Pairwise sequence alignment

Jiankui He
SUSTC, Shenzhen, China

Image by Nicolas Bouvier; courtesy of Genevieve Almouzni, Curie Institute, Paris, France

南方科技大学 李政道
SOUTH UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA

PPT adopted from John Hopkins Universi

Schedule

- Class I: lecture
- Class II: homework, programming
- Class III: Demo

Challenges in Computational Biology

4 Genome Assembly

5 Regulatory motif discovery

1 Gene Finding

DNA

2 **Sequence alignment**

6 Comparative Genomics

7 Evolutionary Theory

TCATGCTAT
TCGTGATAA
TGAGGATAT
TTATCATAT
TTATGATTT

3 Database lookup

8 Gene expression analysis

RNA transcript

9 Cluster discovery

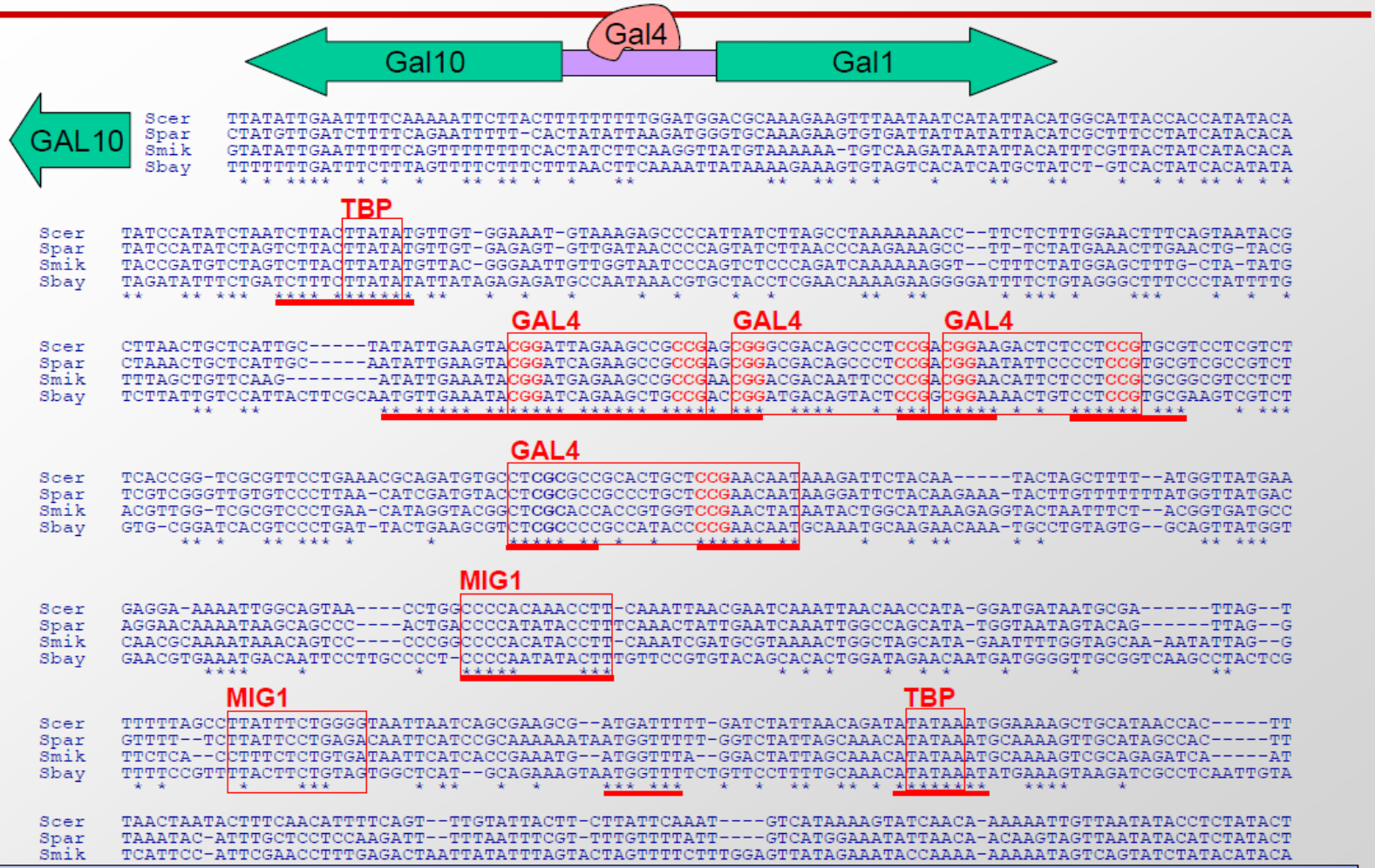
10 Gibbs sampling

11 Protein network analysis

12 Regulatory network inference

13 Emerging network properties

Evolution preserved functional elements!



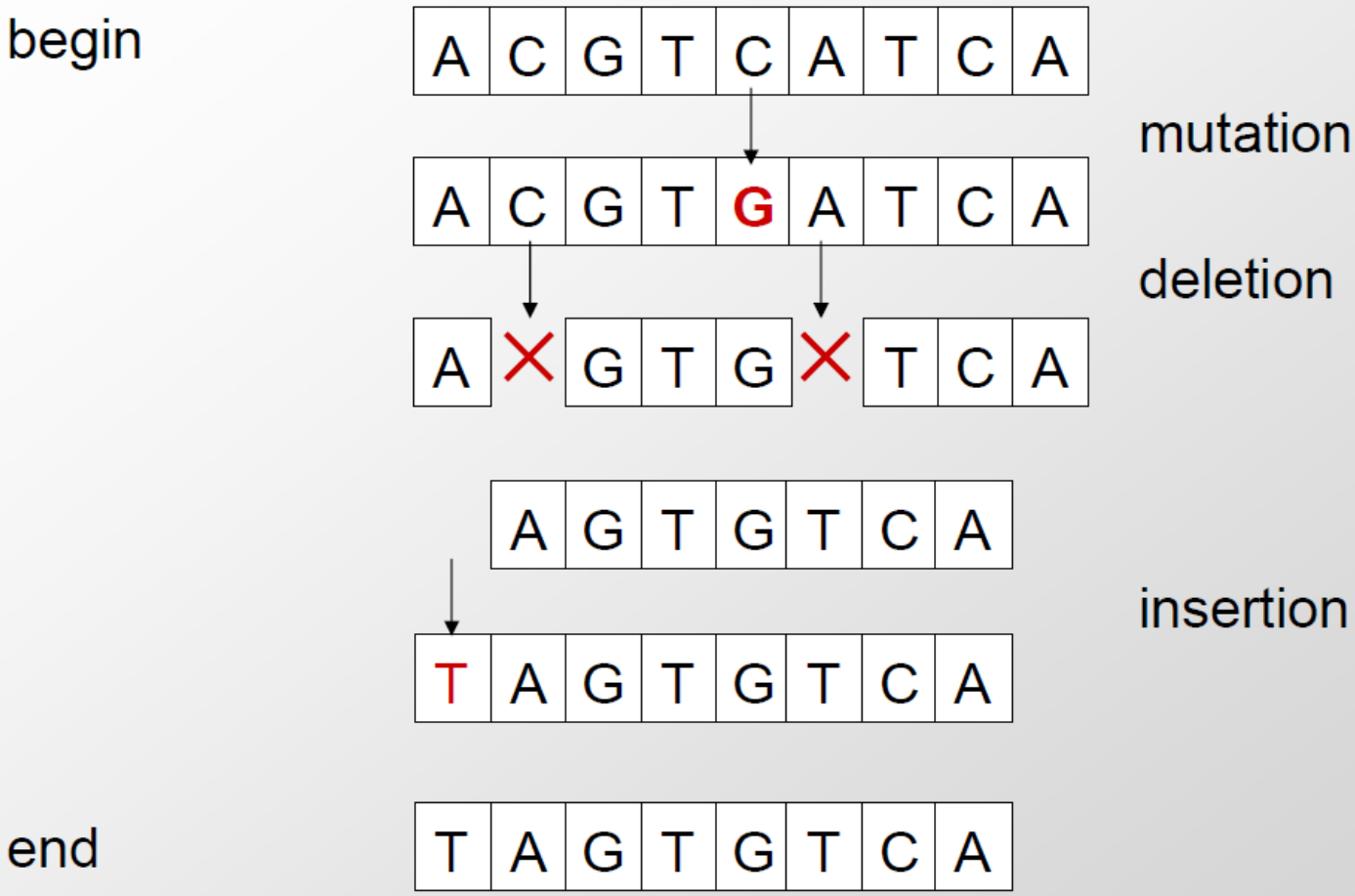
We can 'read' evolution to reveal functional elements

