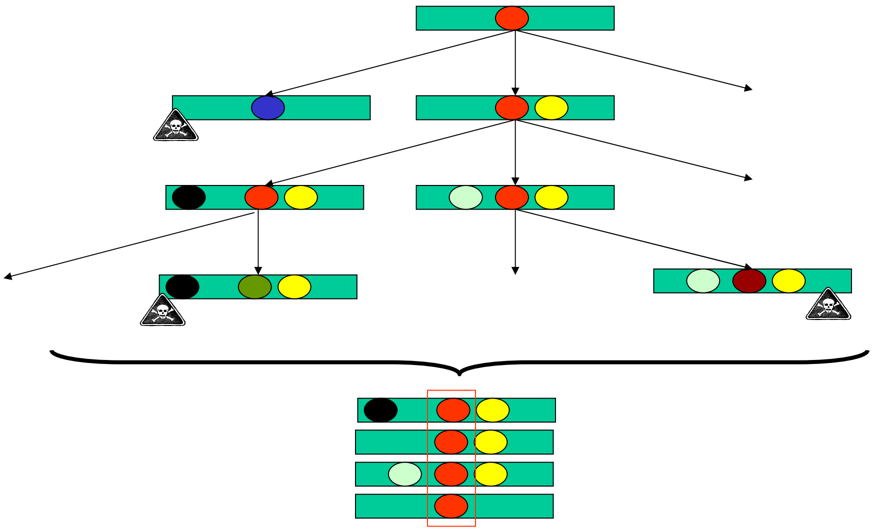
```
>gi|475902|emb|CAA83657.1| protein-tyrosine-phosphatase alpha
MDLWFFVLLLGSGLISVGATNVTTEPPTTVPTSTRIPTKAPTAAPDGGTTPRVSSLNVSSPMTTSAPASE
PPTTTATSISPNATTASLNASTPGTSVPTSAPVAISLPPSATPSALLTALPSTEAEMTERNVSATVTTQE
TSSASHNGNSDRRDETPIIAVMVALSSLLVIVFIIIVLYMLRFKKYKQAGSHSNSFRLPNGRTDDAEPQS
MPLLARSPSTNRKYPPLPVDKLEEEINRRIGDDNKLFREEFNALPACPIQATCEAASKEENKEKNRYVNI
LPYDHSRVHLTPVEGVPDSHYINTSFINSYQEKNKFIAAQGPKEETVNDFWRMIWEQNTATIVMVTNLKE
RKECKCAQYWPDQGCWTYGNIRVSVEDVTVLVDYTVRKFCIQQVGDVTNKKPQRLVTQFHFTSWPDFGVP
FTPIGMLKFLKKVKTCNPQYAGAIVVHCSAGVGRTGTFIVIDAMLDMMHAERKVDVYGFVSRIRAQRCQM
VQTDMQYVFIYQALLEHYLYGDTELEVTSLEIHLQKIYNKVPGTSSNGLEEEFKKLTSIKIQNDKMRTGN
LPANMKKNRVLQIIPYEFNRVIIPVKRGEENTDYVNASFIDGYRRRTPTCQPRPVQHTIEDFWRMIWEWK
SCSIVMLTELEERGQEKCAQYWPSDGSVSYGDINVELKKEEECESYTVRDLLVTNTRENKSRQIRQFHFH
GWPEVGIPSDGKGMINIIAAVQKQQQQSGNHPMHCHCSAGAGRTGTFCALSTVLERVKAEGILDVFQTVK
SLRLQRPHMVQTLEQYEFCYKVVQEYIDAFSDYANFK
```

- **How do we find the domain and associated active sites in the protein above?**

