Dynamic programming

Torgeir R. Hvidsten

This lecture

- ➢Sequence alignment
  - – Edit distance
  - – Global alignment and scoring
  - – Local alignment
  - – Gap penalties
  - – Multiple alignments
- ➢Gene prediction
- ➢Dynamic programming

DNA sequence comparison: First success story

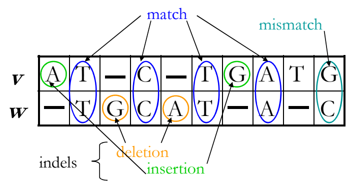
- ➢In 1984 Russell Doolittle and colleagues found similarities between a cancer-causing gene and a normal growth factor (PDGF) gene using a database search
- ➢Finding sequence similarities with genes of known function is a common approach to infer the function of a newly sequenced gene

Aligning DNA sequences

$\boldsymbol{v}$ = ATCTGATG $n = 8$

$\boldsymbol{w}$ = TGCATAC $m = 7$

4	matches
1	mismatches
2	insertions
2	deletions



Edit distance

Hamming distance (I)

Given two DNA sequences ***v*** and ***w***:

***v* :** ATATATAT

***w*:** TATATATA

The Hamming distance $d_{H(\boldsymbol{v},\,\boldsymbol{w})}$ = 8 is large, but the sequences are very similar

Hamming distance (II)

By shifting one sequence over one position

***v* :** ATATATAT-

***w*:** - TATATATA

the distance is $d_{H(v,\,w)}$ = 2

Hamming distance neglects insertions and deletions in DNA

Edit distance

Levenshtein (1966) introduced edit distance between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

$d(\boldsymbol{v},\boldsymbol{w})$ = minimum number of elementary operations to transform ***v*** into ***w***

Hamming distance vs Edit distance

Hamming distance always compares the ith letter of $\boldsymbol{v}$ with the ith letter of $\boldsymbol{w}$

```
V  = ATATATAT
      | | | | | | | |
W = TATATATA
```

Hamming distance:
$d(\boldsymbol{v}, \boldsymbol{w})=8$

Edit distance may compare the ith letter of $\boldsymbol{v}$ with the jth letter of $\boldsymbol{w}$

```
V  = - ATATATAT
         | | | | | | |
W = TATATATA-
```

Edit distance:
$d(\boldsymbol{v}, \boldsymbol{w})=2$

(one insertion and one deletion)

How to find which $\boldsymbol{j}$ goes with which $\boldsymbol{i}$?

Longest common subsequence (LCS) – alignment without mismatches

- Given two sequences

  $\boldsymbol{v} = v_1\, v_2 \ldots v_m$

  $\boldsymbol{w} = w_1\, w_2 \ldots w_n$

- The LCS of $\boldsymbol{v}$ and $\boldsymbol{w}$ is the longest sequence of positions

  $\boldsymbol{v}$: $1 \le i_1 < i_2 < \ldots < i_k \le m$

  $\boldsymbol{w}$: $1 \le j_1 < j_2 < \ldots < j_k \le n$

  such that

  $v_{i_t} = w_{j_t}$ for $1 \le t \le k$

LCS: Example

i coords:	0	0	1	2	3	4	5	5	6	6	7
elements of $\boldsymbol{v}$		–	T	G	C	A	T	–	A	–	C
elements of $\boldsymbol{w}$		A	T	–	C	–	T	G	A	T	C
j coords:	0	1	2	2	3	3	4	5	6	7	8

Matches shown in red

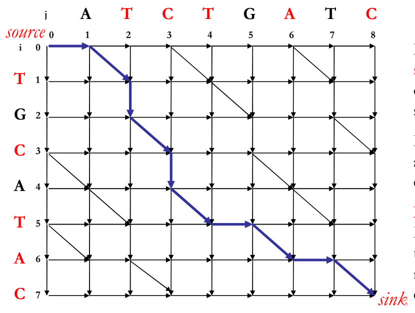
positions in $\boldsymbol{v}$: $1 < 3 < 5 < 6 < 7$