Kellis et al. Nature 2003

Today's goal:

How do we actually align two genes?

Genomes change over time



Goal of alignment: Infer edit operations

begin

A	C	G	T	C	A	T	C	A

?

end

T	A	G	T	G	T	C	A

Importance of sequence comparison

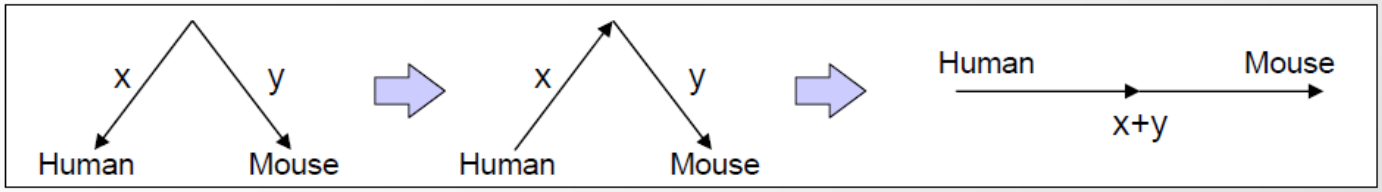
Discovering functional and evolutional relationships in biological sequences:

- Similar sequences → evolutionary relationship
- evolutionary relationship → related function
- Orthologs → same (almost same) function in different organisms.

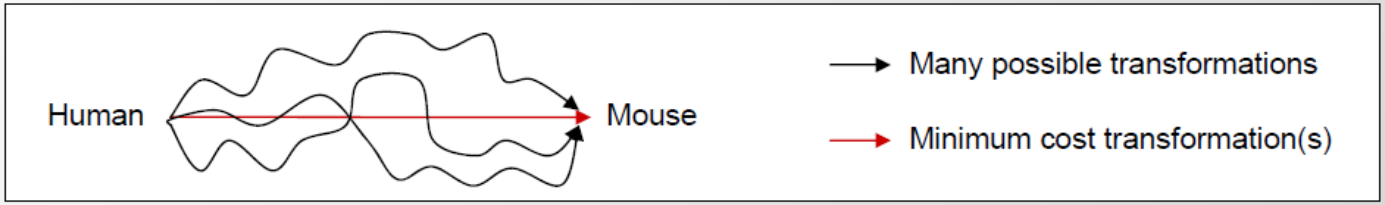
"→" should be read usually implies

From Bio to CS: Formalizing the problem

- Define set of evolutionary operations (insertion, deletion, mutation)
  - Symmetric operations allow time reversibility (part of design choice)

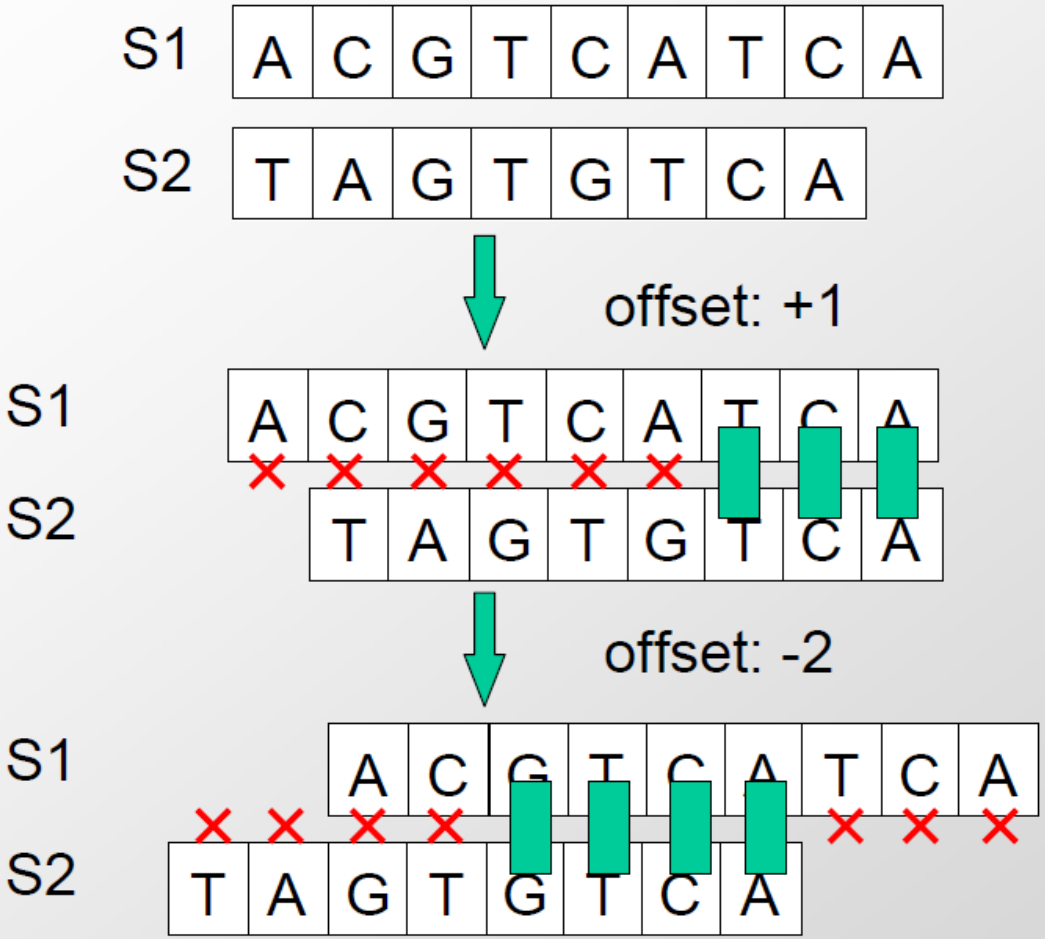


- Define optimality criterion (min number, min cost)
  - Impossible to infer exact series of operations (Occam's razor: find min)