Domain/Active Sites as Emerging Patterns

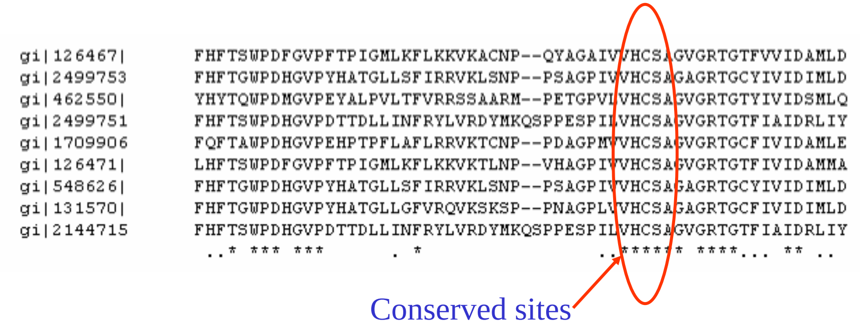
- **How to discover active site and/or domain?**
- **If you are lucky, domain has already been modelled**
  - – BLAST,
  - – HMMPFAM, …
- **If you are unlucky, domain not yet modelled**
  - – Find homologous seqs
  - – Do multiple alignment of homologous seqs
  - – Determine conserved positions
  - ⇒ Emerging patterns relative to background
  - ⇒ Candidate active sites and/or domains

In the course of evolution…



Multiple Alignment: An Example

- **Multiple seq alignment maximizes number of positions in agreement across several seqs**
- **seqs belonging to same "family" usually have more conserved positions in a multiple seq alignment**



```
gi|126467|      FHFTSWPDFGVPFTPIGMLKFLKKVKACNP--QYAGAIVVHCSAGVGRTGTFVVIDAMLD
gi|2499753      FHFTGWPDHGVPYHATGLLSFIRRVKLSNP--PSAGPIVVHCSAGAGRTGCYIVIDIMLD
gi|462550|      YHYTQWPDMGVPEYALPVLTFVRRSSAARM--PETGPVLVHCSAGVGRTGTYIVIDSMLQ
gi|2499751      FHFTSWPDHGVPDTTDLLINFRYLVRDYMKQSPPESPILVHCSAGVGRTGTFIAIDRLIY
gi|1709906      FQFTAWPDHGVPEHPTPFLAFLRRVKTCNP--PDAGPMVVHCSAGVGRTGCFIVIDAMLE
gi|126471|      LHFTSWPDFGVPFTPIGMLKFLKKVKTLNP--VHAGPIVVHCSAGVGRTGTFIVIDAMMA
gi|548626|      FHFTGWPDHGVPYHATGLLSFIRRVKLSNP--PSAGPIVVHCSAGAGRTGCYIVIDIMLD
gi|131570|      FHFTGWPDHGVPYHATGLLGFVRQVKSKSP--PNAGPLVVHCSAGAGRTGCFIVIDIMLD
gi|2144715      FHFTSWPDHGVPDTTDLLINFRYLVRDYMKQSPPESPILVHCSAGVGRTGTFIAIDRLIY
                 ..* *** ***      . *                ..****** ****... ** ..
```

Conserved sites

Multiple Alignment: Naïve Approach

- **Let S(A) be the score of a multiple alignment A. The optimal multiple alignment A of sequences $U_1, \ldots, U_r$ can be extracted from the following dynamic programming computation of $S_{m1,\ldots,mr}$:**

$$S_{m_1,\ldots,m_r} = \max_{\epsilon_1 \in \{0,1\},\ldots,\epsilon_r \in \{0,1\}} \left\{ \begin{array}{l} S_{m_1-\epsilon_1,\ldots,m_r-\epsilon_r} + \\ s(\epsilon_1 \cdot u'_{1,m_1}, \ldots, \epsilon_r \cdot u'_{r,m_r}) \end{array} \right\}$$

where

$$\epsilon_i \cdot a = \begin{cases} a & \text{if } \epsilon_i = 1 \\ - & \text{if } \epsilon_i = 0 \end{cases}$$