positions in $\boldsymbol{w}$: $2 < 3 < 4 < 6 < 8$

TCTAC is a common subsequence of $\boldsymbol{v}$ and $\boldsymbol{w}$

Every common subsequence is a path in 2-D grid

Edit graph for the LCS problem

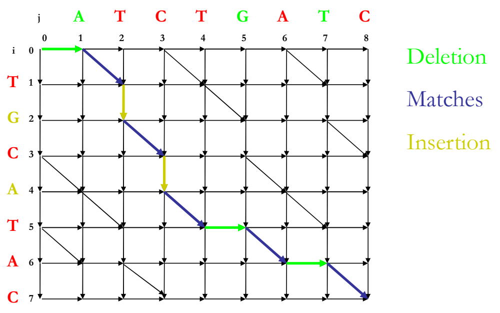


Every path from source to sink is a common subsequence (CS)

Every diagonal edge adds an extra element to the CS

LCS Problem:
Find the path with the maximum number of diagonal edges

Edit graph for the LCS problem



LCS: Dynamic programming

- <u>Goal</u>: Find the LCS of two strings
- <u>Input</u>: A weighted graph *G*, where diagonals are +1 edges, with two distinct vertices, one labeled "*source*" one labeled "*sink*"
- <u>Output</u>: A longest path in *G* from "*source*" to "*sink*"

Computing LCS (I)

Let $\mathbf{v}_i$ = prefix of $\mathbf{v}$ of length i: $v_1 \dots v_i$

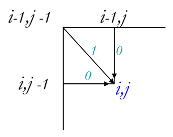
and $\mathbf{w}_j$ = prefix of $\mathbf{w}$ of length j: $w_1 \dots w_j$

The length of LCS($\mathbf{v}_i$,$\mathbf{w}_j$) is computed by:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$

Computing LCS (II)

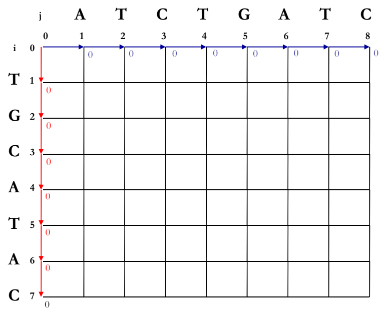
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + 0 & \text{Insertion} \\ s_{i,j-1} + 0 & \text{Deletion} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j & \text{Match} \end{cases}$$



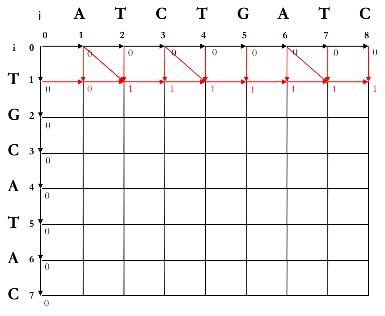
LCS algorithm

```
LCS($\mathbf{v}$, $n$, $\mathbf{w}$, $m$)
1   for $i \leftarrow 1$ to $n$
2       $s_{i,0} \leftarrow 0$
3   for $j \leftarrow 1$ to $m$
4       $s_{0,j} \leftarrow 0$
5   for $i \leftarrow 1$ to $n$
6       for $j \leftarrow 1$ to $m$
8           $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$
10  return $s_{n,m}$
```

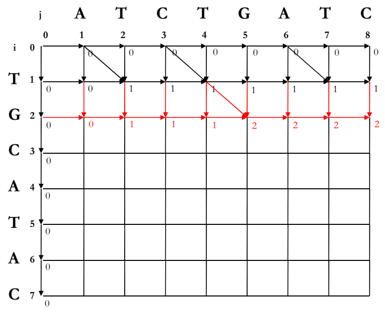
Example: initiation



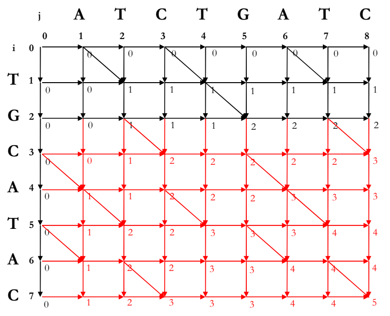
Example: For *i* = *1*, *j* = *1*… *m*



Example: For *i* = *2*, *j* = *1*… *m*



Example: For *i* = *3 … n*, *j* = *1… m*



LCS Runtime

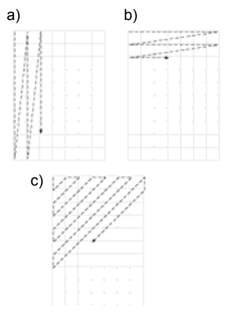
- It takes O(nm) time to fill in the $n \times m$ dynamic programming matrix
- The pseudocode consists of a nested "**for**" loop inside of another "**for**" loop to set up a $n \times m$ matrix

What's so great about dynamic programming?

