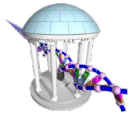


# Comp 555 - BioAlgorithms - Spring 2022



CLUSTAL O(1.2.1) multiple sequence alignment

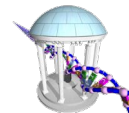
```
Cat      MAPWTRLLPLLALLSLWIPAPTRAFVNHLCGSHLVEALYLVCGERGFFYTPKARREAED 60
Pig      MALWTRLLPLLALLALWAPAPAQAFVNHLCGSHLVEALYLVCGERGFFYTPKARREAEN 60
Human    MALWMRLLPLLALLALWGPDPAAAFVNHLCGSHLVEALYLVCGERGFFYTPKTRREAED 60
Dog      MALWMRLLPLLALLALWAPAPTRAFVNHLCGSHLVEALYLVCGERGFFYTPKARREVED 60
** * *****: ** * *: *****:***.*:
```

```
Cat      LQKDAELGEAPGAGGLQPSALEAPLQKRGIVEQCASVCSLYQLEHYCN 110
Pig      PQAGAVELGG--GLGGLQALALEGPPQKRGIVEQCCTSICSLYQLENYCN 108
Human    LQ-----GSLQPLALEGSLQKRGIVEQCCTSICSLYQLENYCN 98
Dog      LQVRDVELAGAPGEGGLQPLALEGALQKRGIVEQCCTSICSLYQLENYCN 110
*                * . **   *** . *****: *: ***** . ***
```

- **PS#4 IS DUE ON THURSDAY 3/24**
- **I MAY NEED TO CALL OFF CLASS THURSDAY (MONITOR THE COURSE WEBSITE)**

## Advanced Sequence Alignment

# A Refresher

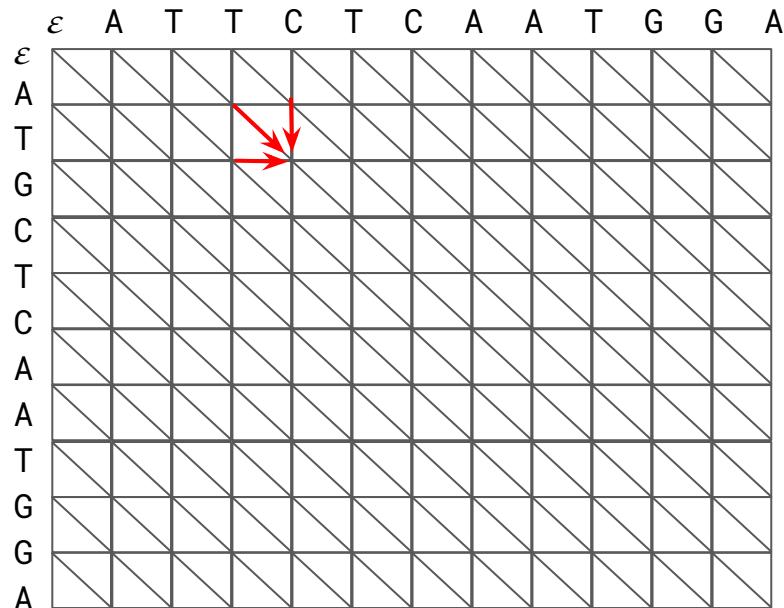


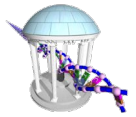
## Global Sequence Alignments

1. Construct a graph
2. Define a recurrence relation
3. Solve it for all paths from (0,0) to (n,m)
4. Used a dynamic program where each step relies only on solutions already computed and saved in our tableau

Global Alignment Recurrence relation:

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + \text{score}(v_i, w_j) \\ S_{i-1,j} - \text{indel} \\ S_{i,j-1} - \text{indel} \end{cases}$$





# Local vs. Global Alignment

- The *Global Alignment Problem* tries to find the highest scoring path between vertices  $(0,0)$  and  $(n,m)$  in the edit graph.
- The *Local Alignment Problem* tries to find the highest scoring subpath between *all vertex pairs*  $(i_1, j_1)$  and  $(i_2, j_2)$  in the edit graph where  $i_2 > i_1$  and  $j_2 > j_1$ .
- In an edit graph with negatively-weighted scores, a Local Alignment may score higher than a Global Alignment

Example:

- Global Alignment finds a match for the entire sequence

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

- Local Alignment finds a long conserved subsequence

```
                tccCAGTTATGTcAGgggacacgagcatgcagagac
                  ||| ||| ||| ||| ||| ||| ||| |||
aattgccgccgctcgttttcagCAGTTATGTcAGatc
```

# Local Alignments: Why?



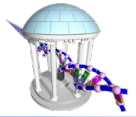
Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.