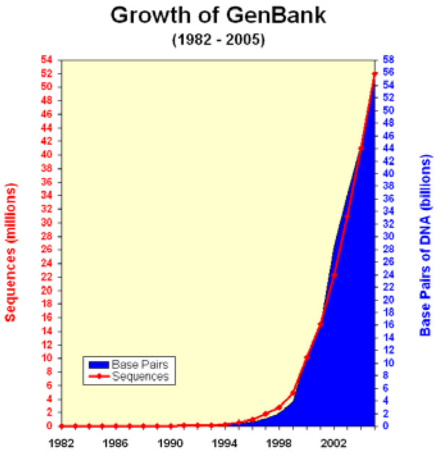
- **This requires $O(2^r)$ steps**

Exercise for the Brave:
Propose a practical approximation

Popular Tools for Sequence Comparison: FASTA, BLAST, Pattern Hunter

Scalability of Software

- **Increasing # of sequenced genomes: yeast, human, rice, mouse, fly, …**
- **S/w must be "linearly" scalable to large datasets**



Need Heuristics for Sequence Comparison

- **Time complexity for optimal alignment is O(n^2), where n is seq length**

⇒ **Given current size of seq databases, use of optimal algorithms is not practical for database search**

- **Heuristic techniques:**
  - BLAST
  - FASTA
  - Pattern Hunter
  - MUMmer, ...
- **Speed up:**
  - 20 min (optimal alignment)
  - 2 min (FASTA)
  - 20 sec (BLAST)

Exercise: Describe MUMer

Basic Idea: Indexing & Filtering

- **Good alignment includes short identical, or similar fragments**

⇒ Break entire string into substrings, index the substrings

⇒ Search for matching short substrings and use as seed for further analysis

⇒ Extend to entire string find the most significant local alignment segment

---

BLAST in 3 Steps

Altschul et al, *JMB* 215:403-410, 1990

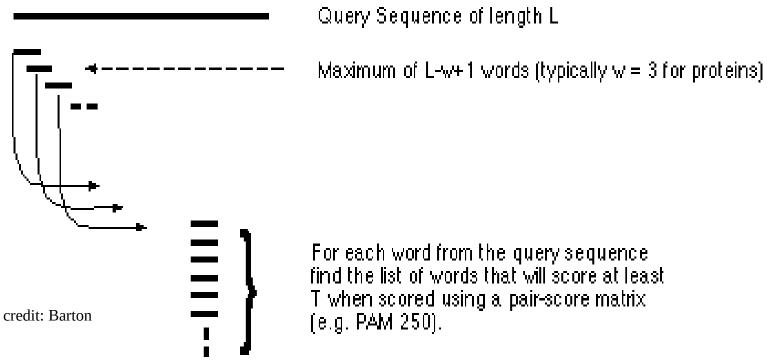
- **Similarity matching of words (3 aa's, 11 bases)**
  - No need identical words
- **If no words are similar, then no alignment**
  - Won't find matches for very short sequences
- **MSP: Highest scoring pair of segments of identical length. A segment pair is locally maximal if it cannot be improved by extending or shortening the segments**
- **Find alignments w/ optimal max segment pair (MSP) score**
- **Gaps not allowed**
- **Homologous seqs will contain a MSP w/ a high score; others will be filtered out**

BLAST in 3 Steps

Altschul et al, *JMB* 215:403-410, 1990

Step 1

- **For the query, find the list of high scoring words of length w**



Query Sequence of length L

Maximum of L-w+1 words (typically w = 3 for proteins)

For each word from the query sequence find the list of words that will score at least T when scored using a pair-score matrix (e.g. PAM 250).

Image credit: Barton

BLAST in 3 Steps

Altschul et al, *JMB* 215:403-410, 1990

Step 2

- **Compare word list to db & find exact matches**

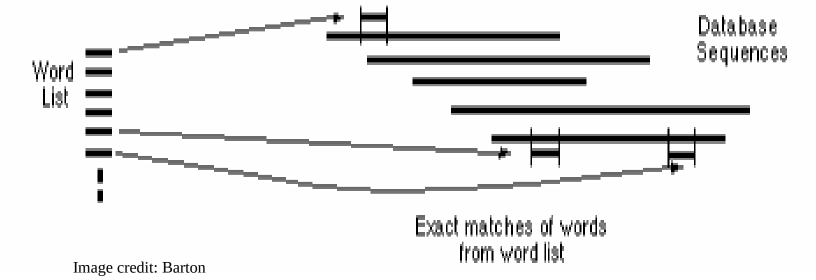


Image credit: Barton

BLAST in 3 Steps

Altschul et al, *JMB* 215:403-410, 1990

Step 3

- **For each word match, extend alignment in both directions to find alignment that score greater than a threshold s**