- A naive exhaustive search would have the running time O($3^{f(n,m)}$)
- An exhaustive search would recompute the same subpaths several times
- Dynamic programming takes advantage of the rich computational structure in the search space, and reuse already computed subpaths

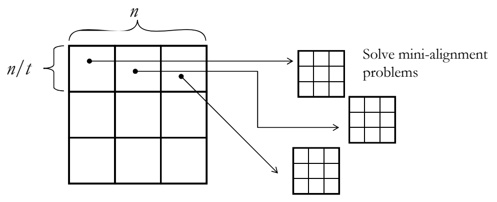
Traversing the edit graph

3 different strategies:
- a) Column by column
- b) Row by row
- c) Along diagonals



Align sequences in subquadratic time

Divide and conquer techniques can be used to solve the LCS problem in O(n^2/log n) time



Global alignment and scoring

From LCS to alignment

- The Longest Common Subsequence (LCS) problem is the simplest form of sequence alignment
- We scored *1* for matches and *0* for indels
- We did not allow mismatches, only insertions and deletions

Simple scoring

- Mismatches are penalized by $-\mu$,
- Indels are penalized by $-\sigma$,
- Matches are rewarded with $+1$

- The resulting score is:

  $\#matches - \mu \cdot \#mismatches - \sigma \cdot \#indels$

The global alignment problem

Goal: Find the best alignment between two strings under a given scoring schema

Input : Strings $\boldsymbol{v}$ and $\boldsymbol{w}$ and a scoring schema
Output : Alignment of maximum score

$$s_{i,j} = max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} - \mu \text{ if } v_i \neq w_j \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$

Scoring matrices

- To generalize scoring, consider a $(4+1) \times (4+1)$ scoring matrix δ
- In the case of an amino acid sequence alignment, the scoring matrix would be $(20+1) \times (20+1)$
- The addition of 1 is to include the score for comparison of a gap character "-" (indels)

$$s_{i,j} = max \begin{cases} s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

Making a scoring matrix

- Scoring matrices are created based on biological evidence
- Alignments can be thought of as two sequences that differ due to mutations
- Some of these mutations have little effect on the protein's function, therefore some penalties, $\delta(i, j)$, will be less harsh than others
- $\delta(i, j) \approx$ how often do amino acid i substitutes amino acid j in alignments of related proteins

Scoring matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

- Notice that although **R** and **K** are different amino acids, they have a positive score
- Why? They are both positively charged amino acids and will not greatly change the function of protein

Scoring matrices

- Amino acid substitution matrices
  - PAM
  - BLOSUM
- DNA substitution matrices
  - DNA is less conserved than protein sequences
  - Less effective to compare coding regions at nucleotide level

PAM

- **P**oint **A**ccepted **M**utation
- 1 PAM = PAM_1 = 1% average change of all amino acid positions
- After 100 PAMs of evolution, not every residue will have changed
  - some residues may have mutated several times
  - some residues may have returned to their original state
  - some residues may not changed at all

PAM_X

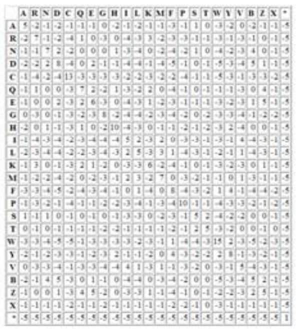
- $PAM_x = PAM_1^x$
  - $PAM_{250} = PAM_1^{250}$
- PAM_{250} is a widely used scoring matrix:

```
        Ala  Arg  Asn  Asp  Cys  Gln  Glu  Gly  His  Ile  Leu  Lys ...
         A    R    N    D    C    Q    E    G    H    I    L    K  ...
Ala A   13    6    9    9    5    8    9   12    6    8    6    7  ...
Arg R    3   17    4    3    2    5    3    2    6    3    2    9
Asn N    4    4    6    7    2    5    6    4    6    3    2    5
Asp D    5    4    8   11    1    7   10    5    6    3    2    5
Cys C    2    1    1    1   52    1    1    2    2    2    1    1
Gln Q    3    5    5    6    1   10    7    3    7    2    3    5
...
Trp W    0    2    0    0    0    0    0    0    1    0    1    0
Tyr Y    1    1    2    1    3    1    1    1    3    2    2    1
Val V    7    4    4    4    4    4    4    4    5    4   15   10
```