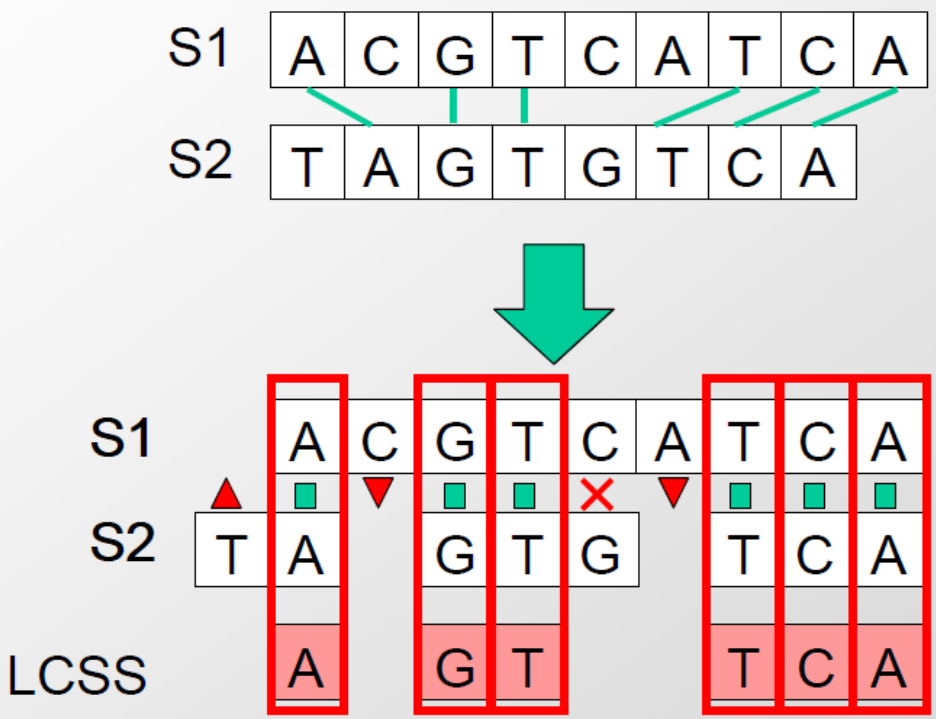
Formulation 1: Longest common substring

- Given two possibly related strings S1 and S2
  - What is the longest common substring? (no gaps)



Formulation 2: Longest common subsequence

- Given two possibly related strings S1 and S2
  - What is the longest common subsequence? (gaps allowed)



Edit distance:

- Number of changes needed for S1→S2
- Uniform scoring function

Formulation 3: Sequence alignment

- Allow gaps (fixed penalty)
  - Insertion & deletion operations
  - Unit cost for each character inserted or deleted
- Varying penalties for edit operations
  - Transitions (Pyrimidine⇔Pyrimidine, Purine⇔Purine)
  - Transversions (Purine ⇔ Pyrimidine changes)
  - Polymerase confuses Aw/G and Cw/T more often

Scoring function:

Match(x,x) = +1

Mismatch(A,G)= -½

Mismatch(C,T)= -½

Mismatch(x,y) = -1

	A	G	T	C
A	+1	-½	-1	-1
G	-½	+1	-1	-1
T	-1	-1	+1	-½
C	-1	-1	-½	+1

purine pyrimid.

Transitions:

A⇔G, C⇔T common (lower penalty)

Transversions:

All other operations

Etc…
(e.g. varying gap penalties)

How can we compute best alignment

S1	A	C	G	T	C	A	T	C	A
S2	T	A	G	T	G	T	C	A	

- Given additive scoring function:
  - Cost of mutation (AG, CT, other)
  - Cost of insertion / deletion
  - Reward of match
- Need algorithm for inferring best alignment
  - Enumeration?
  - How would you do it?
  - How many alignments are there?

Can we simply enumerate all possible alignments?

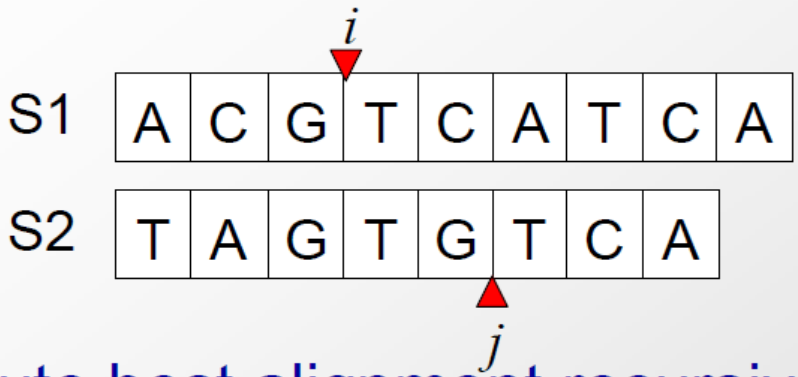
- Ways to align two sequences of length m, n

$$\binom{n+m}{m} = \frac{(m+n)!}{(m!)^2} \approx \frac{2^{m+n}}{\sqrt{\pi \cdot m}}$$