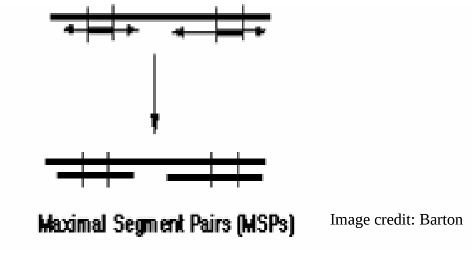


Image credit: Barton

Spaced Seeds

- **111010010100110111 is an example of a spaced seed model with**
  - 11 required matches (weight=11)
  - 7 “don’t care” positions

```
GAGTACTCAACACCAACATTAGTGGCAATGGAAAAT…
|| ||||||||| ||||| || |||||   ||||||
GAATACTCAACAGCAACACTAATGGCAGCAGAAAAT…
       111010010100110111
```

- **11111111111 is the BLAST seed model for comparing DNA seqs**

Observations on Spaced Seeds

- **Seed models w/ different shapes can detect different homologies**
  - the 3rd base in a codon “wobbles” so a seed like 110110110… should be more sensitive when matching coding regions

⇒ **Some models detect more homologies**
  - More sensitive homology search
  - PatternHunter I

⇒ **Use >1 seed models to hit more homologies**
  - Approaching 100% sensitive homology search
  - PatternHunter II

Exercise: Why does the 3rd base wobbles?

PatternHunter I

Ma et al., *Bioinformatics* 18:440-445, 2002

- **BLAST’s seed usually uses more than one hits to detect one homology**

⇒ **Wasteful**

```
TTGACCTCACC?
|||||||||||?
TTGACCTCACC?
11111111111
 11111111111
```

1/4 chances to have 2nd hit next to the 1st hit

- **Spaced seeds uses fewer hits to detect one homology**

⇒ **Efficient**

```
CAA?A??A?C??TA?TGG?
|||?|??|?|??||?|||?
CAA?A??A?C??TA?TGG?
111010010100110111
 111010010100110111
```

1/4^6 chances to have 2nd hit next to the 1st hit

PatternHunter I

Ma et al., *Bioinformatics* 18:440-445, 2002

Proposition. The expected number of hits of a weight-*W* length-*M* model within a length-*L* region of similarity *p* is $(L - M + 1) * p^W$

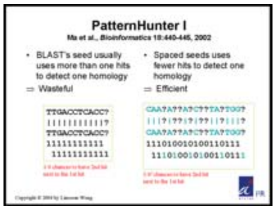
Proof.

For any fixed position, the prob of a hit is p^W.

There are $L - M + 1$ candidate positions.

The proposition follows.

Implication



PatternHunter I

Ma et al., *Bioinformatics* 18:440-445, 2002

- BLAST's seed usually uses more than one hits to detect one homology

⇒ Wasteful

```
TTGACCTCACC?
|||||||||||?
TTGACCTCACC?
11111111111
 11111111111
```

- Spaced seeds uses fewer hits to detect one homology

⇒ Efficient

```
CAA?A??A?C??TA?TGG?
|||?|??|?|??||?|||?
CAA?A??A?C??TA?TGG?
111010010100110111
 111010010100110111
```

PatternHunter I

Ma et al., *Bioinformatics* 18:440-445, 2002

Proposition. The expected number of hits of a weight-W length-M model within a length-L region of similarity p is $(L - M + 1) * p^W$

Proof. For any fixed position, the prob of a hit is p^W. There are L – M + 1 positions. The proposition follows.

