# Sequence Alignments

Hammad Naveed
hammad.naveed@nu.edu.pk

# Local vs Global Alignments

- Global Alignment

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

- Local Alignment

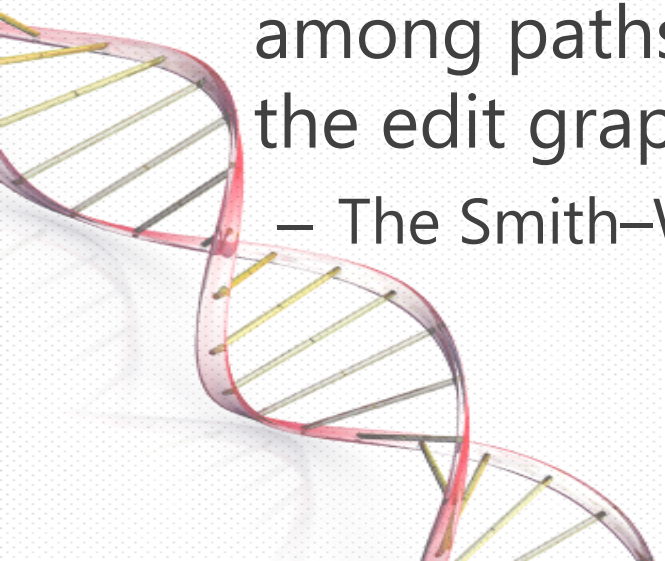
```
                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       |||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

better alignment to find conserved segment

# Local vs Global Alignments

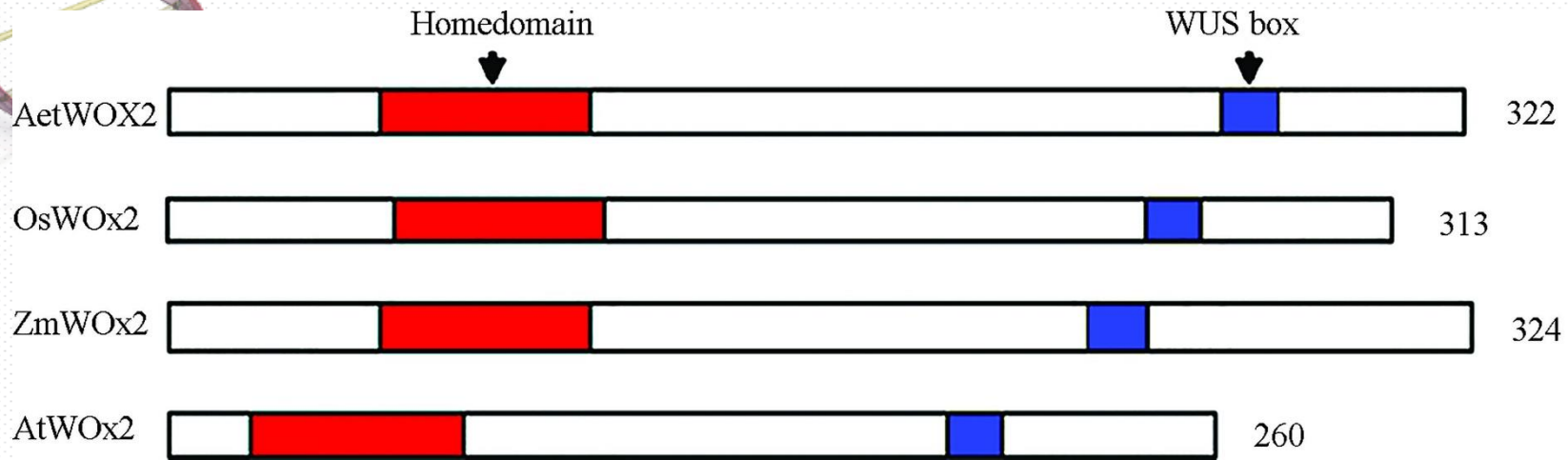
- The Global Alignment Problem tries to find the longest path between vertices *(0,0)* and *(n,m)* in the edit/alignment graph.