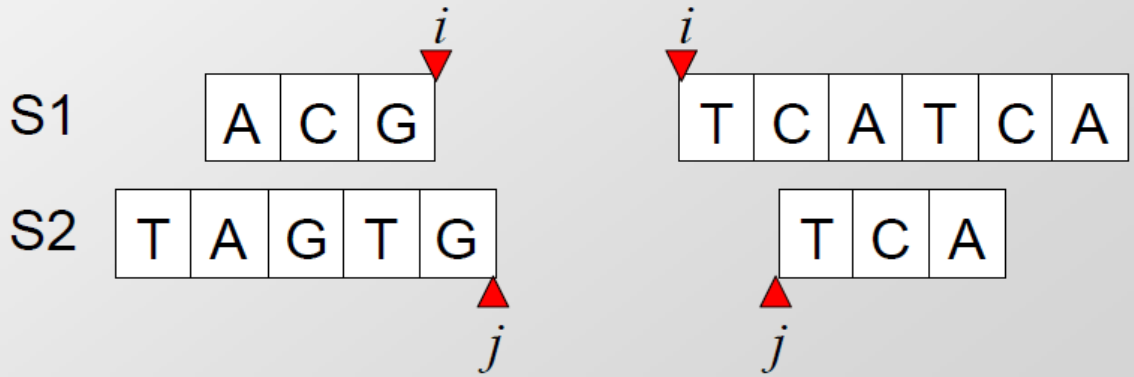
- For two sequences of length n

n	Enumeration	Today's lecture
10	184,756	100
20	1.40E+11	400
100	9.00E+58	10,000

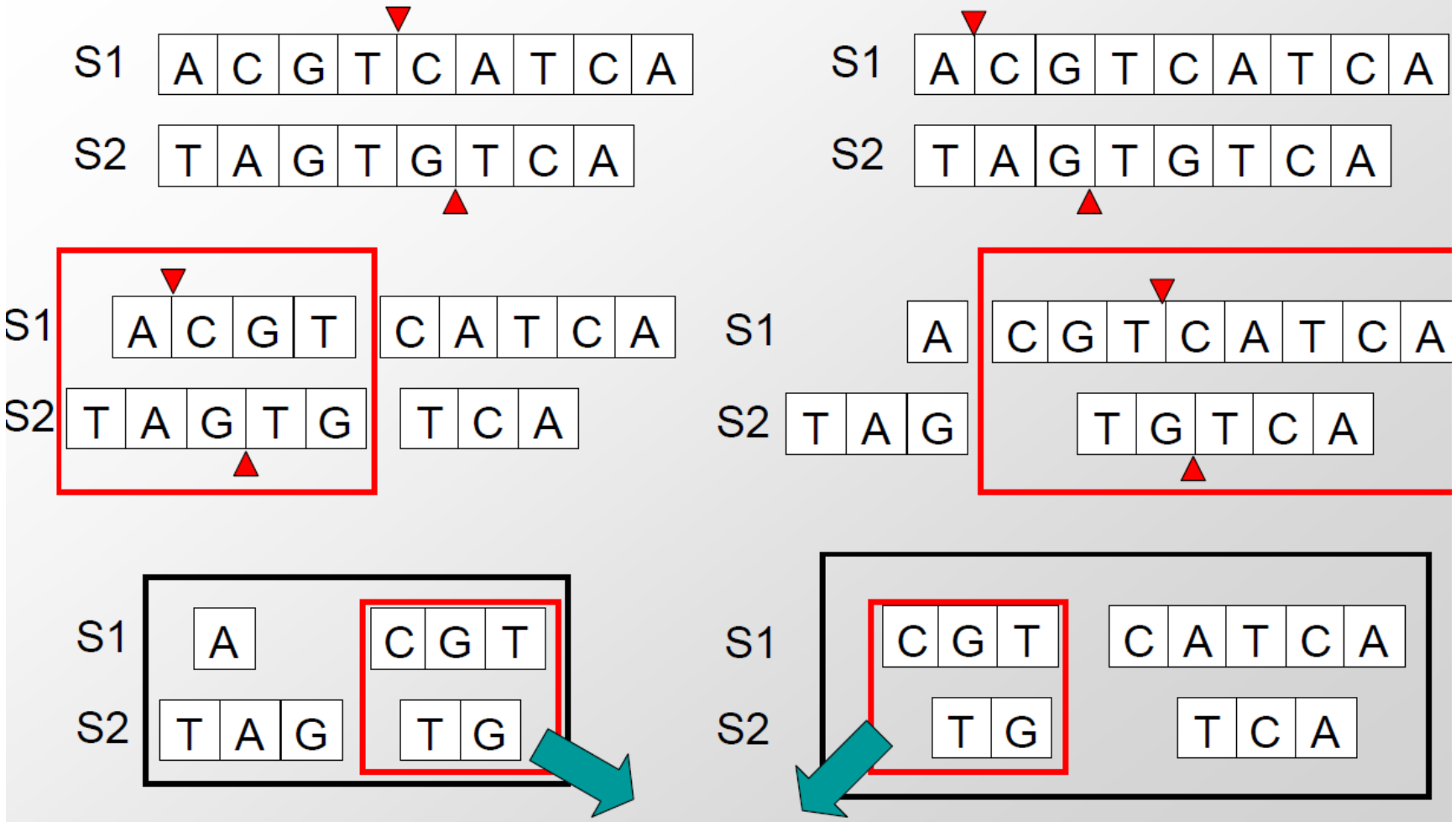
Key insight: score is additive!



- Compute best alignment recursively
  - For a given aligned pair ***(i, j)***, the best alignment is:
    - Best alignment of S1[1..i] and S2[1..j]
    - + Best alignment of S1[i..n] and S2[j..m]
  - Proof: cut-and-paste argument (see 6.046)



Key insight: re-use computation



Identical sub-problems! We can reuse our work!

Solution #1 – Memoization

- Create a big dictionary, indexed by aligned seqs
  - When you encounter a new pair of sequences
  - If it is in the dictionary:
    - Look up the solution
  - If it is not in the dictionary
    - Compute the solution
    - Insert the solution in the dictionary
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!

Top down approach

Solution #2 – Dynamic programming

- Create a big table, indexed by (i,j)
  - Fill it in from the beginning all the way till the end
  - You know that you'll need every subpart
  - Guaranteed to explore entire search space
- Ensures that there is no duplicated work
  - Only need to compute each sub-alignment once!
- Very simple computationally!

Bottom up approach

Identity score

Let (x,y) be an aligned pair of elements of two sequences (at least one of x,y must not be a gap).

$$id(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y \end{cases}$$

Score of an alignment = sum of scores of aligned pairs

```
TGK - G
AGKVG
-------
0+1+1+0+1 = 3
```

60 % identical

Identity score =number of matched elements / length of alignment

Gap penalties

Consider two pairs of alignments:

```
ATCG        AT – C G
ATTG   and  AT T  - G
```

They have the same identity score but alignment on the left is more likely to be correct

```
ATC - -  T A   and   AT - C  - T A
ATT T T TA           AT T T T TA
```

- The first problem is corrected by introducing "gap penalty".
- Second problem is corrected by introducing additional penalty for opening a gap.

Example

Score the above alignment using identity score; gap penalty = 1

Gap opening penalty = 2

```
ATCG
ATTG
```

1+1+0+1=3

```
AT – C G
AT T  - G
```

1+1-2-1-2-1+1=-3

```
ATC - -  T A
ATT T T TA
```

1+1+0-2-1-1+1+1=0

```
AT - C  - T A
AT T T T TA
```

1+1-2-1+0-2-1+1+1=-2

Problems with identity score

- In the two pairs of aligned sequence below there are mutations at the first and 6th position and insertion (or deletion) on the 4th position. However while V and A share significant biophysical similarity and we often see mutation between them, W and A do not often substitute one for the other.

  VGK – GI… WGK – GI…

  AGKVGL… AGKVGL

- What if I mutated to V and then back to I should this have the same score as when I was unchanged?

If we will like to use the score to estimate evolutionary distances it would be important to take into account possibility of such double mutation.

Second base

First base	U	C	A	G	Third base
U	UUU, UUC — Phenylalanine F UUA, UUG — Leucine L	UCU, UCC, UCA, UCG — Serine S	UAU, UAC — Tyrosine Y UAA — Stop codon UAG — Stop codon	UGU, UGC — Cysteine C UGA — Stop codon UGG — Tryptophan W	U C A G
C	CUU, CUC, CUA, CUG — Leucine L	CCU, CCC, CCA, CCG — Proline P	CAU, CAC — Histidine H CAA, CAG — Glutamine Q	CGU, CGC, CGA, CGG — Arginine R	U C A G
A	AUU, AUC, AUA — Isoleucine I AUG — Methionine start codon M	ACU, ACC, ACA, ACG — Threonine T	AAU, AAC — Asparagine N AAA, AAG — Lysine K	AGU, AGC — Serine S AGA, AGG — Arginine R	U C A G
G	GUU, GUC, GUA, GUG — Valine V	GCU, GCC, GCA, GCG — Alanine A	GAU, GAC — Aspartic acid D GAA, GAG — Glutamic acid E	GGU, GGC, GGA, GGG — Glycine G	U C A G

Scoring Matrices

An amino-acid scoring matrix is a 20x20 table such that positio
indexed with amino-acids so that position X,Y in the table gives
score of aligning amino-acid X with amino-acid Y

Identity matrix – Exact matches receive one score and non-exa
matches a different score (1 on the diagonal 0 everywhere else)
Mutation data matrix – a scoring matrix compiled based on observation of protein mutation rates: some mutations are observed more often then other (PAM, BLOSUM).
Not used:
Physical properties matrix – amino acids with similar biophysical properties receive high score.
Genetic code matrix – amino acids are scored based on similarities in the coding triple.