BLOSUM

- **Blo**cks **Su**bstitution **M**atrix
- Scores derived by observing the frequencies of substitutions in blocks of local alignments in related proteins
- Matrix name indicates evolutionary distance
  - BLOSUM62 was created using sequences sharing no more than 62% sequence identity

BLOSUM50



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0	-2	-1	-1	-5
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3	-1	0	-1	-5
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3	4	0	-1	-5
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4	5	1	-1	-5
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1	-3	-3	-2	-5
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3	0	4	-1	-5
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3	1	5	-1	-5
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4	-1	-2	-2	-5
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4	0	0	-1	-5
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4	-4	-3	-1	-5
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1	-4	-3	-1	-5
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3	0	1	-1	-5
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1	-3	-1	-1	-5
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1	-4	-4	-2	-5
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3	-2	-1	-2	-5
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2	0	0	-1	-5
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0	0	-1	0	-5
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-3	15	2	-3	-5	-2	-3	-5
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1	-3	-2	-1	-5
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5	-4	-3	-1	-5
B	-2	-1	4	5	-3	0	1	-1	0	-4	-4	0	-3	-4	-2	0	0	-5	-3	-4	5	2	-1	-5
Z	-1	0	0	1	-3	4	5	-2	0	-3	-3	1	-1	-4	-1	0	-1	-2	-2	-3	2	5	-1	-5
X	-1	-1	-1	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-1	0	-3	-1	-1	-1	-1	-1	-5
*****	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	1

Local alignment

Local vs. global alignment (I)

- The Global alignment problem : find the longest path between vertices (0,0) and (n,m) in the edit graph
- The Local alignment problem tries to find the longest path between arbitrary vertices (i, j) and (i', j') in the edit graph
- In the edit graph with negative scores, local alignment may score higher than global alignment

Local vs. global alignment (II)

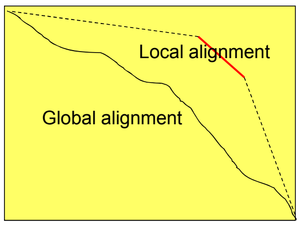
- Global Alignment

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

- Local Alignment—better alignment to find conserved segment

```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

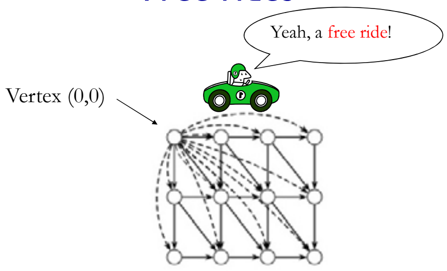
Local vs. global alignment (III)



The local alignment problem

- Goal: Find the best local alignment between two strings
- Input : Strings ***v***, ***w*** and scoring matrix δ
- Output : Alignment of substrings of ***v*** and ***w*** whose alignment score is maximum among all possible alignment of all possible substrings

Free rides



The dashed edges represent the free rides from (0,0) to every other node.

The local alignment recurrence

- The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$

- The 0 is the only difference from the recurrence of the global alignment problem

Gap penalties

Scoring indels: Naive approach

- A fixed penalty σ is given to every indel:
  - $-\sigma$ for 1 indel,
  - -2σ for 2 consecutive indels,
  - -3σ for 3 consecutive indels, etc
- Can be too severe penalty for a series of *100* consecutive indels

Gap penalties

In nature, a series of k indels often come as a single event rather than a series of k single nucleotide events:

```
ATA– –GC        ATAG– GC
ATATTGC         AT– GTGC
```

This is more likely

Normal scoring would give the same score for both alignments

This is less likely

Accounting for gaps

Score for a gap of length x is:

$-(\rho + \sigma x)$

where $\rho > 0$ is the penalty for introducing a gap:

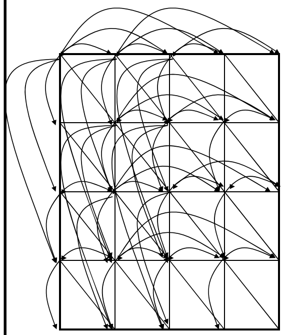
gap opening penalty

ρ will be large relative to σ:

gap extension penalty

because you do not want to add too much of a penalty for extending the gap

Adding "penalty" edges to the edit graph

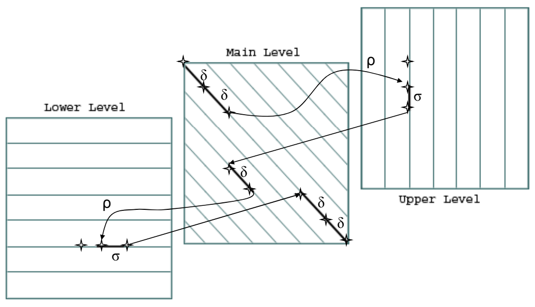


- To reflect gap penalties we have to add "long" horizontal and vertical edges to the edit graph of weight: - ϱ - $x\cdot\sigma$
- This increases the running time of the alignment algorithm by a factor of n (where n is the number of vertices)
- So the complexity increases from $O(n^2)$ to $O(n^3)$

Gap penalties and 3 layer edit graphs

- The three recurrences for the scoring algorithm creates a 3-layered graph
- The upper level creates/extends gaps in the sequence $\boldsymbol{w}$
- The lower level creates/extends gaps in sequence $\boldsymbol{v}$
- The main level extends matches and mismatches

3 layer edit grap



Gap penalty recurrences

$$\overset{\downarrow}{s}_{i,j} = \max \begin{cases} \overset{\downarrow}{s}_{i-1,j} - \sigma & \text{Continue gap in } \boldsymbol{w} \text{ (insertion): upper level} \\ s_{i-1,j} - (\varrho+\sigma) & \text{Start gap in } \boldsymbol{w} \text{ (insertion): from main level} \end{cases}$$

$$\vec{s}_{i,j} = \max \begin{cases} \vec{s}_{i,j-1} - \sigma & \text{Continue gap in } \boldsymbol{v} \text{ (deletion): lower level} \\ s_{i,j-1} - (\varrho+\sigma) & \text{Start gap in } \boldsymbol{v} \text{ (deletion): from main level} \end{cases}$$

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) & \text{Match or mismatch: main level} \\ \overset{\downarrow}{s}_{i,j} & \text{End insertion: from upper level} \\ \vec{s}_{i,j} & \text{End deletion: from lower level} \end{cases}$$

BLAST (I)

- Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences
- The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches

BLAST (II)

- First stage: Identify exact matches of length W (default W=3) between the query and the sequences in the database
- Second stage: Extend the match in both directions in an attempt to boost the alignment score (insertions and deletions are not considered)
- Third stage: If a high-scoring ungapped alignment is found: Perform a gapped local alignment using dynamic programming

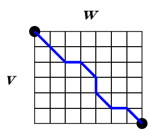
Multiple alignments

Multiple alignment

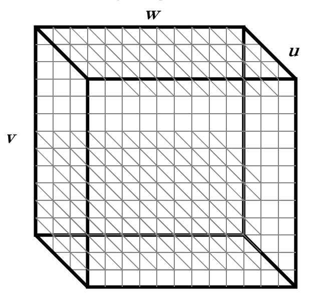
- A faint similarity between two sequences becomes significant if present in many
- Multiple alignments can reveal subtle similarities that pairwise alignments do not reveal