  - The Needleman–Wunsch algorithm

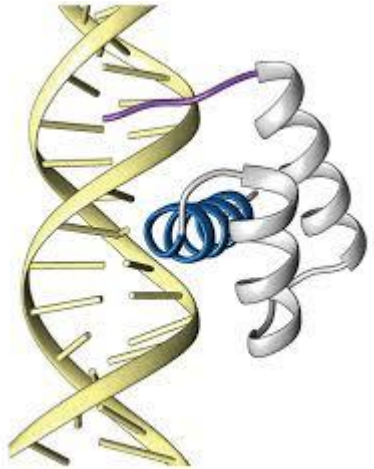
- The Local Alignment Problem tries to find the longest path among paths between **arbitrary vertices** *(i,j)* and *(i', j')* in the edit graph
  - The Smith–Waterman algorithm



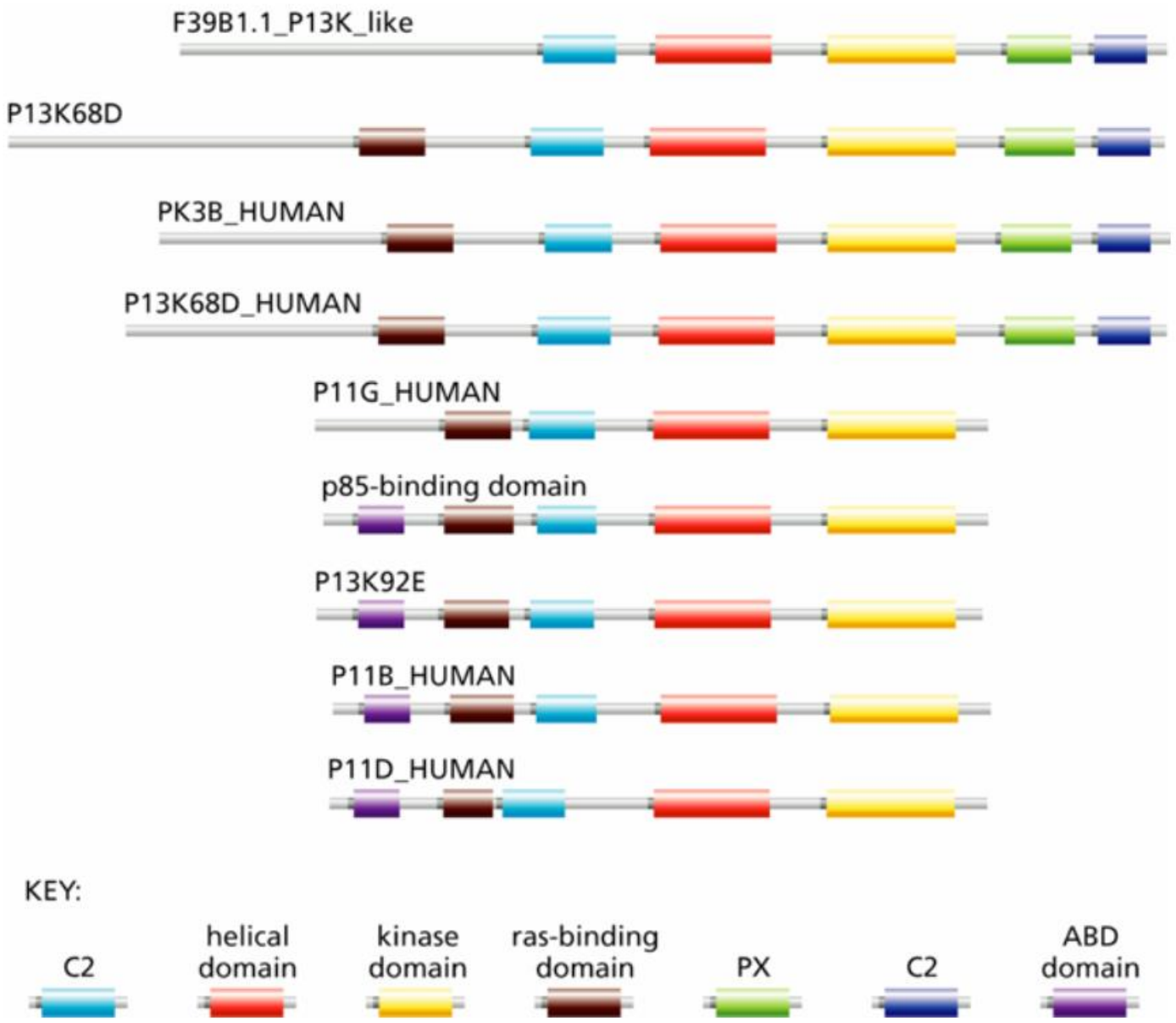
# Local alignments: why?

- Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions
- Example
  - Homeobox genes have a short region called the homeodomain that is highly conserved between species
