

---

# CS481/CS583: Bioinformatics Algorithms

---

Can Alkan

EA509

`calkan@cs.bilkent.edu.tr`

<http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/>

---

# MORE ON PAIRWISE ALIGNMENT

---

---

# From LCS to Alignment: Change up the Scoring

- The Longest Common Subsequence (LCS) problem—the simplest form of sequence alignment – allows only insertions and deletions (no mismatches).
  - In the LCS Problem, we scored 1 for matches and 0 for indels
  - Consider penalizing indels and mismatches with negative scores
  - Simplest *scoring schema*:
    - +1 : match premium**
    - $-\mu$  : mismatch penalty**
    - $-\sigma$  : indel penalty**
-

# Simple Scoring

- When mismatches are penalized by  $-\mu$ , indels are penalized by  $-\sigma$ , and matches are rewarded with  $+1$ , the resulting score is:

$$\#matches - \mu(\#mismatches) - \sigma(\#indels)$$

# The Global Alignment Problem

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

Input : Strings **v** and **w** and a scoring schema

Output : Alignment of maximum score

$$\begin{array}{l} \uparrow \rightarrow = -\sigma \\ \swarrow \left\{ \begin{array}{l} = 1 \text{ if match (or +score)} \\ = -\mu \text{ if mismatch} \end{array} \right. \end{array}$$

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + 1 \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu \text{ if } v_i \neq w_j \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \end{array} \right.$$

**$\mu$  : mismatch  
penalty  
 $\sigma$  : indel penalty**

# Example

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
A								
C								
G								
C								
G								
A								

# Example: initialize

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2							
C	-4							
G	-6							
C	-8							
G	-10							
A	-12							

# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)						
C	-4							
G	-6							
C	-8							
G	-10							
A	-12							

D: diagonal L: left U: up



# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4							
G	-6							
C	-8							
G	-10							
A	-12							

D: diagonal L: left U: up

# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6							
C	-8							
G	-10							
A	-12							

D: diagonal L: left U: up

# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8							
G	-10							
A	-12							

D: diagonal L: left U: up

# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: backtrack

S1 = A

S2 = A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: backtrack

S1 = G A

S2 = G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: backtrack

S1 = C G A  
S2 = T G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: backtrack

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

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal   L: left   U: up



# Example: backtrack

S1 =            - G C G A

S2 =            A G T G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal   L: left   U: up

# Example: backtrack

S1 = C - G C G A  
S2 = G A G T G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: backtrack

S1 = A C - G C G A

S2 = C G A G T G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Example: finished

S1 = A C - G C G A

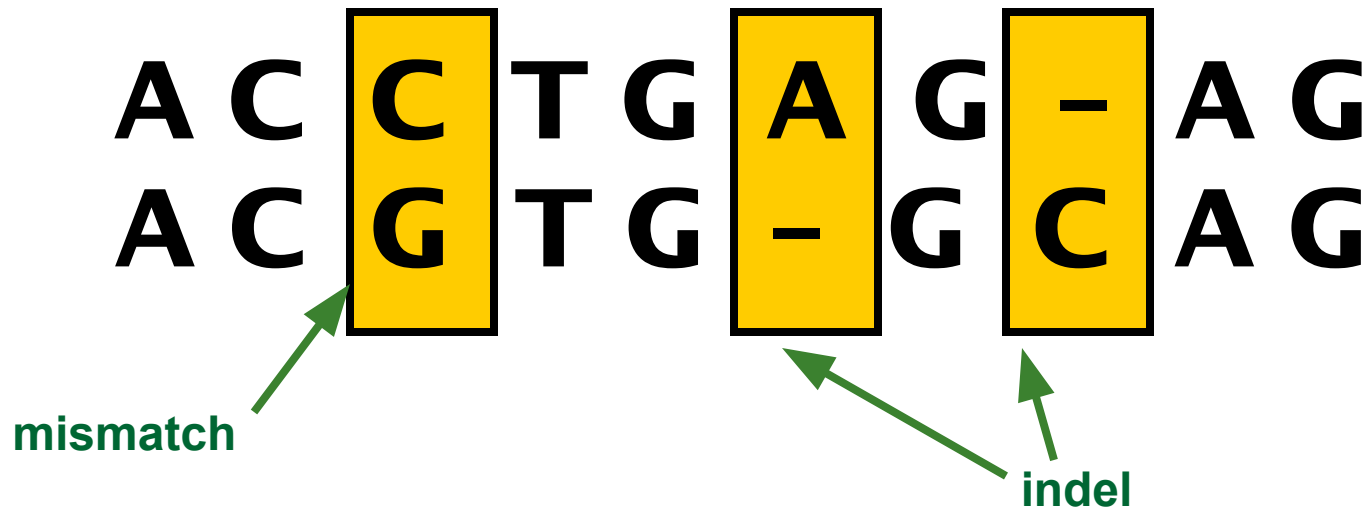
S2 = C G A G T G A

		C	G	A	G	T	G	A
	0	-2	-4	-6	-8	-10	-12	-14
A	-2	-1 (d)	-3 (l)	-3 (d)	-5 (l)	-7 (l)	-9 (l)	-11 (l)
C	-4	-1(d)	-2 (d)	-4 (l)	-4 (d)	-6 (d)	-8 (d)	-10 (d)
G	-6	-3 (u)	0 (d)	-2 (l)	-3 (d)	-5 (l)	-5 (d)	-7 (l)
C	-8	-5 (u)	-2 (u)	-1 (d)	-3 (d)	-4 (d)	-6 (l)	-6 (d)
G	-10	-7 (u)	-4 (u)	-3 (d)	0 (d)	-2 (l)	-3 (d)	-5 (l)
A	-12	-9 (u)	-6 (u)	-3 (d)	-2 (u)	-1 (d)	-3 (d)	-2 (d)