- **For *L = 1017***
  - BLAST seed expects $(1017 - 11 + 1) * p^{11} = 1007 * p^{11}$ hits
  - But ~*1/4* of these overlap each other. So likely to have only ~$750 * p^{11}$ distinct hits
  - Our example spaced seed expects $(1017 - 18 + 1) * p^{11} = 1000 * p^{11}$ hits
  - But only $1/4^6$ of these overlap each other. So likely to have ~$1000 * p^{11}$ distinct hits

Spaced seeds likely to be more sensitive & more efficient

Sensitivity of PatternHunter I

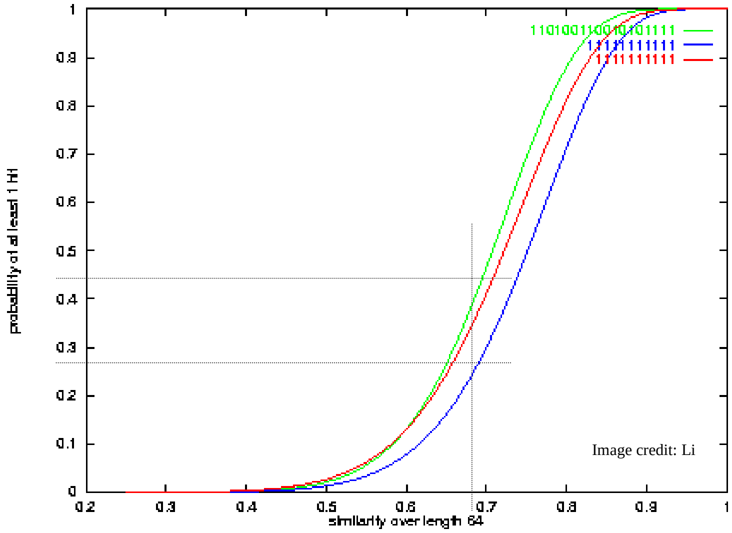
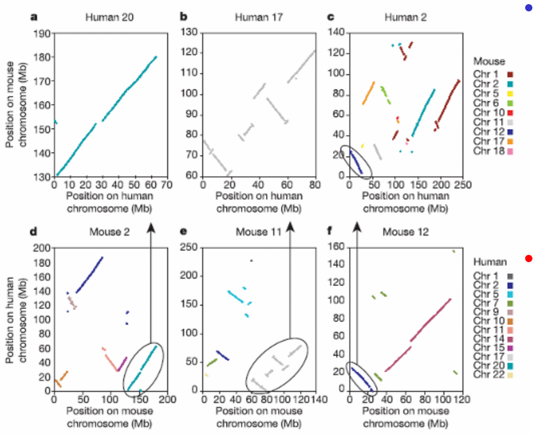


Image credit: Li

Speed of PatternHunter I



Nature, 420:520-522, 2002

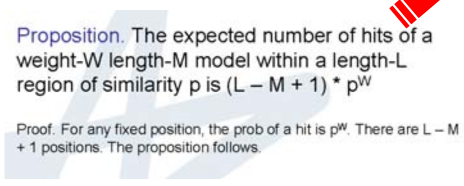
- **Mouse Genome Consortium used PatternHunter to compare mouse genome & human genome**
- **PatternHunter did the job in a 20 CPU-days --- it would have taken BLAST 20 CPU-years!**

How to Increase Sensitivity?

- **Ways to increase sensitivity:**
  - “Optimal” seed
  - Reduce weight by 1
  - Increase number of spaced seeds by 1
- **Intuitively, for DNA seq,**
  - Reducing weight by 1 will increase number of matches 4 folds
  - Doubling number of seeds will increase number of matches 2 folds
- **Is this really so?**

How to Increase Sensitivity?

- **Ways to increase sensitivity:**
  - “Optimal” seed
  - Reduce weight by 1
  - Increase number of spaced seeds by 1



Proposition. The expected number of hits of a weight-W length-M model within a length-L region of similarity p is $(L - M + 1) * p^W$

Proof. For any fixed position, the prob of a hit is p^W. There are L – M + 1 positions. The proposition follows.

- **For *L = 1017* & *p = 50%***
  - 1 weight-11 length-18 model expects *$1000/2^{11}$* hits
  - 2 weight-12 length-18 models expect *$2 * 1000/2^{12} = 1000/2^{11}$* hits
  - ⇒ When comparing regions w/ >50% similarity, using 2 weight-12 spaced seeds together is more sensitive than using 1 weight-11 spaced seed!