  - A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence



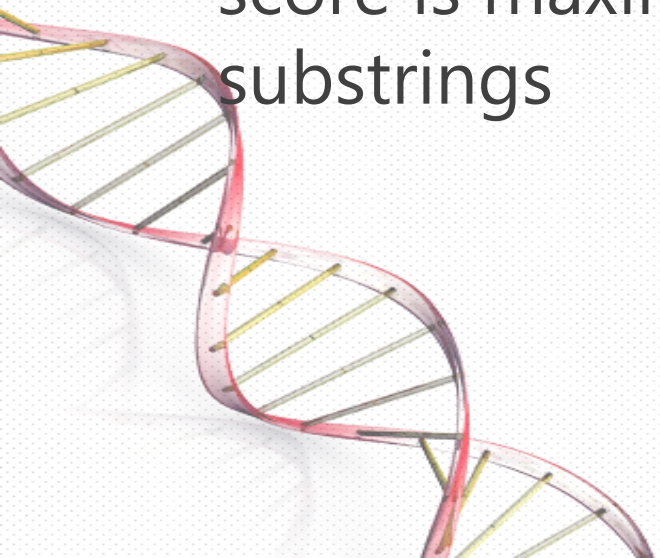



# Multidomain proteins

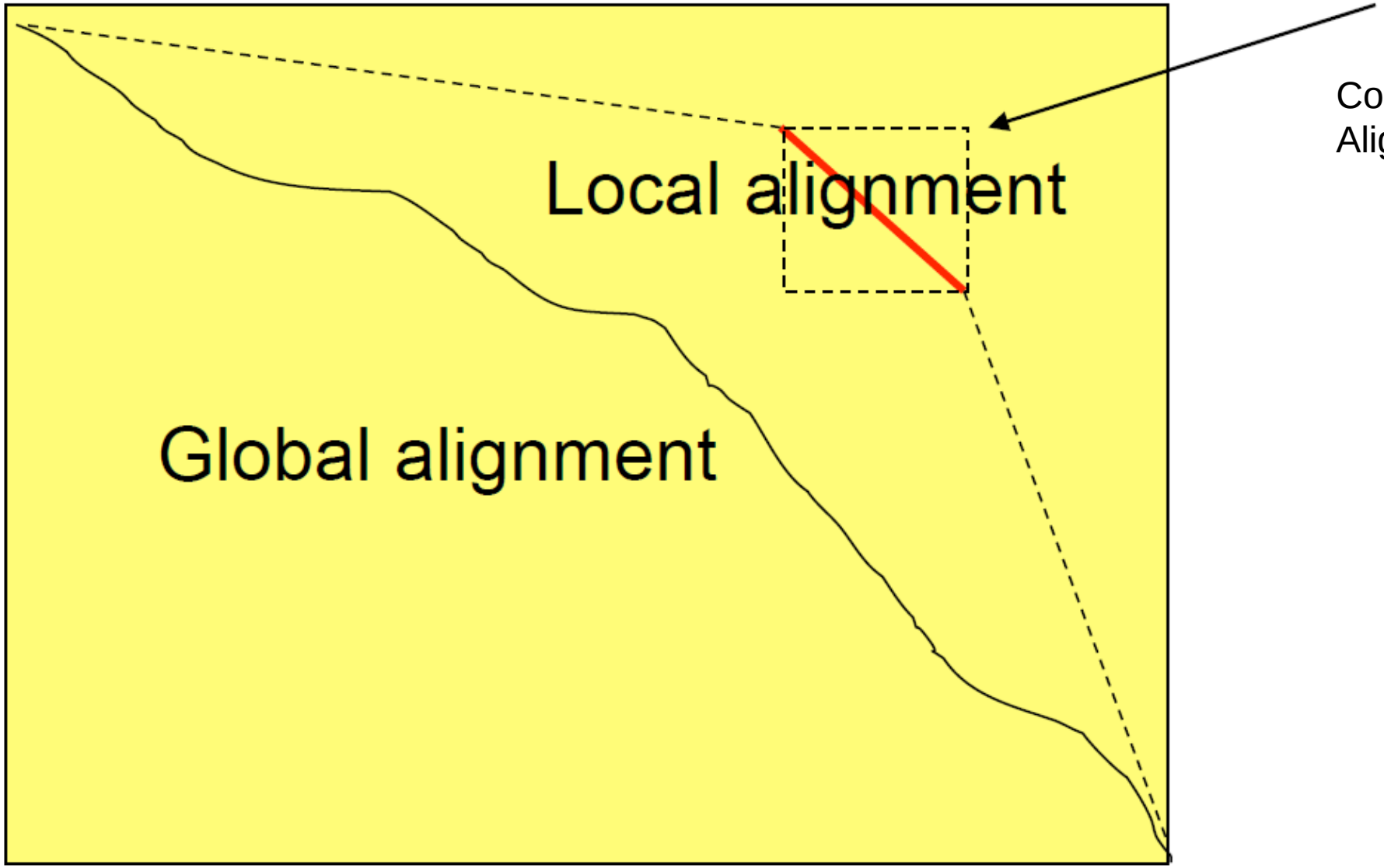


# The local alignment problem

- 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

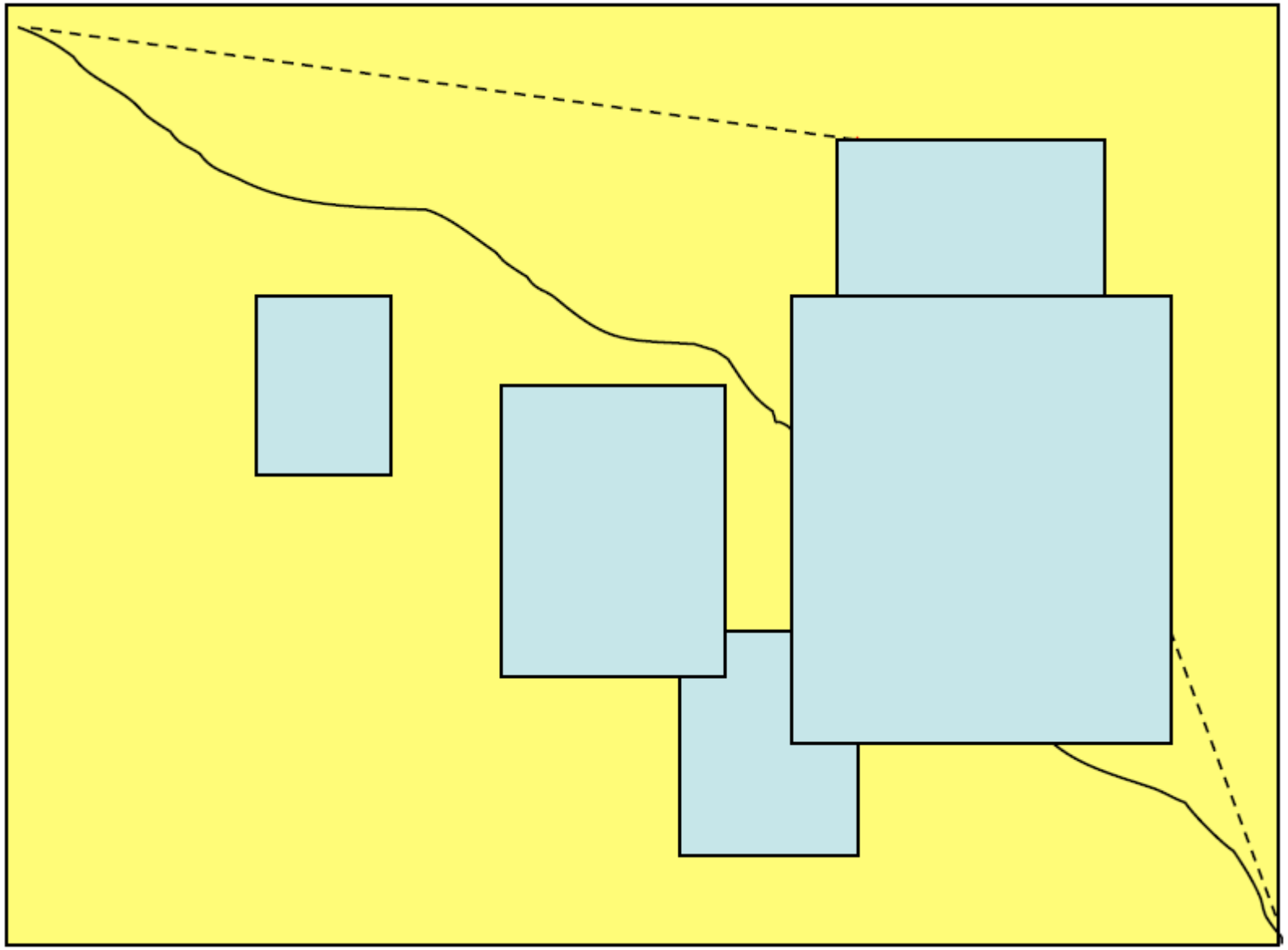


# Local alignment: example




Compute a “mini” Global Alignment to get Local

# Local alignment: running time



- In the grid of size *n x n* there are ~$n^2$ vertices *(i,j)* that may serve as a source and ~$n^2$ vertices *(i',j')* that may serve as a sink.