```
A  T  –  G  C  G  –
A  –  C  G  T  –  A
A  T  C  A  C  –  A
```

2D vs 3D edit graph

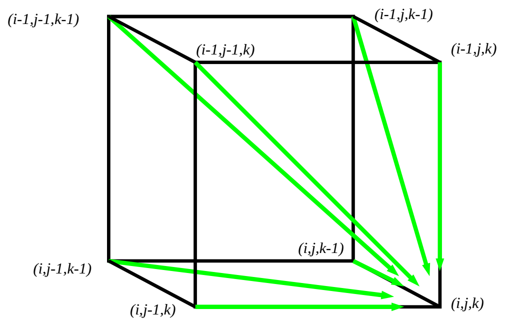


2-D edit graph



3-D edit graph

Architecture of 3D edit graph



Multiple alignment of three sequences: Dynamic programming

$$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, \_) \\ s_{i-1,j,k-1} + \delta(v_i, \_, u_k) \\ s_{i,j-1,k-1} + \delta(\_, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, \_, \_) \\ s_{i,j-1,k} + \delta(\_, w_j, \_) \\ s_{i,j,k-1} + \delta(\_, \_, u_k) \end{cases}$$

$\delta(x, y, z)$ is an entry in the 3D scoring matrix

Multiple alignment: Running time

- For three sequences of length n, the run time is $O(n^3)$
- For k sequences, build a k-dimensional edit graph, with run time $O(n^k)$
- Conclusion: dynamic programming approach for alignment between two sequences is easily extended to k sequences, but it is impractical due to exponential running time

Multiple alignment induces pairwise alignments

Every multiple alignment:

```
x: AC-GCGG-C
y: AC-GC-GAG
z: GCCGC-GAG
```

induces pairwise alignment:

```
x: ACGCGG-C   x: AC-GCGG-C   y: AC-GCGAG
y: ACGC-GAC   z: GCCGC-GAG   z: GCCGCGAG
```

Reverse problem: Constructing multiple alignment from pairwise alignments

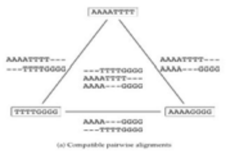
Given three pairwise alignments:

```
x: ACGCTGG-C   x: AC-GCTGG-C   y: AC-GC-GAG
y: ACGC--GAC   z: GCCGCA-GAG   z: GCCGCAGAG
```

can we construct the multiple alignment that induces them?

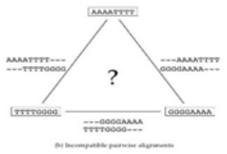
Combining optimal pairwise alignments into multiple alignment

Can combine pairwise alignments into multiple alignment



(a) Compatible pairwise alignments

Can not combine pairwise alignments into multiple alignment



(b) Incompatible pairwise alignments

Profile representation of multiple alignment

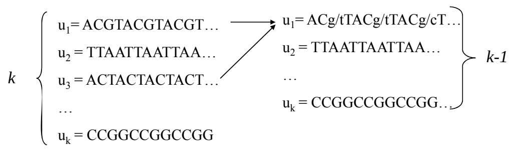
```
      -   A   G   G   C   T   A   T   C   A   C   C   T   G
      T   A   G   -   C   T   A   C   C   A   -   -   -   G
      C   A   G   -   C   T   A   C   C   A   -   -   -   G
      C   A   G   -   C   T   A   T   C   A   C   -   G   G
      C   A   G   -   C   T   A   T   C   G   C   -   G   G