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

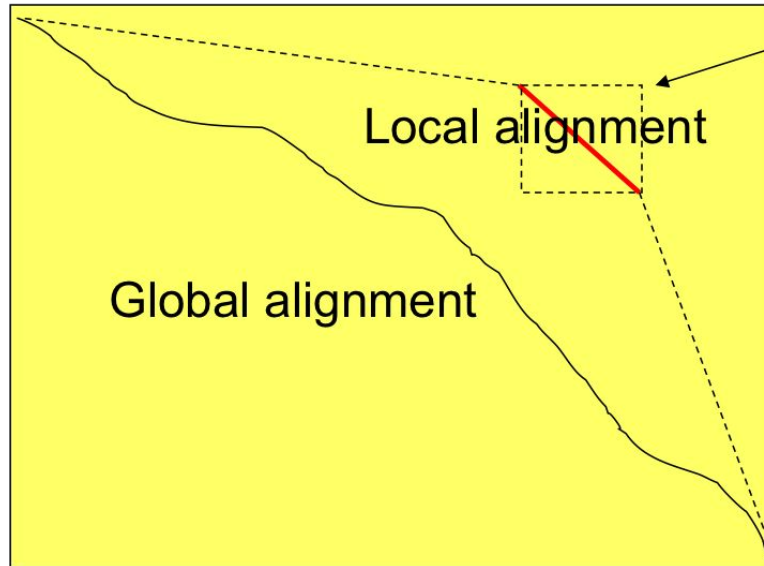
## Local Alignment Problem:

- **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 Approach

A local alignment is a subpath in a global alignment

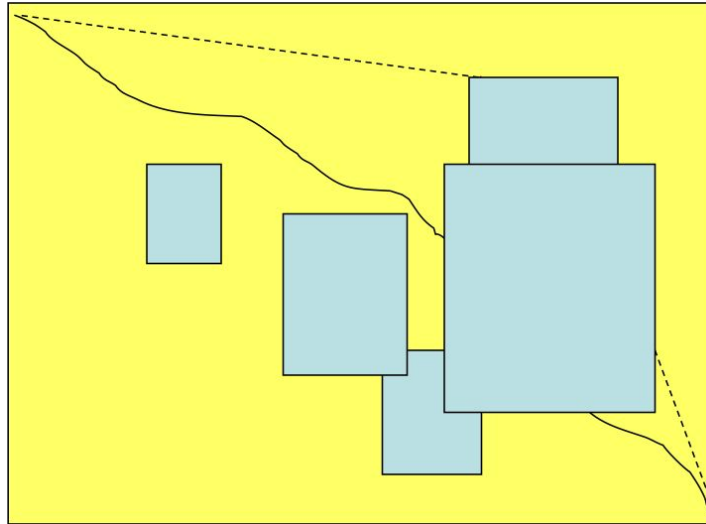


Compute a “mini”  
Global Alignment to  
get Local



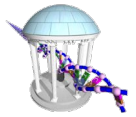
# Brute Force Local Alignment

Find the best global alignment among all blocks  $(i_1, j_1, i_2, j_2)$



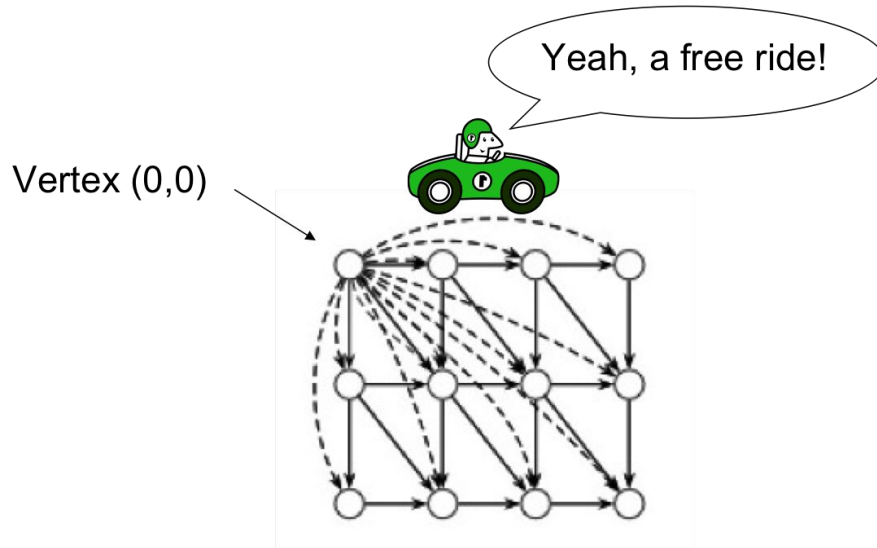
Long run time  $O(n^4)$ :