- - For each such pair of vertices, computing alignments from *(i,j)* to *(i',j')* takes $O(n^2)$ time

We do NOT go with this algorithm!

# 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:

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

Notice there is only this change from the original recurrence of a Global Alignment

- Complexity: $O(N^2)$, or $O(MN)$

# Local Alignment Example

Match: +1
Mismatch: -1
Space: -2

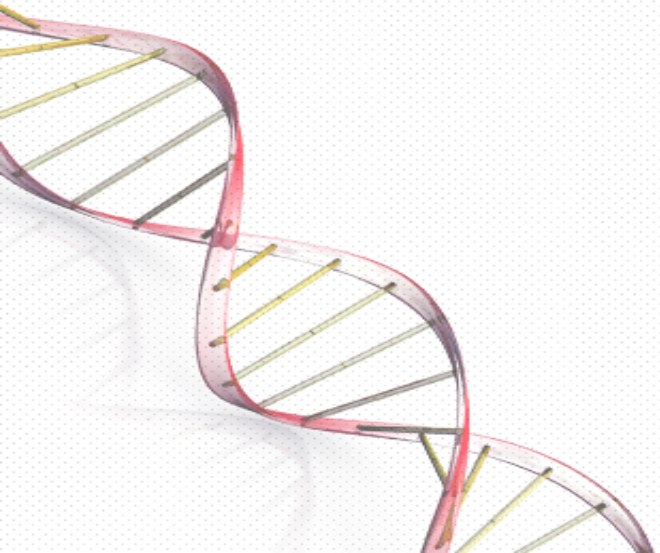
| | | A | A | G | A |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 0 | 0 |