D: diagonal L: left U: up

# Percent Sequence Identity

- The extent to which two nucleotide or amino acid sequences are invariant



Alignment length = 10

Matches = 7

70% identical

# Scoring Matrices

To generalize scoring, consider a  $(4+1) \times (4+1)$  **scoring matrix**  $\delta$ .

In the case of an amino acid sequence alignment, the scoring matrix would be a  $(20+1) \times (20+1)$  size. The addition of 1 is to include the score for comparison of a gap character “-”.

This will simplify the algorithm as follows:

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{array} \right.$$

# 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,  $\delta(v_i, w_j)$ , will be less harsh than others.

# Scoring Matrix: Example

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

**AKRANR**

**KAAANK**

$$-1 + (-1) + (-2) + 5 + 7 + 3 = 11$$

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



# Conservation

- Amino acid changes that tend to preserve the physico-chemical properties of the original residue
  - Polar to polar
    - aspartate □ glutamate
  - Nonpolar to nonpolar
    - alanine □ valine
  - Similarly behaving residues
    - leucine to isoleucine

---

# 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 (Dayhoff et al.)
- PAM<sub>250</sub> 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

- **B**locks **S**ubstitution **M**atrix
  - Scores derived from *observations* of 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% identity
-

# The Blossum50 Scoring Matrix

[illegible]

# Scoring Indels: Naive Approach

- A fixed penalty  $\sigma$  is given to every indel:
  - $-\sigma$  for 1 indel,
  - $-2\sigma$  for 2 consecutive indels
  - $-3\sigma$  for 3 consecutive indels, etc.

Can be too severe penalty for a series of 100 consecutive indels

# Affine 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  
ATATTGC

**This is more likely.**

Normal scoring  
would give the same  
score for both  
alignments

ATAG\_GC  
AT\_GTGC

**This is less likely.**

# Accounting for Gaps

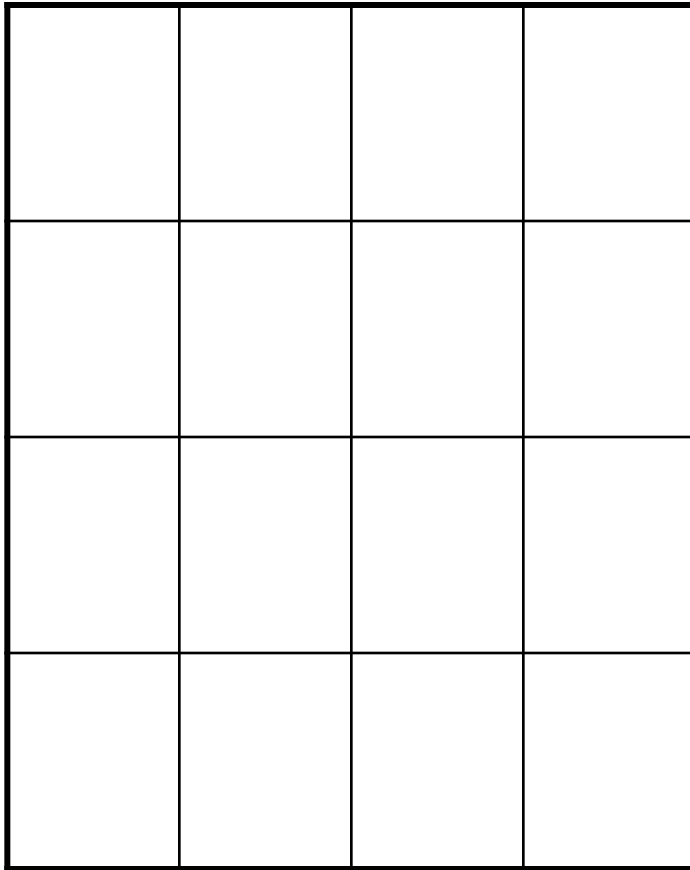
- *Gaps*- contiguous sequence of spaces in one of the rows
- Score for a gap of length  $x$  is:  
$$-(\rho + \sigma x)$$
where  $\rho > 0$  is the penalty for introducing a gap:  
**gap opening penalty**  
 $\rho$  will be large relative to  $\sigma$ :  
**gap extension penalty**  
because you do not want to add too much of a penalty for extending the gap.