Principles of Dynamic programming

- Need to figure out how to use solution to smaller problems for solving larger problem.
- We need to keep a reasonable bound on how many sub-problems we solve
- Make sure that each sub-problem is solved only once

Dynamic programming algorithm for computing the score of the best alignmen

For a sequence $S = a_1, a_2, \ldots, a_n$ let $S_j = a_1, a_2, \ldots, a_j$

S,S' – two sequences

$Align(S_i,S'_j)$ = the score of the highest scoring alignment betwee S^1_i, S^2_j

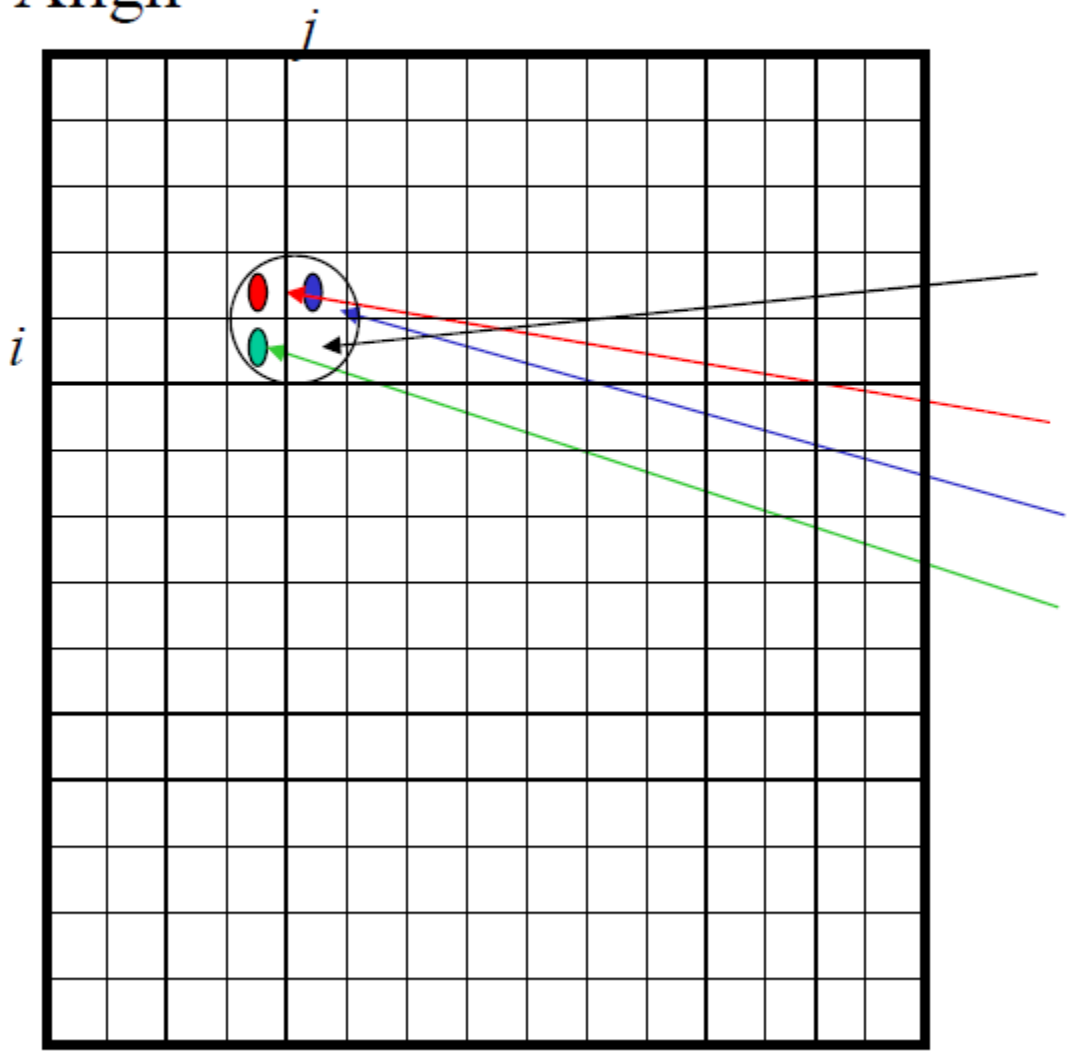
$S(a_i, a'_j)$= similarity score between amino acids a_i and a_j given by a scoring matrix like PAM, BLOSUM

g – gap penalty

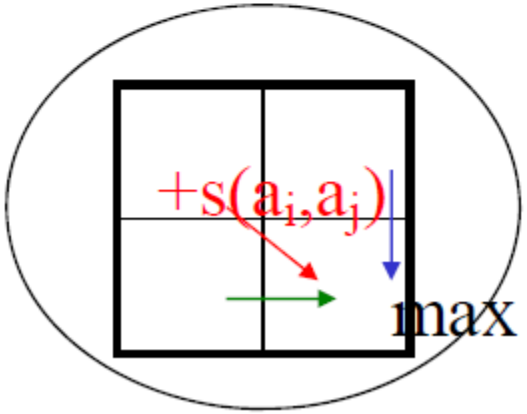
$$Align(S_i,S'_j) = \max \begin{cases} Align(S_{i-1},S'_{j-1}) + S(a_i, a'_j) \\ Align(S_i,S'_{j-1}) - g \\ Align(S_{i-1},S'_j) - g \end{cases}$$

Organizing the computation – dynamic programming table

Align



$$\text{Align}(i,j) = \text{Align}(S_i, S'_j) = \max \begin{cases} \text{Align}(S_{i-1}, S'_{j-1}) + s(a \\ \text{Align}(S_{i-1}, S'_j) - g \\ \text{Align}(S_i, S'_{j-1}) - g \end{cases}$$

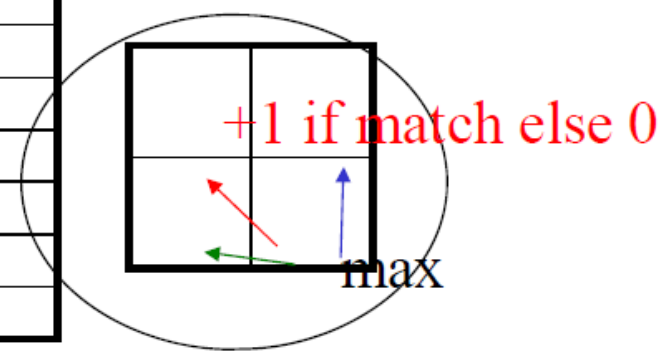


Example of DP computation with g = 0; match = 1; mismatch=0

Maximal Common Subsequence

initialization

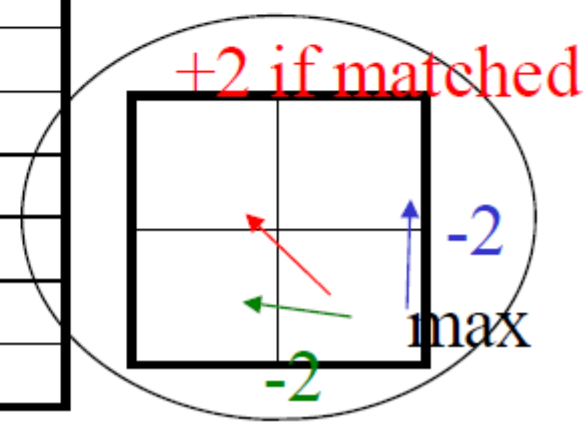
		A	T	T	G	C	G	C	G	C	A	T
	0	0	0	0	0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1	1	1	1	1	1
T	0	1	2	2	2	2	2	2	2			
G	0	1	2									
C	0	1										
T	0	1										
T	0	1										
A	0	1										
A	0	1										
C	0	1										
C	0	1										
A	0	1										



Example of DP computation with g = 2 match = 2; mismatch = -1

Initialization (penalty for starting with a gap)