- In the grid of size  $n \times n$  there are  $O(n^2)$  vertices  $(i_1, j_1)$  that may serve as a source.
- For each such vertex computing alignments from  $(i_1, j_1)$  to  $(i_2, j_2)$  takes  $O(n^2)$  time.



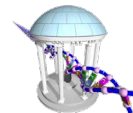
# Local Alignment with Free Rides

- **Key Ideas:** Add extra edges to our graph, consider all scores in matrix



- The dashed edges represent a *free ride* from (0,0) to any other node
- The largest value of  $s_{i,j}$  over the *whole score matrix* is the end point of the best local alignment (instead of  $s_{n,m}$ )

# Local Alignment Recurrence



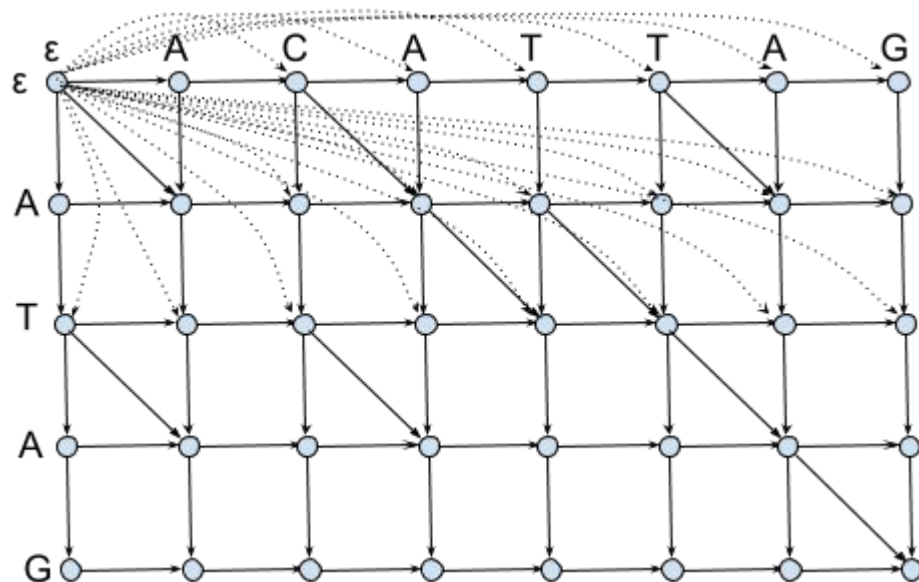
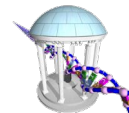
$$s_{i,j} = \max \left\{ \begin{array}{l} 0 \\ s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{array} \right.$$

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

- The *zero* is our *free ride* that allows the node to restart with a score of 0 at any point
  - What does this imply?
- After solving for the entire score matrix, we then search for  $s_{i,j}$  with the highest score, this is  $(i_2, j_2)$
- We follow our back tracking matrix until we reach a score of 0, whose coordinate becomes  $(i_1, j_1)$

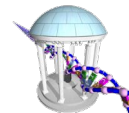


# Smith-Waterman Local Alignment



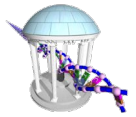
**Key Idea:** Add edges from the source to any intersection. These free rides might be better than any other path reaching an intersection.