# Affine Gap Penalties

- Gap penalties:
  - $-\rho - \sigma$  when there is 1 indel
  - $-\rho - 2\sigma$  when there are 2 indels
  - $-\rho - 3\sigma$  when there are 3 indels, etc.
  - $-\rho - x \cdot \sigma$  (-gap opening -  $x$  gap extensions)
- Somehow reduced penalties (as compared to naïve scoring) are given to runs of horizontal and vertical edges

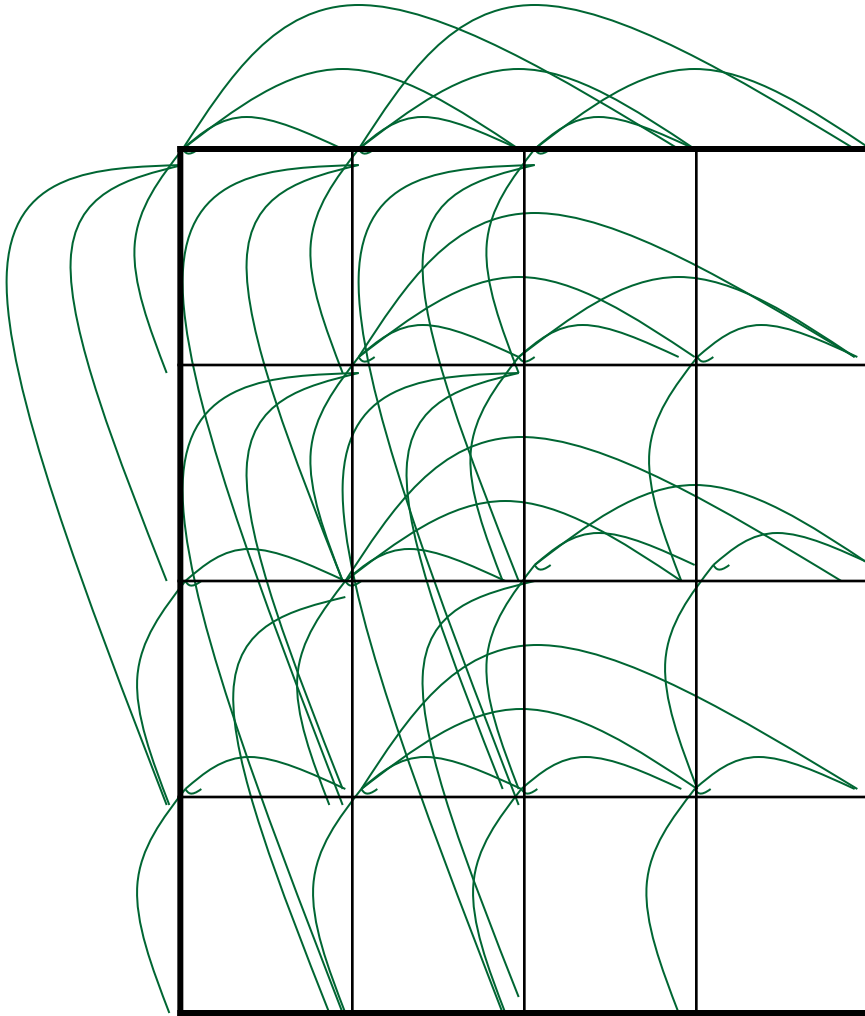
# Affine Gap Penalties and Edit Graph



**To reflect affine gap penalties we have to add “long” horizontal and vertical edges to the edit graph. Each such edge of length  $x$  should have weight**

$$-\rho - x * \sigma$$

# Adding “Affine Penalty” Edges to the Edit Graph



Adding them to the graph 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  $O(n^3)$

We can still achieve  $O(n^2)$  with dynamic programming

# Affine Gap Penalty Recurrences

$$F_{i,j}^{\downarrow} = \max \begin{cases} F_{i-1,j}^{\downarrow} - \sigma \\ G_{i-1,j} - (\rho + \sigma) \end{cases}$$

Continue Gap in  $w$  (deletion)  
Start Gap in  $w$  (deletion): from middle

$$E_{i,j}^{\rightarrow} = \max \begin{cases} E_{i,j-1}^{\rightarrow} - \sigma \\ G_{i,j-1} - (\rho + \sigma) \end{cases}$$

Continue Gap in  $v$  (insertion)  
Start Gap in  $v$  (insertion): from middle

$$G_{i,j} = \max \begin{cases} G_{i-1,j-1} + \delta(v_i, w_j) \\ F_{i,j}^{\downarrow} \\ E_{i,j}^{\rightarrow} \end{cases}$$