Exercise: Proof this claim

PatternHunter II

Li et al, *GIW*, 164-175, 2003

- **Idea**
  - Select a group of spaced seed models
  - For each hit of each model, conduct extension to find a homology
- **Selecting optimal multiple seeds is NP-hard**
- **Algorithm to select multiple spaced seeds**
  - Let A be an empty set
  - Let s be the seed such that A ∪ {s} has the highest hit probability
  - A = A ∪ {s}
  - Repeat until |A| = K
- **Computing hit probability of multiple seeds is NP-hard**

But see also Ilie & Ilie, “Multiple spaced seeds for homology search”, *Bioinformatics*, 23(22):2969-2977, 2007

Sensitivity of PatternHunter II

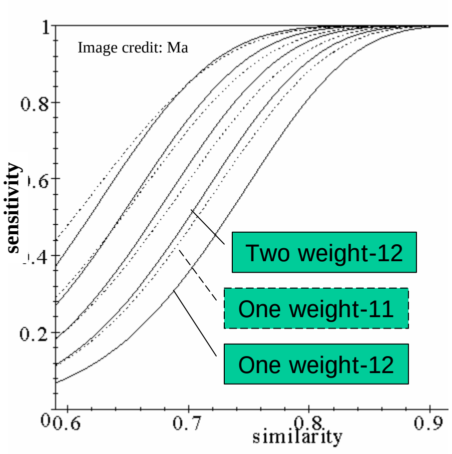


Image credit: Ma

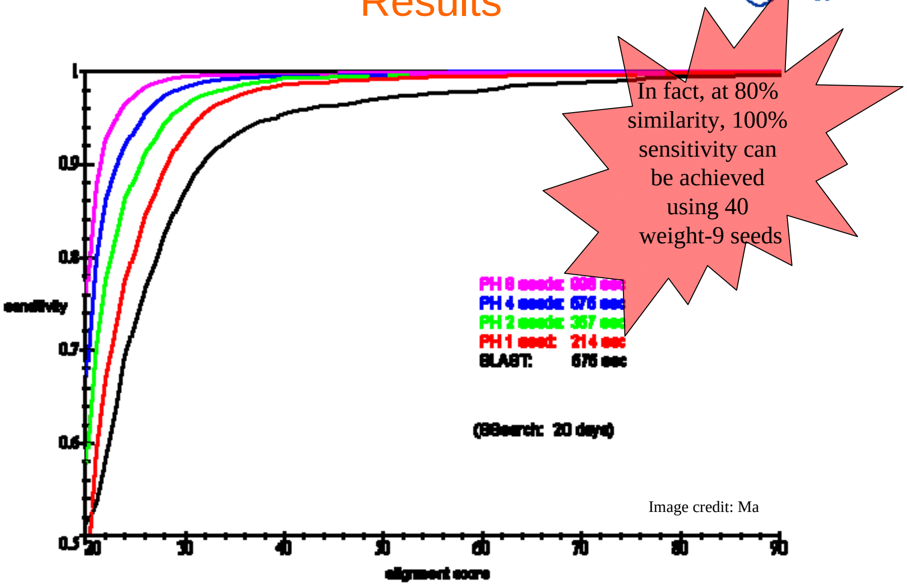
- **Solid curves: Multiple (1, 2, 4, 8,16) weight-12 spaced seeds**
- **Dashed curves: Optimal spaced seeds with weight = 11,10, 9, 8**

⇒ **“Double the seed number” gains better sensitivity than “decrease the weight by 1”**