# Local Alignment Example



	j=0	1	2	3	4	5	6	7	8	9	10	11	12
i=	-	G	C	T	G	G	A	A	G	G	C	A	T
0	-	0	0	0	0	0	0	0	0	0	0	0	0
1	G	0											
2	C	0											
3	A	0											
4	G	0											
5	A	0											
6	G	0											
7	C	0											
8	A	0											
9	C	0											
10	T	0											

Match = 5, Mismatch = -4, Indel = -7

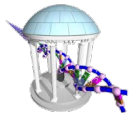


# Local Alignment Example - continued

	j=0	1	2	3	4	5	6	7	8	9	10	11	12
i=	-	G	C	T	G	G	A	A	G	G	C	A	T
0	-	0	0	0	0	0	0	0	0	0	0	0	0
1	G	0	$S_{1,1}$										
2	C	0											
3	A	0											
4	G	0											
5	A	0											
6	G	0											
7	C	0											
8	A	0											
9	C	0											
10	T	0											

$$S_{1,1} = \max \left\{ \begin{array}{l} S_{0,0} + S_{G,G} = 0 + 5 = 5 \\ S_{1,0} + w = 0 - 7 = -7 \\ S_{0,1} + w = 0 - 7 = -7 \\ 0 \end{array} \right\} = 5$$

Match = 5, Mismatch = -4, Indel = -7

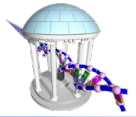


# Local Alignment Example - continued

	j=0	1	2	3	4	5	6	7	8	9	10	11	12
i=	-	G	C	T	G	G	A	A	G	G	C	A	T
0	-	0	0	0	0	0	0	0	0	0	0	0	0
1	G	0	5	0									
2	C	0	0	$S_{2,2}$									
3	A	0											
4	G	0											
5	A	0											
6	G	0											
7	C	0											
8	A	0											
9	C	0											
10	T	0											

$$S_{2,2} = \max \left\{ \begin{array}{l} S_{1,1} + s_{C,C} = 5 + 5 = 10 \\ S_{2,1} + w = 0 - 7 = -7 \\ S_{1,2} + w = 0 - 7 = -7 \\ 0 \end{array} \right\} = 10$$

Match = 5, Mismatch = -4, Indel = -7



# Local Alignment Example - continued

	0	G	C	T	G	G	A	A	G	G	C	A	T
0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	0	0	5	5	0	0	5	5	0	0	0
C	0	0	10	3	0	1	1	0	0	1	10	3	0
A	0	0	3	6	0	0	6	6	0	0	3	15	8
G	0	5	0	0	11	5	0	2	11	5	0	8	11
A	0	0	1	0	4	7	10	5	4	7	1	5	4
G	0	5	0	0	5	9	3	6	10	9	3	0	1
C	0	0	10	3	0	2	5	0	3	6	14	7	0
A	0	0	3	6	0	0	7	10	3	0	7	19	12
C	0	0	5	0	2	0	0	3	6	0	5	12	15
T	0	0	0	10	3	0	0	0	0	2	0	5	17

Match = 5, Mismatch = -4, Indel = -7

- Once the matrix is filled in we find the best alignment
- Rather than using the score of the last entry as we did for a global alignment, we **search for the entire matrix for the maximum entry** ( $O(m n)$  steps)



# Local Alignment Example - continued

	0	G	C	T	G	G	A	A	G	G	C	A	T
0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	0	0	5	5	0	0	5	5	0	0	0
C	0	0	10	3	0	1	1	0	0	1	10	3	0
A	0	0	3	6	0	0	6	6	0	0	3	15	8
G	0	5	0	0	11	5	0	2	11	5	0	8	11
A	0	0	1	0	4	7	10	5	4	7	1	5	4
G	0	5	0	0	5	9	3	6	10	9	3	0	1
C	0	0	10	3	0	2	5	0	3	6	14	7	0
A	0	0	3	6	0	0	7	10	3	0	7	19	12
C	0	0	5	0	2	0	0	3	6	0	5	12	15
T	0	0	0	10	3	0	0	0	0	2	0	5	17

Match = 5, Mismatch = -4, Indel = -7

- From the largest score attained, then backtrack from there until "0" is reached is reached to find the alignment.
- A global alignment would continue to backtrack until  $i,j = (0,0)$

# Local Alignment Example - continued



<b>G</b>	<b>C</b>	<b>T</b>	<b>G</b>	<b>G</b>	<b>A</b>	<b>A</b>	<b>G</b>	<b>-</b>	<b>G</b>	<b>C</b>	<b>A</b>	<b>T</b>	
				<b>G</b>	<b>C</b>	<b>A</b>	<b>G</b>	<b>A</b>	<b>G</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>