Match or Mismatch

End deletion: from top

End insertion: from bottom

# Affine Gap Penalty Recurrences (cont)

S ..... i  
T ..... j

**Type 1:**  $G(i,j)$  is the max value of any alignment  
where  $s_i$  and  $t_j$  match (or mismatch)

S ..... i -----  
T ..... j

**Type 2:**  $E(i,j)$  is the max value of any alignment  
where  $t_j$  matches a space

S ..... i  
T ..... j -----

**Type 3:**  $F(i,j)$  is the max value of any alignment  
where  $s_i$  matches a space

---

# Affine Gap Penalty Recurrences (cont)

## Initialization:

$$E(0,0) = -\text{Infinity}$$

$$E(0,j) = 0, \text{ for } j > 0$$

$$E(i,0) = Wg + i * We, \text{ for } i > 0$$

$$F(0,0) = -\text{Infinity}$$

$$F(i,0) = 0, \text{ for } i > 0$$

$$F(0,j) = Wg + j * We, \text{ for } j > 0$$

$$G(0,0) = -\text{Infinity}$$

$$G(i,0) = 0, \text{ for } i > 0$$

$$G(0,j) = 0, \text{ for } j > 0$$

$$V(0,0) = 0$$

$$V(i,0) = Wg + i * We, \text{ for } i > 0$$

$$V(0,j) = Wg + j * We, \text{ for } j > 0$$

**Wg**: gap opening penalty

**We**: gap extension penalty

$$E(i,j) = \max\{E(i,j-1)+We, V(i,j-1)+Wg+We\}$$

$$F(i,j) = \max\{F(i-1,j)+We, V(i-1,j)+Wg+We\}$$

$$G(i,j) = V(i-1,j-1) + \text{match}; \text{ if } \text{seq1}[i] == \text{seq2}[j]$$

$$G(i,j) = V(i-1,j-1) + \text{mismatch}; \text{ otherwise}$$

$$V(i,j) = \max\{G(i,j), E(i,j), F(i,j)\}$$

**S** ..... **i**                      **G(i,j)**  
**T** ..... **j**

**S** ..... **i** -----                      **E(i,j)**  
**T** ..... **j**

**S** ..... **i**                      **F(i,j)**  
**T** ..... **j** -----

# Affine Gap Penalty Recurrences (cont)

E:									
		A	T	A	T	T	G	C	
	-10000	0	0	0	0	0	0	0	Wg -16
A	-20	-24	-15	-19	-23	-27	-31	-35	Ws -4
T	-24	-28	-32	-10	-14	-18	-22	-26	Match 5
A	-28	-32	-36	-30	-5	-9	-13	-17	Mismatch -4
G	-32	-36	-40	-34	-25	-9	-13	-17	
C	-36	-40	-44	-38	-29	-29	-13	-17	
F:									
		A	T	A	T	T	G	C	
	-10000	-20	-24	-28	-32	-36	-40	-44	Alignment:
A	0	-24	-28	-32	-36	-40	-44	-48	ATA—GC
T	0	-15	-32	-36	-40	-44	-48	-52	ATATTGC
A	0	-19	-10	-30	-34	-38	-42	-46	
G	0	-23	-14	-5	-25	-29	-33	-37	
C	0	-27	-18	-9	-9	-29	-24	-37	
G:									
		A	T	A	T	T	G	C	
	-10000	0	0	0	0	0	0	0	
A	0	5	-24	-19	-32	-36	-40	-44	
T	0	-24	10	-19	-14	-18	-31	-35	
A	0	-19	-19	15	-14	-18	-22	-26	
G	0	-32	-23	-14	11	-9	-4	-17	
C	0	-36	-27	-18	-9	7	-13	1	
V:									
		A	T	A	T	T	G	C	
	0	-20	-24	-28	-32	-36	-40	-44	
A	-20	5	-15	-19	-23	-27	-31	-35	
T	-24	-15	10	-10	-14	-18	-22	-26	
A	-28	-19	-10	15	-5	-9	-13	-17	
G	-32	-23	-14	-5	11	-9	-4	-17	
C	-36	-27	-18	-9	-9	7	-13	1	
P:									
		A	T	A	T	T	G	C	
	←	←	←	←	←	←	←	←	
A	↑	↖	←	↖	←	←	←	←	
T	↑	↑	↖	←	↖	↖	←	←	
A	↑	↖	↑	↖	←	←	←	←	
G	↑	↑	↑	↑	↖	↖	↖	↖	
C	↑	↑	↑	↑	↖	↖	↖	↖	

---

# LOCAL ALIGNMENT

---



---

# Local vs. Global Alignment

- The Global Alignment Problem tries to 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 among paths between **arbitrary vertices**  $(i,j)$  and  $(i', j')$  in the edit graph.
-

# Local vs. Global Alignment

- The Global Alignment Problem tries to 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 among paths between **arbitrary vertices**  $(i,j)$  and  $(i', j')$  in the edit graph.
- In the edit graph with negatively-scored edges, Local Alignment may score higher than Global Alignment