A         1                   1          .8
C    .6               1          .4   1      .6  .2
G             1  .2                      .2          .4   1
T    .2                   1      .6                  .2
-    .2          .8                          .4  .8  .4
```

PSSM: Position Specific Scoring Matrix

- ➢ In the past we were aligning a sequence against a sequence
- ➢ With profiles we can align a sequence against a profile and even a profile against a profile

Multiple alignment: Greedy approach

- Choose most similar pair of strings and combine into a profile, thereby reducing the alignment of *k* sequences to an alignment of of *k-1* sequences/profiles. Repeat!
- This is a heuristic greedy method



CLUSTALW (I)

1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule "once a gap always a gap“.

CLUSTALW (II)

1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule "once a gap always a gap“.

Details:

1.1. clustalw uses a pairwise alignment to compute pairwise alignments.

1.2. Using the alignments from 1.1 it computes a distance.

1.2.1. The distance is calculated by looking at the non-gapped positions and count the number of mistmatches between the two sequences. Then divide this value by the number of non-gapped pairs to calculate the distance. Once all distances for all pairs are calculated they go into a matrix.

CLUSTALW (III)

1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule "once a gap always a gap“.

Details:

2. Using the matrix from 1.2.1. and Neighbor-Joining*, Clustalw constructs the similarity tree. The root is placed in the middle of the longest chain of consecutive edges.

* Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol., 4: 406-425

CLUSTALW (IV)

1. Determine all pairwise alignments between sequences and the degree of similarity between them.
2. Construct a similarity tree.
3. Combine the alignments from 1 in the order specified in 2 using the rule "once a gap always a gap".

Details:

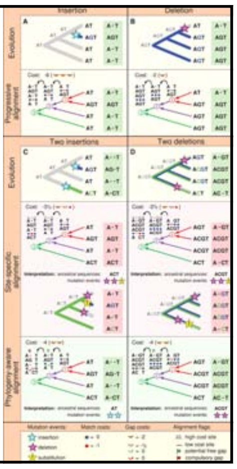
2. Combine the alignments, starting from the closest related groups (going from the tips of the tree towards the root).

Phylogeny-aware gap placement (I)

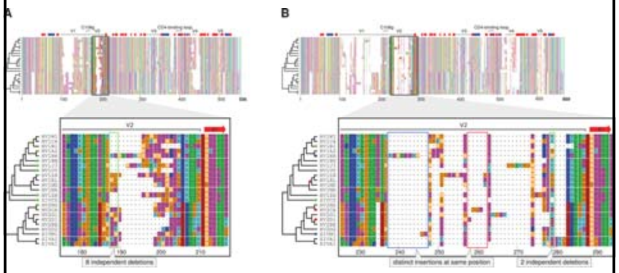
A. Löytynoja and N. Goldman. Phylogeny-Aware Gap Placement Prevents Errors in Sequence Alignment and Evolutionary Analysis. *Science* 320: 1632-35, 2008.

Conclusion:

"*The resulting alignments may be fragmented by many gaps and may not be as visually beautiful as the traditional alignments, but if they represent correct homology, we have to get used to them.*"



Phylogeny-aware gap placement (II)



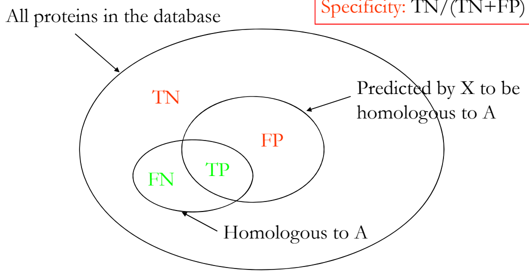
PSI-BLAST

- Position-Specific Iterative (PSI) BLAST detect weak relationships between the query and sequences in the database (higher sensitivity than BLAST)
- PSI-BLAST first constructs a multiple alignment from the highest scoring hits in a initial BLAST search and generate a profile from this alignment i.e. PSSM
- The profile is used to iteratively perform additional BLAST searches (called iterations) and the results of each iteration is used to refine the profile
- The iteration stops when no new matches with a satisfactory score are obtained

Method power

You want to find homologous proteins to a specific protein A using some computational method X:

Sensitivity: TP/(TP+FN)