Expts on Real Data

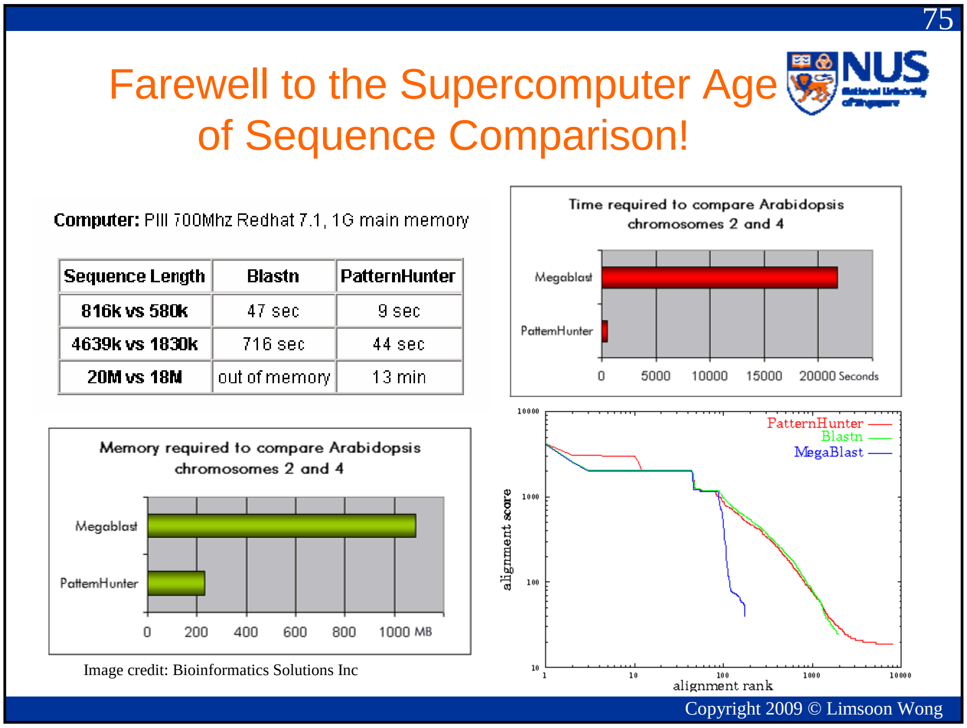
- **30k mouse ESTs (25Mb) vs 4k human ESTs (3Mb)**
  - downloaded from NCBI genbank
  - "low complexity" regions filtered out
- **SSearch (Smith-Waterman method) finds "all" pairs of ESTs with significant local alignments**
- **Check how many percent of these pairs can be "found" by BLAST and different configurations of PatternHunter II**

Results



In fact, at 80% similarity, 100% sensitivity can be achieved using 40 weight-9 seeds

Image credit: Ma



Farewell to the Supercomputer Age of Sequence Comparison!

Computer: PIII 700Mhz Redhat 7.1, 1G main memory

Sequence Length	Blastn	PatternHunter
816k vs 580k	47 sec	9 sec
4639k vs 1830k	716 sec	44 sec
20M vs 18M	out of memory	13 min

Image credit: Bioinformatics Solutions Inc

Concluding Remarks

What have we learned?

- **General methodology**
  - Dynamic programming
- **Dynamic programming applications**
  - Pairwise Alignment
    - **Needleman-Wunsch global alignment algorithm**
    - **Smith-Waterman local alignment algorithm**
  - Multiple Alignment
- **Important tactics**
  - Indexing & filtering (BLAST)
  - Spaced seeds (Pattern Hunter)

Any Question?

Acknowledgements

- **Some slides on popular sequence alignment tools are based on those given to me by Bin Ma and Dong Xu**
- **Some slides on Needleman-Wunsch and Smith-Waterman are based on those given to me by Ken Sung**

References

- S.F.Altshcul et al. “Basic local alignment search tool”, *JMB*, 215:403--410, 1990
- S.F.Altschul et al. “Gapped BLAST and PSI-BLAST: A new generation of protein database search programs”, *NAR*, 25(17):3389--3402, 1997
- S.B.Needleman, C.D.Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”, *JMB*, 48:444—453, 1970
- T.F.Smith, M.S.Waterman. “Identification of common molecular subsequences”, *JMB*, 147:195—197, 1981
- B. Ma et al. “PatternHunter: Faster and more sensitive homology search”, *Bioinformatics*, 18:440—445, 2002
- M. Li et al. “PatternHunter II: Highly sensitive and fast homology search”, *GIW*, 164—175, 2003
- D. Brown et al. “Homology Search Methods”, *The Practical Bioinformatician*, Chapter 10, pp 217—244, WSPC, 2004