# Local vs. Global Alignment (cont'd)

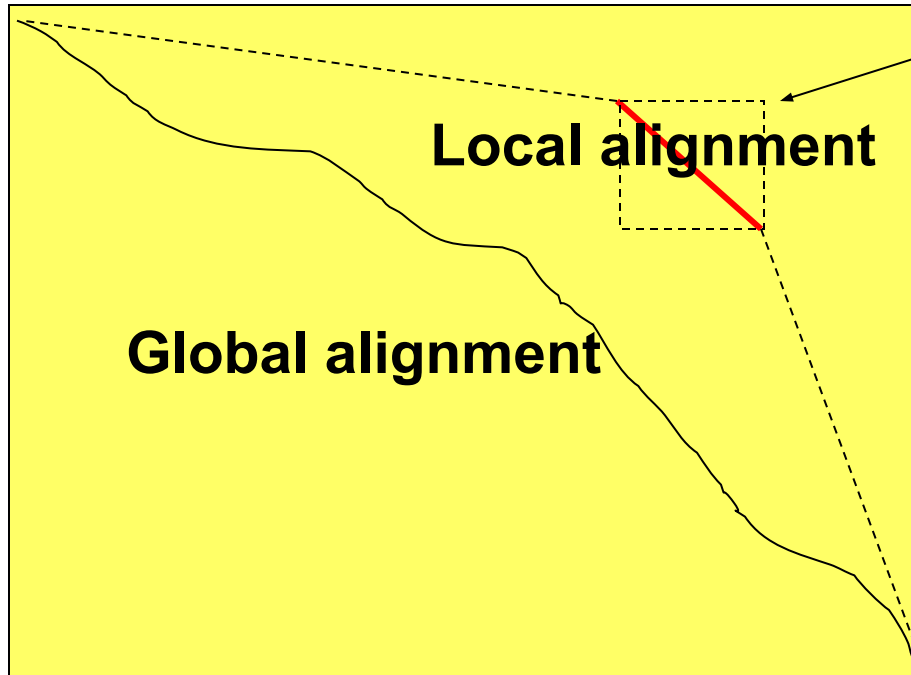
- **Global Alignment**

```
--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
            |||||
aattgccgccgctcgtttttcagCAGTTATGTCAGatc
```

# Local Alignment: Example



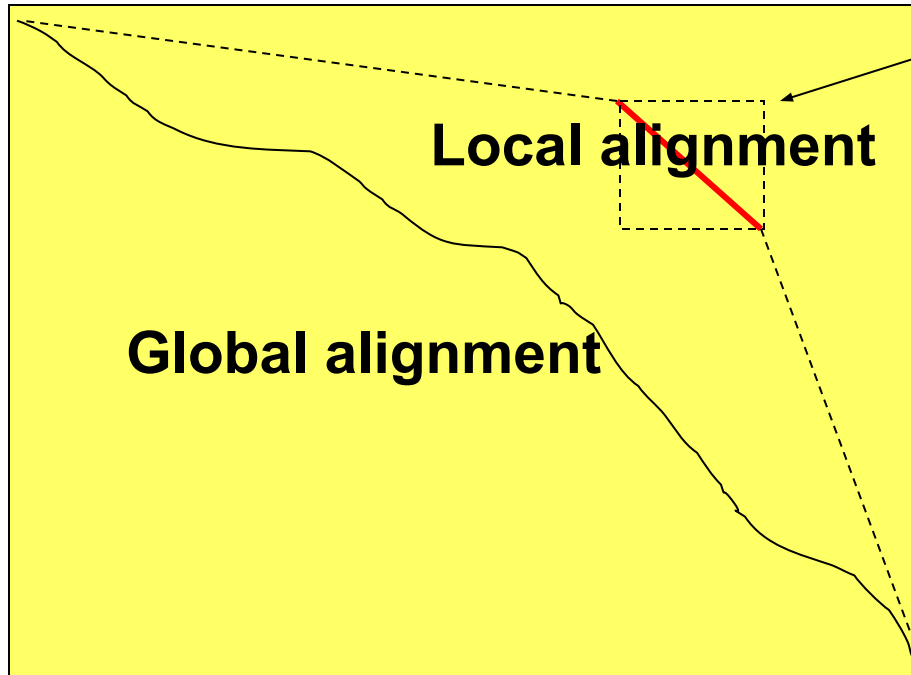
**Compute a “mini”  
Global Alignment  
to get Local**

---

# The Local Alignment Problem

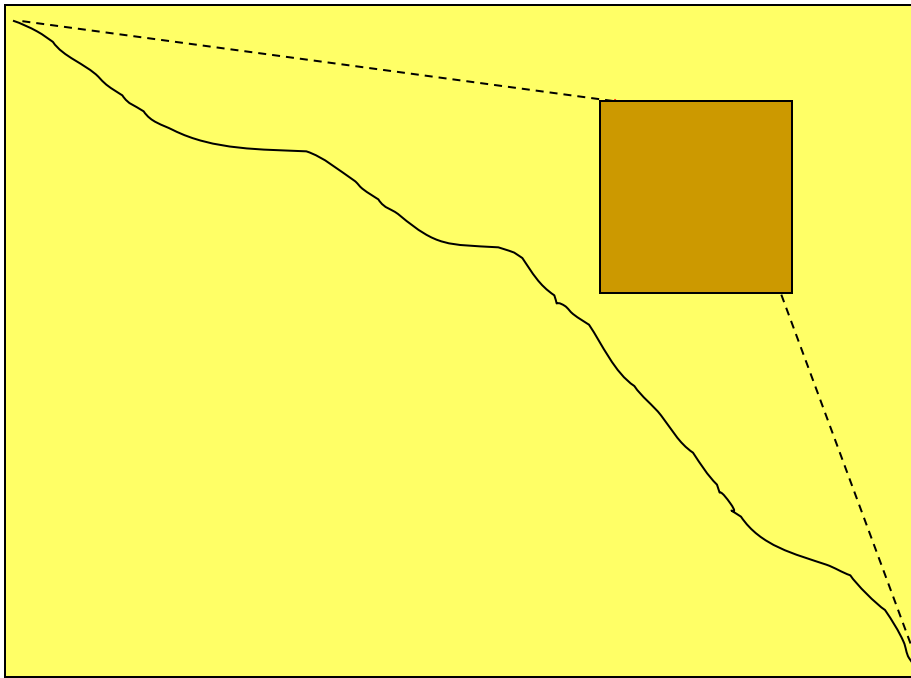
- Goal: Find the best local alignment between two strings
  - Input : Strings **v**, **w** and scoring matrix  $\delta$
  - Output : Alignment of substrings of **v** and **w** whose alignment score is maximum among all possible alignment of all possible substrings
-