		A	T	T	G	C	G	C	G	C	A	T
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
A	-2	2	0	-2								
T	-4	0	4									
G	-6			6								
C	-8											
T	-10											
T	-12											
A	-14											
A	-16											
C	-18											
C	-20											
A	-22											



The iterative algorithm

```
m = |S|; n = |S'|
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
          A[i,j]←max (
                          A[i-1,j] – g
                          A[i-1,j-1] + s(i,j)
                          A[i,j-1] – g
                          )
return(A[m,n])
```

Complexity of the algorithm

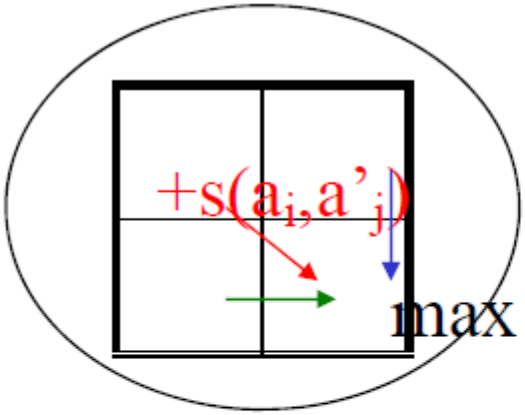
- Time O(nm); Space O(nm) where n, m the lengths of the two sequences.
- Space complexity can be reduced to O(n) by not storing the entries of dynamic programming table that are no longer needed for the computation (keep current row and the previous row only).

From computing the score to computing of the alignment

Desired output:

Sequence of substitutions/insertion/deletions leading to the op score.

ATTGCGTTATAT
AT- GCG- TATAT



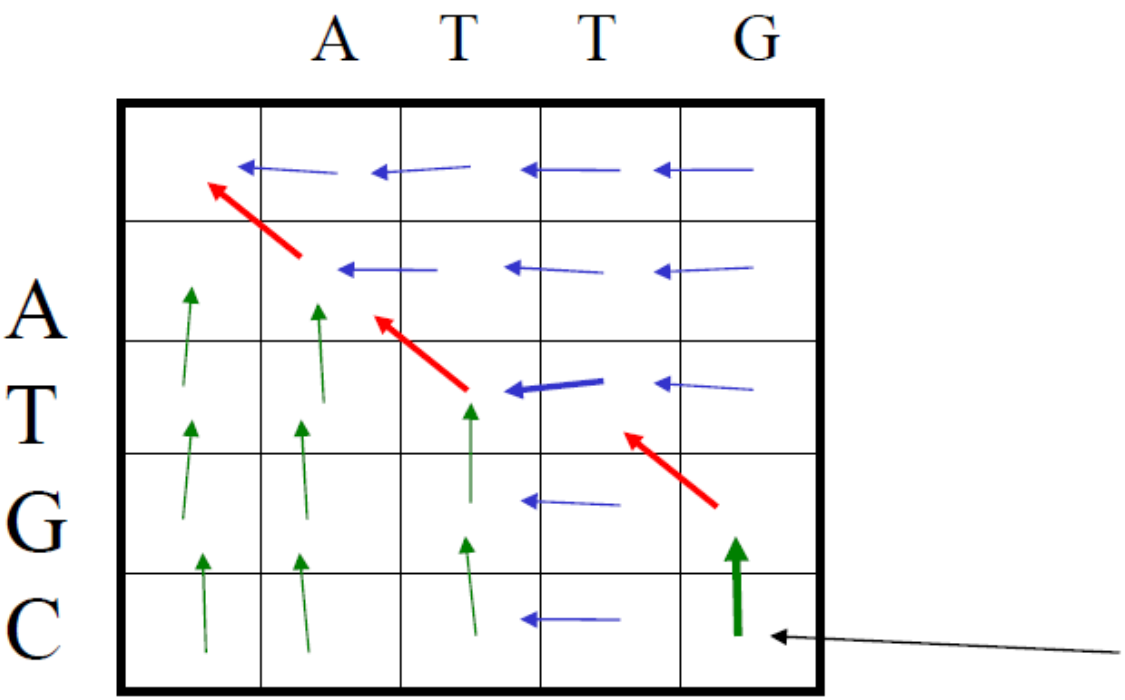
Red direction = mach
Blue direction = gap in horizontal sequence
Green direction = gap in vertical sequence

$a_1, a_2, \ldots\ a_i$
$a'_1, a'_2, \ldots\ a'_j$

$a_1, a_2, \ldots, a_i$ -
$a'_1, a'_2, \ldots, a'_j$

$a_1, a_2, \ldots\ldots\ a_i$
$a'_1, a'_2, \ldots, a'_j$ -

Recovering the path



Start path from here!

A T T G -

A T - G C

If at some position several choices lead to the same max value, the path need not be unique.

Local alignment

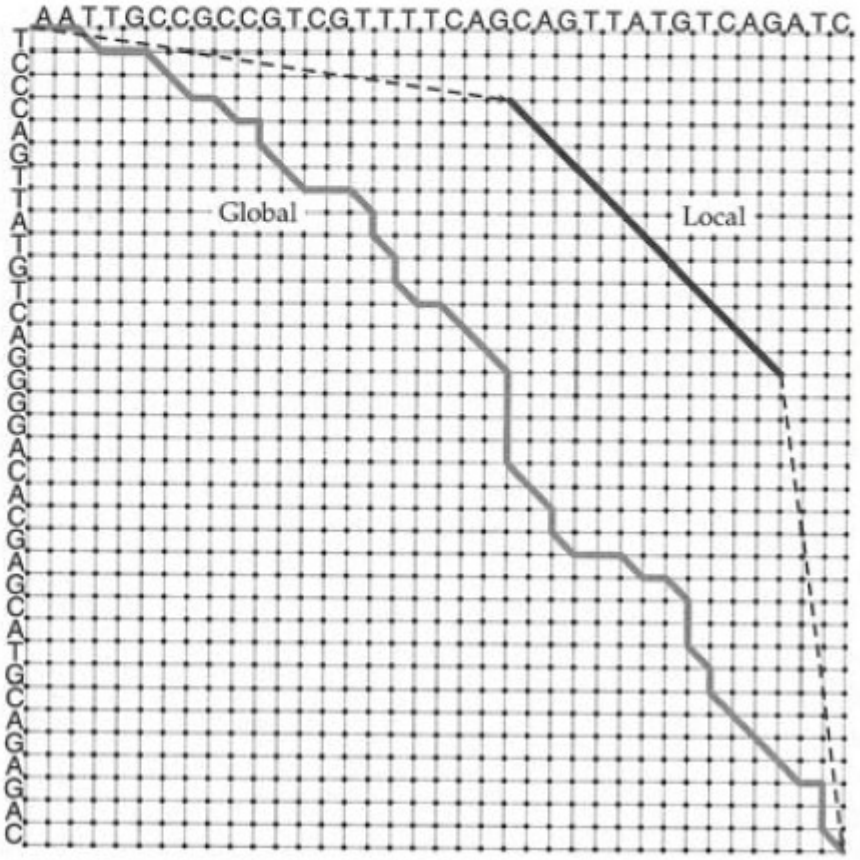
- The alignment techniques considered so far worked well for sequences which are similar over all their length
- This does not need to be the case: example gene from hox family have very short but highly conserved subsequence – the so called hox domain.
- Considered so far global alignment methods (that is algorithm that try to find the best alignment over whole length can miss this local similarity region

Global

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
 |  || |  ||  | | | |||    || |  | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
```

Local

```
                    tccCAGTTATGTCAGggacacgagcatgcagaga
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```



Local alignment (Smith, Waterman)

So far we have been dealing with global alignment.
Local alignment – alignment between substrings.
Main idea: If alignment becomes to bad – drop it.