Specificity: TN/(TN+FP)



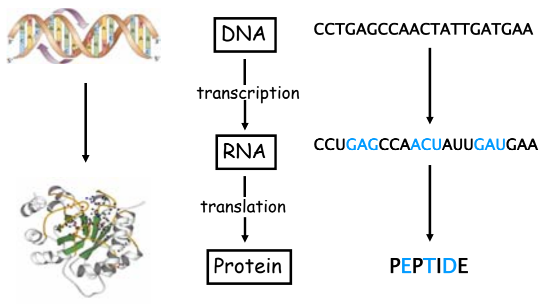
All proteins in the database
TN
FP
TP
FN
Predicted by X to be homologous to A
Homologous to A

Gene prediction

Gene prediction problem

- ➢Gene: A sequence of nucleotides coding for protein
- ➢Gene prediction problem: Determine the beginning and end positions of genes in a genome

Central dogma

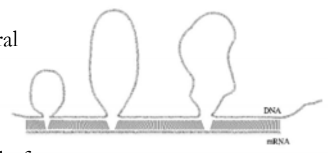


Translating nucleotides into amino acids

- Codon: 3 consecutive nucleotides
- $4^3 = 64$ possible codons
- Genetic code is degenerative and redundant
- Includes start and stop codons
- An amino acid may be coded by more than one codon

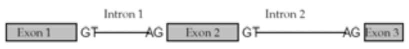
Discovery of split genes

- In 1977, Phillip Sharp and Richard Roberts experimented with mRNA of hexon, a viral protein
- mRNA-DNA hybrids formed three curious loop structures instead of contiguous duplex segments

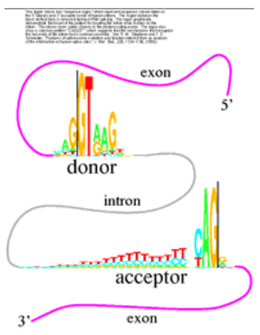


Exons and introns (I)

- In eukaryotes, the gene is a combination of coding segments (exons) that are interrupted by non-coding segments (introns)
- This makes computational gene prediction in eukaryotes even more difficult
- Prokaryotes don't have introns - genes in prokaryotes are continuous



Exons and introns (II)



Two approaches to gene prediction

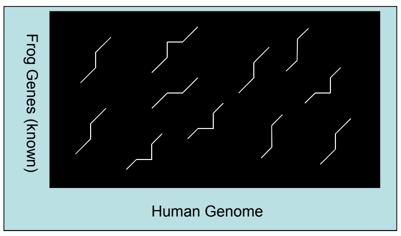
- Statistical: based on detecting subtle statistical variations between coding (exons) and non-coding regions
- Similarity-based: many human genes are similar to genes in mice, chicken, or even bacteria. Therefore, already known mouse, chicken, and bacterial genes may help to find human genes

Gene prediction: Similarity-based approach

Similarity-based approach to gene prediction

- Genes in different organisms are similar
- The similarity-based approach uses known genes in one genome to predict genes in another genome
- Problem: Given a known gene and an unannotated genome sequence, find a set of substrings in the genomic sequence whose concatenation best fits the known gene

Local alignment gives candidate exons



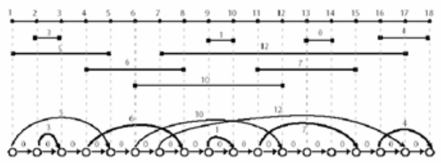
Exon chaining problem

- ➢ Exon chaining problem: Given a set of weighted candidate exons, find a maximum set of non-overlapping exons
- ➢ Candidate exon (*l*, *r*, *w*) : left position, right position, weight (defined as the score of the local alignment)

- ➢ Input: a set of weighted intervals (putative exons):
- ➢ Output: A maximum chain of intervals from this set

Exon chaining problem: Graph representation

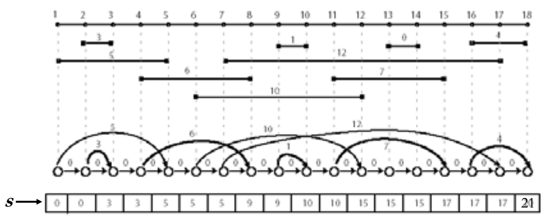
This problem can be solved with dynamic programming in O(*n*) time



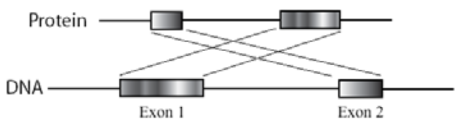
Exon chaining algorithm

```
ExonChaining ($G$, $n$)
1  for $i \leftarrow$ to $2n$
2      $s_i \leftarrow 0$
3  for $i \leftarrow 1$ to $2n$
4      if vertex $v_i$ in $G$ corresponds to the right end of an interval $I$
5          $j \leftarrow$ Index of vertex for left end of the interval $I$
6          $w \leftarrow$ Weight of the interval $I$
7          $s_i \leftarrow \max \{s_j + w, s_{i-1}\}$
8      else
9          $s_i \leftarrow s_{i-1}$
10 return $s_{2n}$
```

Example



Exon chaining: Deficiencies



- The optimal chain of intervals may not correspond to any valid alignment
- Solution: Spliced alignment (see book section 6.14)