# Local Alignment: Example

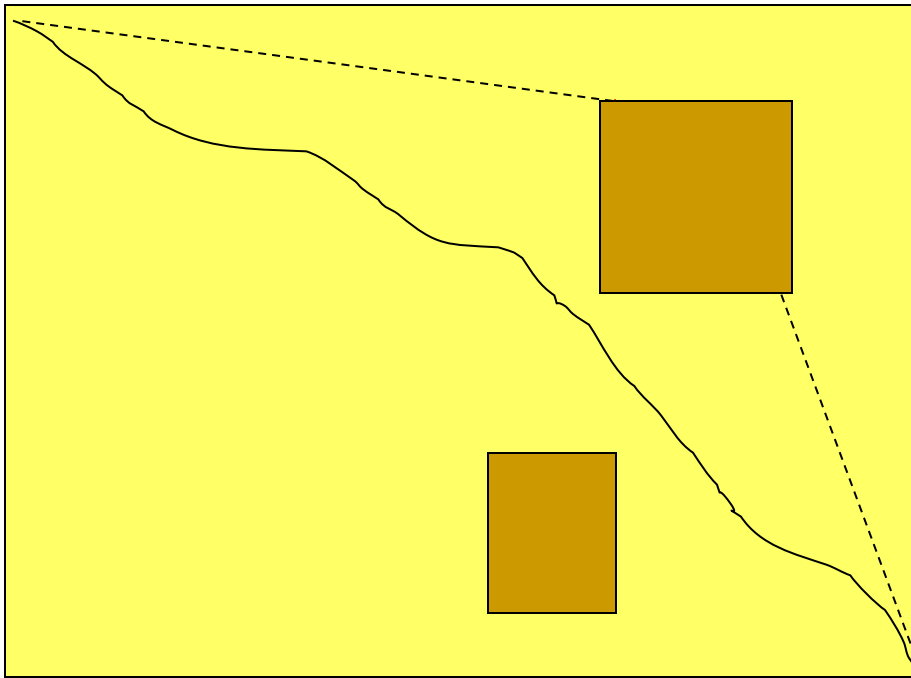


**Compute a “mini”  
Global Alignment  
to get Local**

# Local Alignment: Example

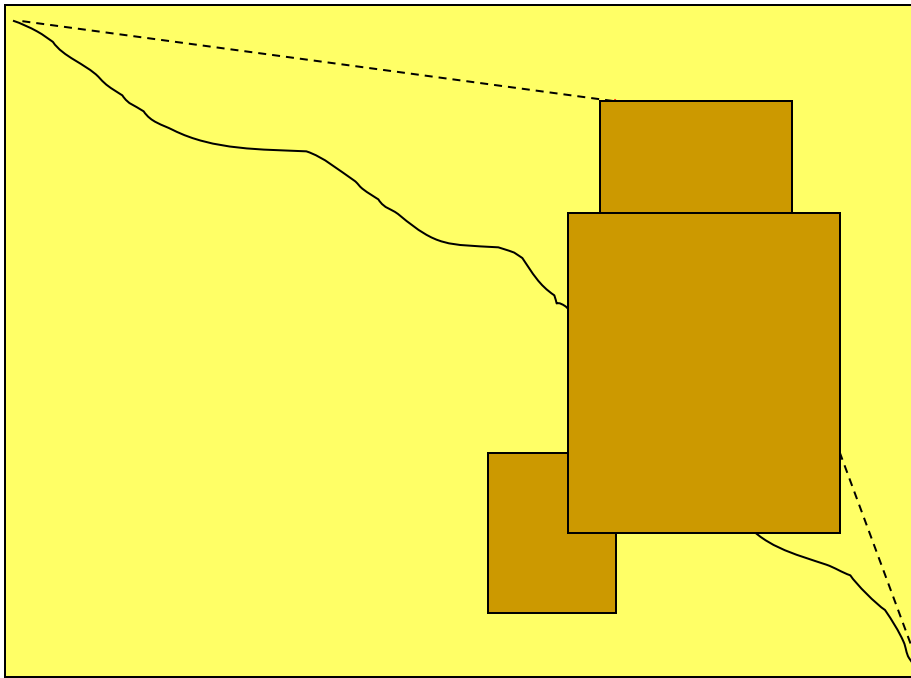


# Local Alignment: Example

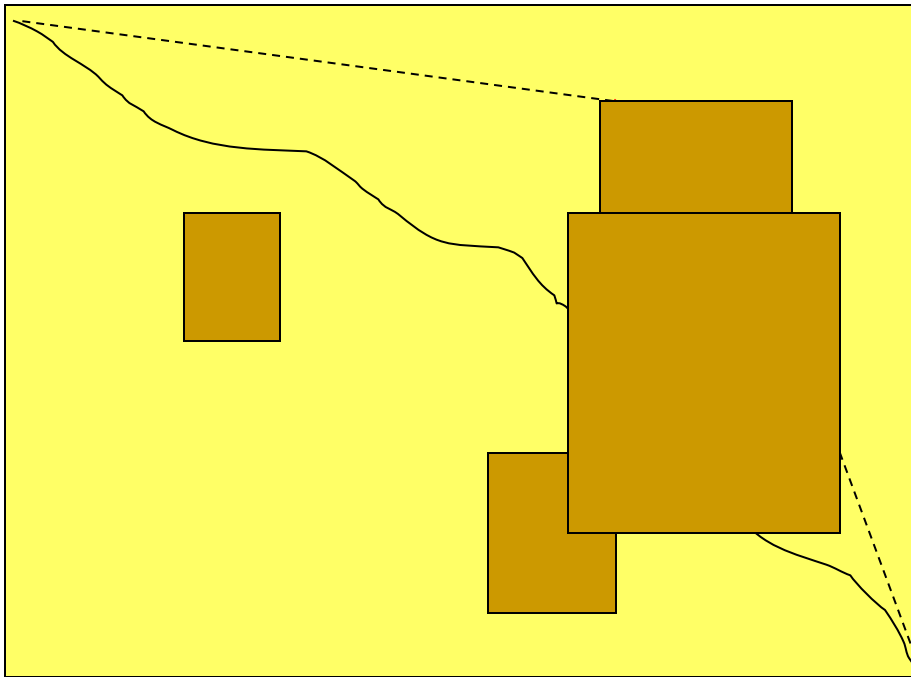




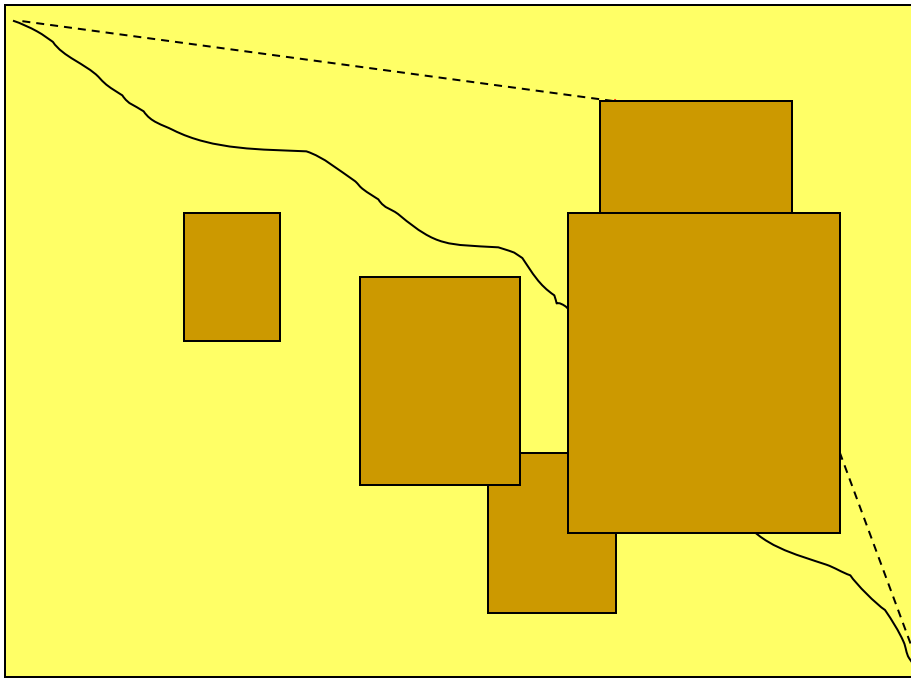
# Local Alignment: Example



# Local Alignment: Example

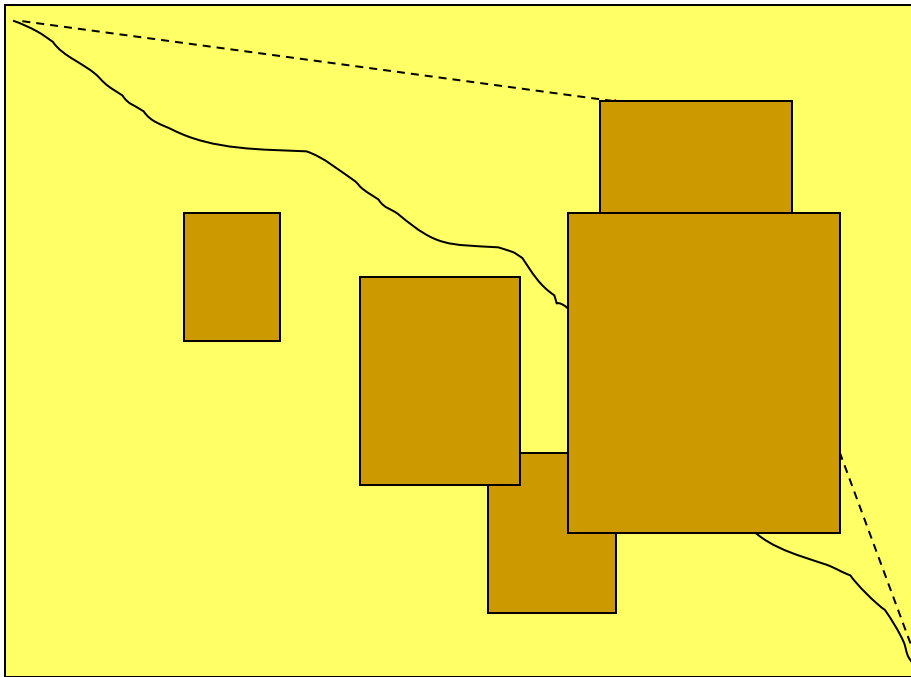


# Local Alignment: Example



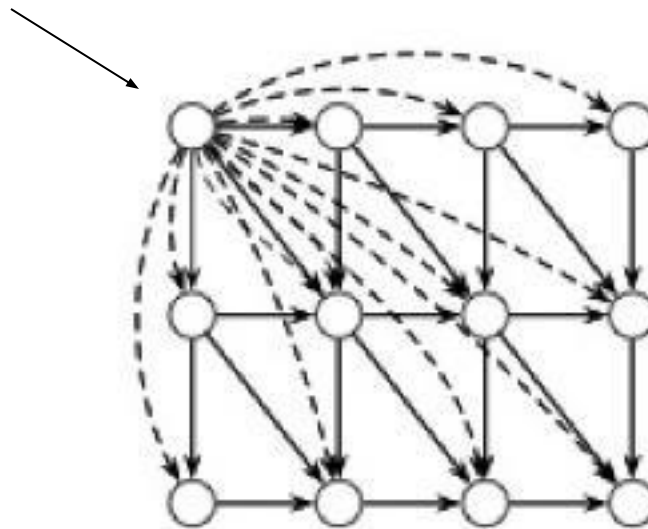
# Local Alignment: Running Time

- Long run time  $O(n^4)$ :
  - In the grid of size  $n \times n$  there are  $\sim n^2$  vertices  $(i,j)$  that may serve as a source.
  - For each such vertex computing alignments from  $(i,j)$  to  $(i',j')$  takes  $O(n^2)$  time.
- This can be remedied by giving free rides