Set p and g so that alignment of random strings gives negative score

$$a[i,j] = \max \begin{cases} a[i-1,j-1] + s(a_i, a_j) \\ a[i-1,j] + g \\ a[i,j-1] + g \\ 0 \end{cases}$$

Finding the alignment: find the highest scoring cell and trace it back

Example

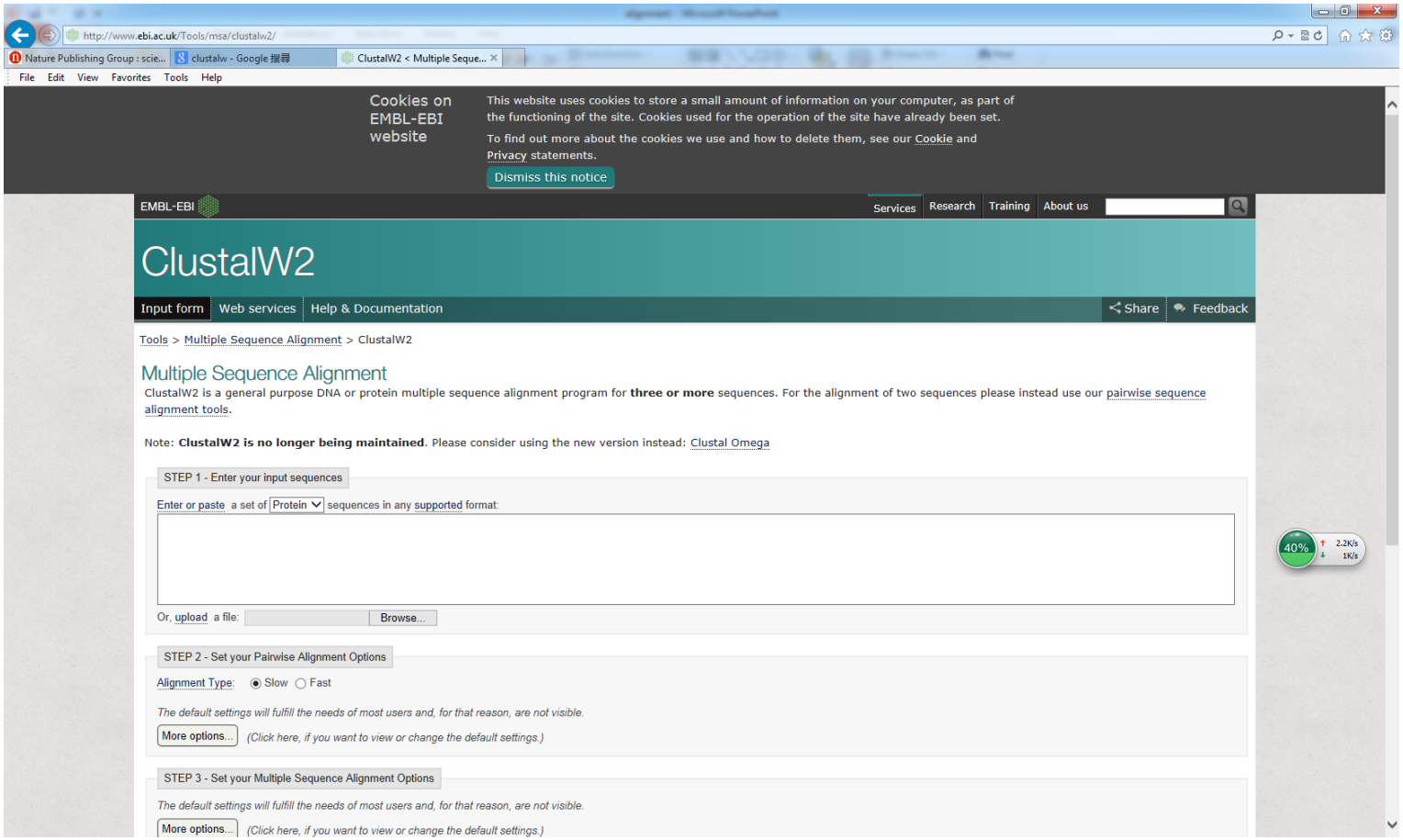
- Sequence 1 = ACACACTA
- Sequence 2 = AGCACACA

$$s(a,b) = +2 \text{ if } a = b(\text{match}),\ -1 \text{ if } a \neq b \text{ (mismatch)}$$

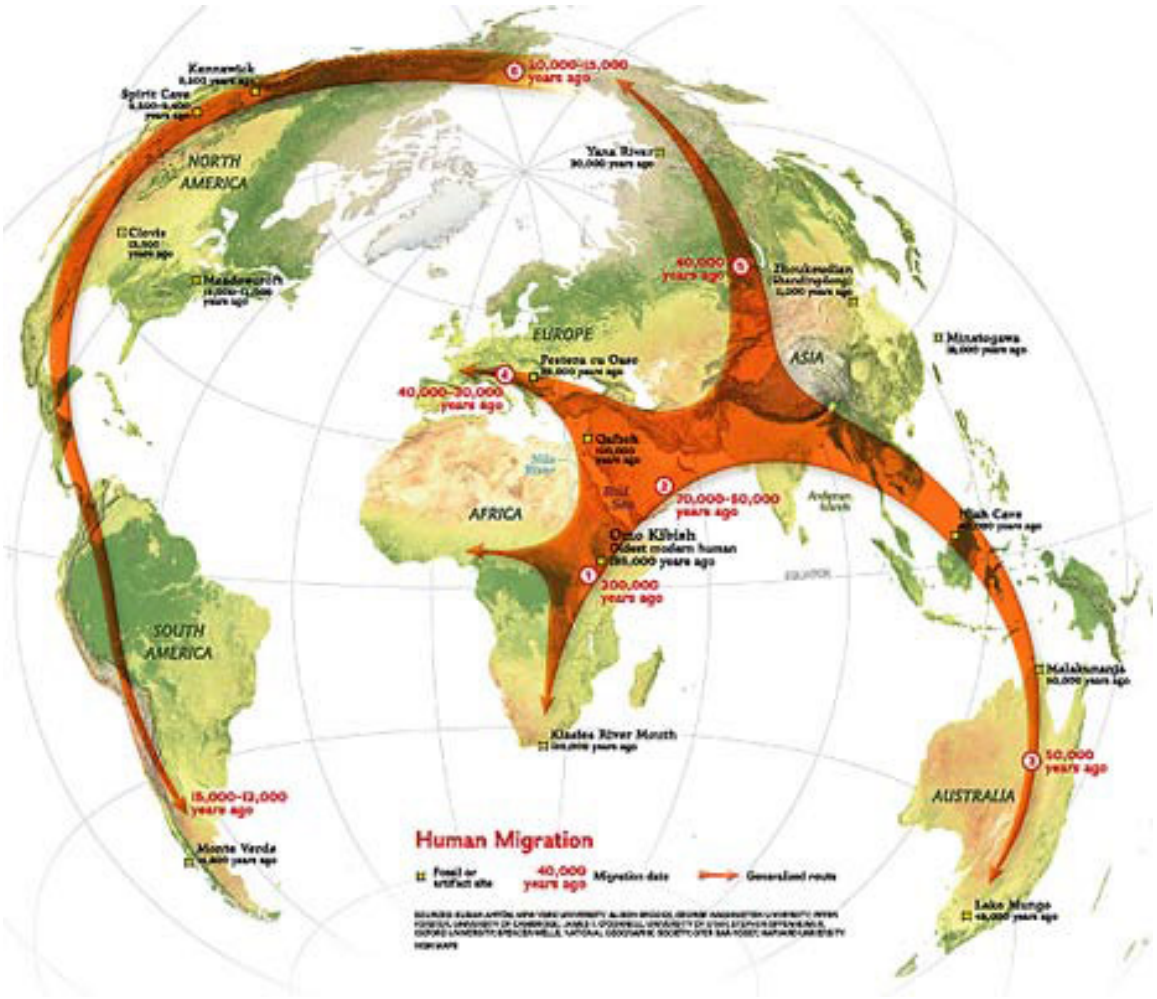
$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix} \quad T = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\ C & 0 & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow \end{pmatrix}$$