| T | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 0 | 1 |
| A | 0 | 1 | 2 | 0 | 1 |
| G | 0 | 0 | 0 | $\boxed{3}$ | 1 |

x: A A G

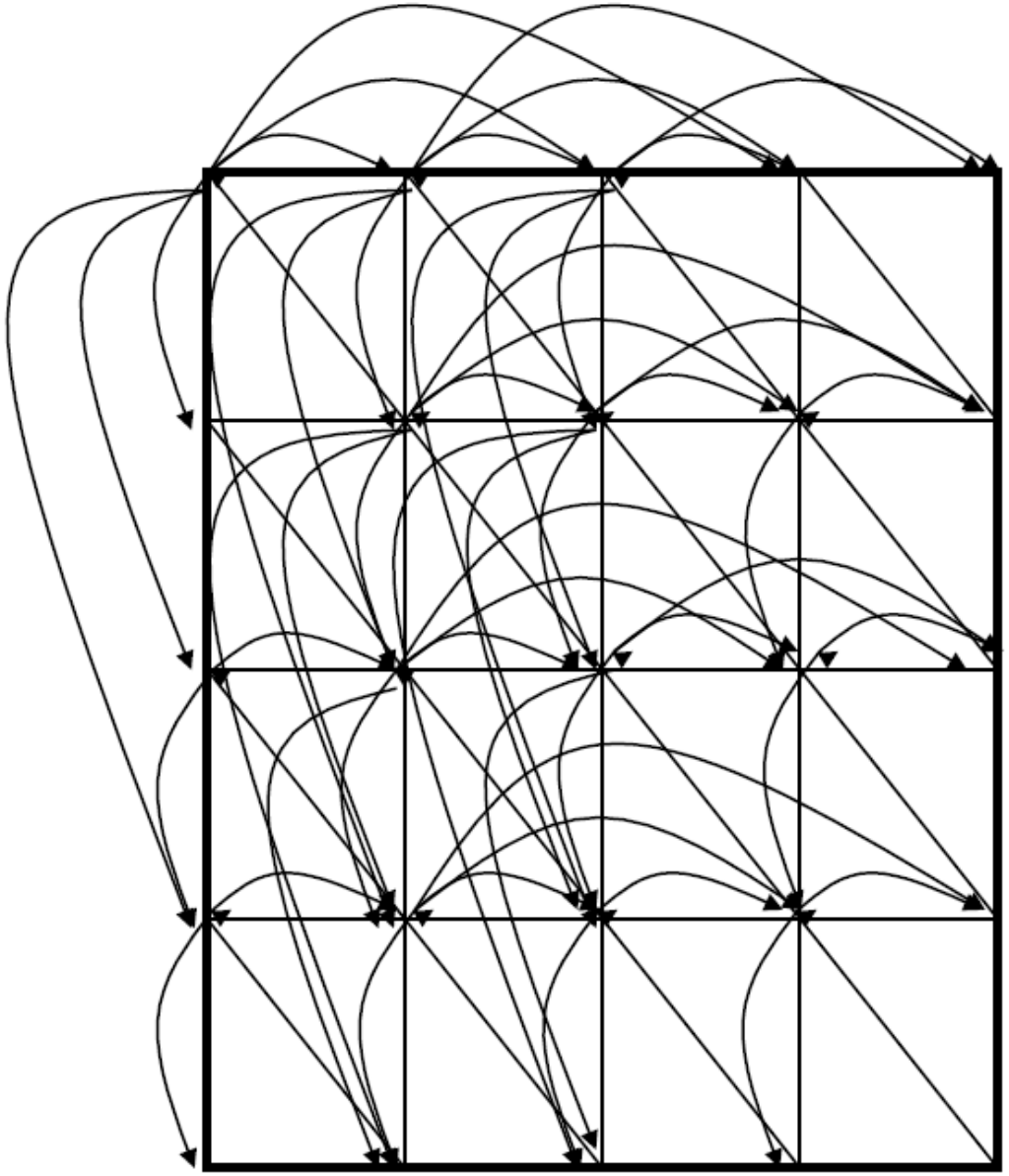
y: A A G

# Scoring indels: naive approach

- A fixed penalty σ is given to every indel:
  - -σ 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



# Arbitrary gap penalty?



- There are many such edges!
- 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 $O(n^3)$

# 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        ATAG_GC
ATATTGC         AT_GTGC
```