# Local Alignment: Free Rides

**Vertex (0,0)**



**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.
- The recurrence:

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

there is only this change from the original recurrence of a Global Alignment - since there is only one “free ride” edge entering into every vertex

---

# Smith-Waterman: Traceback

- In the traceback, start with the cell that has the highest score and work back until a cell with a score of 0 is reached

# Example

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
A								
C								
G								
C								
G								
A								



# Example: initialize

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

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

# Example: fill in

S1 = ACGCGA  
S2 = CGAGTGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	0	0	0	0	0	0	0
A	0	0 (d)	0 (d)	1 (d)	0 (d)	0 (d)	0 (d)	1 (d)
C	0							
G	0							
C	0							
G	0							
A	0							

D: diagonal L: left U: up

# Example: backtrack

A  
A

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	0	0	0	0	0	0	0
A	0	0 (d)	0 (d)	1 (d)	0 (d)	0 (d)	0 (d)	1 (d)
C	0	1	0	0	0	0	0	0
G	0	0	2	0	1	0	1	0
C	0	1	0	1	0	0	0	0
G	0	0	2	0	2	0	1	0
A	0	0	0	3	1	1	0	2

D: diagonal L: left U: up

# Example: backtrack

GA  
GA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	0	0	0	0	0	0	0
A	0	0 (d)	0 (d)	1 (d)	0 (d)	0 (d)	0 (d)	1 (d)
C	0	1	0	0	0	0	0	0
G	0	0	2	0	1	0	1	0
C	0	1	0	1	0	0	0	0
G	0	0	2	0	2	0	1	0
A	0	0	0	3	1	1	0	2

D: diagonal L: left U: up

# Example: backtrack

CGA  
CGA

Match = +1  
Mismatch = -1  
Gap = -2

		C	G	A	G	T	G	A
	0	0	0	0	0	0	0	0
A	0	0 (d)	0 (d)	1 (d)	0 (d)	0 (d)	0 (d)	1 (d)
C	0	1	0	0	0	0	0	0
G	0	0	2	0	1	0	1	0
C	0	1	0	1	0	0	0	0
G	0	0	2	0	2	0	1	0
A	0	0	0	3	1	1	0	2

D: diagonal L: left U: up