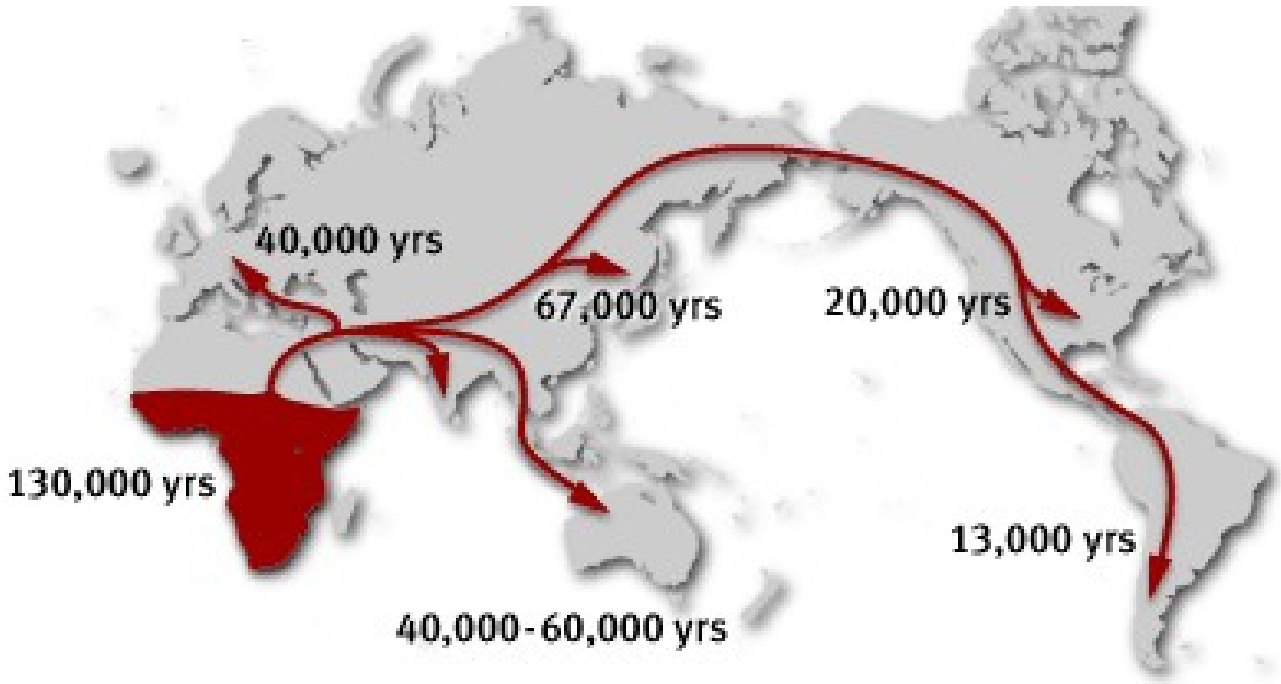
Sequence 1 = A--CACACTA
Sequence 2 = AGCACAC--A

ClustalW


http://www.ebi.ac.uk/Tools/msa/clustalw2/
Nature Publishing Group : scie...
clustalw - Google 搜尋
ClustalW2 < Multiple Seque...
File Edit View Favorites Tools Help
Cookies on EMBL-EBI website
This website uses cookies to store a small amount of information on your computer, as part of the functioning of the site. Cookies used for the operation of the site have already been set.
To find out more about the cookies we use and how to delete them, see our Cookie and Privacy statements.
Dismiss this notice
EMBL-EBI
Services
Research
Training
About us
ClustalW2
Input form
Web services
Help & Documentation
Share
Feedback
Tools > Multiple Sequence Alignment > ClustalW2
Multiple Sequence Alignment
ClustalW2 is a general purpose DNA or protein multiple sequence alignment program for three or more sequences. For the alignment of two sequences please instead use our pairwise sequence alignment tools.
Note: ClustalW2 is no longer being maintained. Please consider using the new version instead: Clustal Omega
STEP 1 - Enter your input sequences
Enter or paste a set of Protein sequences in any supported format:
Or, upload a file:
Browse...
STEP 2 - Set your Pairwise Alignment Options
Alignment Type: Slow Fast
The default settings will fulfill the needs of most users and, for that reason, are not visible.
More options...
(Click here, if you want to view or change the default settings.)
STEP 3 - Set your Multiple Sequence Alignment Options
The default settings will fulfill the needs of most users and, for that reason, are not visible.
More options...
(Click here, if you want to view or change the default settings.)
40%
2.2K/s
1K/s

Example: out of Africa





- Evidence
  - Fossile
  - DNA sequence analysis

Homework

- **Insulin** is a peptide hormone, produced by beta cells of the pancreas, and is central to regulating carbohydrate and fat metabo lism in the body.

>gi|386828|gb|AAA59172.1| insulin [Homo sapiens]
MAWMRLLPLLALLALWGPDPAAAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLAL
EGSLQKRGIVEQCCTSICSLYQLENYCN

>gi|82749718|gb|ABB89743.1| preproinsulin 1 [Rattus losea]
MALWMRFLPLLALLVVWEPKPAQAFVKQHLCGPHLVEALYLVCGERGFFYTPKSRREVEDPQVPQLELGGSPEAGDLQTLAL
EVARQKRGIVDQCCTSICSLYQLENYCN

>gi|402892452|ref|XP_003909428.1| PREDICTED: insulin isoform 4 [Papio anubis]
MALWMRLLPLLALLALWGPDSVPAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDPQVGQVELGGGPGAGSLQPLAL
EGSLQKRGIVEQCCTSICSLYQLENYCN

>gi|344307503|ref|XP_003422420.1| PREDICTED: insulin-like isoform 1 [Loxodonta africana]
MALWTRLLPLLALLAVGAPPPARAFVNQHLCGSHLVEALYLVCGERGFFYTPKTRREVEDTQVGEVELGTGLQPFPAEAPKQK
RGIVEQCCTGVCSLYQLENYCN

Identity score of insulin

Demo

- Write your own program, and demo in class
- A .fasta file will be provided to test your program
- Your program must be able to read fasta files

Score	Criteria
100	1, wrote a alignment program 2, demon in class 3, correctly calculate the identity score
80	1, wrote a alignment program 2, demon in class
0	1, no programming or copy other’s program

Extra

- Write a similar program to estimate the si milarity of two articles.
- Estimate the similarity of your program wit h others' program.