This is more likely

Normal scoring would give the same score for both alignments

This is less likely

# Affine gap penalties

- Score for a gap of length $x$ is:

$$-(\rho + \sigma x)$$

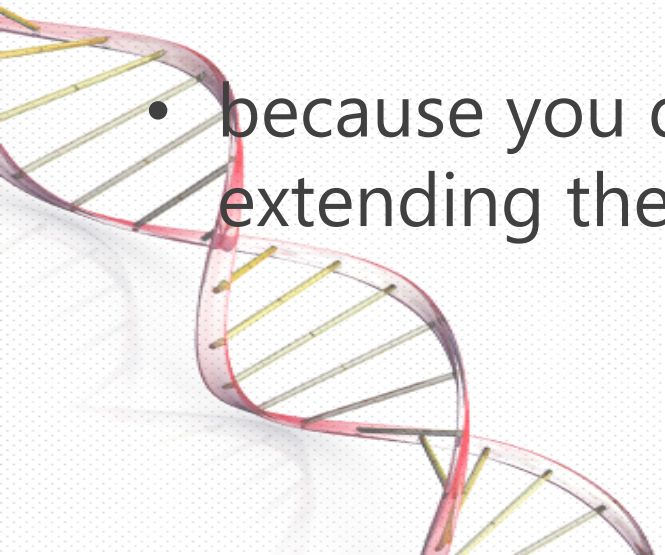
where $\rho > 0$ is the penalty for introducing a gap:

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 penalty recurrences

$$D(i,j) = max \begin{cases} D(i-1,j) + \sigma \\ S(i-1,j) + (\sigma + \rho) \end{cases}$$

Continue gap in *y* (deletion)
Start gap in *y* (deletion)

$$I(i,j) = max \begin{cases} I(i,j-1) + \sigma \\ S(i,j-1) + (\sigma + \rho) \end{cases}$$

Continue gap in x (insertion)
Start gap in x (insertion)

$$S(i,j) = max \begin{cases} S(i-1,j-1) + \delta(x_i, y_j) \\ I(i,j) \\ D(i,j) \end{cases}$$

Match or mismatch
End with deletion
End with insertion

# Substitution Matrices

| | A | R | N | D | C | Q | E | G | H | I | L | K | M | F | P | S | T | W | Y | V |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 4 | | | | | | | | | | | | | | | | | | | |
| R | -1 | 5 | | | | | | | | | | | | | | | | | | |
| N | -2 | 0 | 6 | | | | | | | | | | | | | | | | | |
| D | -2 | -2 | 1 | 6 | | | | | | | | | | | | | | | | |
| C | 0 | -3 | -3 | -3 | 9 | | | | | | | | | | | | | | | |
| Q | -1 | 1 | 0 | 0 | -3 | 5 | | | | | | | | | | | | | | |
| E | -1 | 0 | 0 | 2 | -4 | 2 | 5 | | | | | | | | | | | | | |
| G | 0 | -2 | 0 | -1 | -3 | -2 | -2 | 6 | | | | | | | | | | | | |
| H | -2 | 0 | 1 | -1 | -3 | 0 | 0 | -2 | 8 | | | | | | | | | | | |
| I | -1 | -3 | -3 | -3 | -1 | -3 | -3 | -4 | -3 | 4 | | | | | | | | | | |
| L | -1 | -2 | -3 | -4 | -1 | -2 | -3 | -4 | -3 | 2 | 4 | | | | | | | | | |
| K | -1 | 2 | 0 | -1 | -3 | 1 | 1 | -2 | -1 | -3 | -2 | 5 | | | | | | | | |
| M | -1 | -1 | -2 | -3 | -1 | 0 | -2 | -3 | -2 | 1 | 2 | -1 | 5 | | | | | | | |
| F | -2 | -3 | -3 | -3 | -2 | -3 | -3 | -3 | -1 | 0 | 0 | -3 | 0 | 6 | | | | | | |
| P | -1 | -2 | -2 | -1 | -3 | -1 | -1 | -2 | -2 | -3 | -3 | -1 | -2 | -4 | 7 | | | | | |
| S | 1 | -1 | 1 | 0 | -1 | 0 | 0 | 0 | -1 | -2 | -2 | 0 | -1 | -2 | -1 | 4 | | | | |
| T | 0 | -1 | 0 | -1 | -1 | -1 | -1 | -2 | -2 | -1 | -1 | -1 | -1 | -2 | -1 | 1 | 5 | | | |
| W | -3 | -3 | -4 | -4 | -2 | -2 | -3 | -2 | -2 | -3 | -2 | -3 | -1 | 1 | -4 | -3 | -2 | 11 | | |
| Y | -2 | -2 | -2 | -3 | -2 | -1 | -2 | -3 | 2 | -1 | -1 | -2 | -1 | 3 | -3 | -2 | -2 | 2 | 7 | |
| V | 0 | -3 | -3 | -3 | -1 | -2 | -2 | -3 | -3 | 3 | 1 | -2 | 1 | -1 | -2 | -2 | 0 | -3 | -1 | 4 |
| * | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 | -4 |

BLOSUM62 matrix