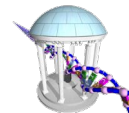
6 matches:  $6 \times 5 = 30$

1 mismatch: -4

1 indel: -7

Total: 19

# Local Alignment Code



```
In [11]: import numpy

def LocalAlign(v, w, scorematrix, indel):
    s = numpy.zeros((len(v)+1,len(w)+1), dtype="int32")
    b = numpy.zeros((len(v)+1,len(w)+1), dtype="int32")
    for i in range(1,len(v)+1):
        for j in range(1,len(w)+1):
            if (j == 0):
                if (i > 0):
                    s[i,j] = max(s[i-1,j] + indel, 0)
                    b[i,j] = 1
                continue
            if (i == 0):
                s[i,j] = max(s[i,j-1] + indel, 0)
                b[i,j] = 2
                continue
            score = s[i-1,j-1] + scorematrix[v[i-1],w[j-1]]
            vskip = s[i-1,j] + indel
            wskip = s[i,j-1] + indel
            s[i,j] = max(vskip, wskip, score, 0)
            if (s[i,j] == vskip):
                b[i,j] = 1
            elif (s[i,j] == wskip):
                b[i,j] = 2
            elif (s[i,j] == score):
                b[i,j] = 3
            else:
                b[i,j] = 0
    return (s, b)

match = {('A','A'): 5, ('A','C'): -4, ('A','G'): -4, ('A','T'): -4,
         ('C','A'): -4, ('C','C'): 5, ('C','G'): -4, ('C','T'): -4,
         ('G','A'): -4, ('G','C'): -4, ('G','G'): 5, ('G','T'): -4,
         ('T','A'): -4, ('T','C'): -4, ('T','G'): -4, ('T','T'): 5}

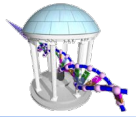
v = "GCTGGAAGGCAT"
w = "GCAGAGCACT"

s, b = LocalAlign(v,w,match,-7)
print(s)
print()
print(b)
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0]
 [ 0  5  0  0  5  0  5  0  0  0  0]
 [ 0  0 10  3  0  1  0 10  3  5  0]
 [ 0  0  3  6  0  0  0  3  6  0 10]
 [ 0  5  0  0 11  4  5  0  0  2  3]
 [ 0  5  1  0  5  7  9  2  0  0  0]
 [ 0  0  1  6  0 10  3  5  7  0  0]
 [ 0  0  0  6  2  5  6  0 10  3  0]
 [ 0  5  0  0 11  4 10  3  3  6  0]
 [ 0  5  1  0  5  7  9  6  0  0  2]
 [ 0  0 10  3  0  1  3 14  7  5  0]
 [ 0  0  3 15  8  5  0  7 19 12  5]
 [ 0  0  0  8 11  4  1  0 12 15 17]]
```

```
[[0 0 0 0 0 0 0 0 0 0 0]
 [0 3 0 0 3 0 3 0 0 0 0]
 [0 0 3 2 0 3 0 3 2 3 0]
 [0 0 1 3 0 0 0 1 3 0 3]
 [0 3 0 0 3 2 3 0 0 3 1]
 [0 3 3 0 3 3 3 2 0 0 0]
 [0 0 3 3 0 3 2 3 3 2 0]
 [0 0 0 3 3 3 3 0 3 2 0]
 [0 3 0 0 3 2 3 2 1 3 0]
 [0 3 3 0 3 3 3 3 0 0 3]
 [0 0 3 2 0 3 3 3 2 3 0]
 [0 0 1 3 2 3 0 1 3 2 2]
 [0 0 0 1 3 2 3 1 1 3 3]]
```





# Extracting the Local Alignment

```

def LocalAlignment(b,v,w,i,j):
    if (b[i,j] == 0):
        return ['', '']
    if (b[i,j] == 3):
        result = Alignment(b,v,w,i-1,j-1)
        result[0] += v[i-1]
        result[1] += w[j-1]
        return result
    if (b[i,j] == 2):
        result = Alignment(b,v,w,i,j-1)
        result[0] += "-"
        result[1] += w[j-1]
        return result
    if (b[i,j] == 1):
        result = Alignment(b,v,w,i-1,j)
        result[0] += v[i-1]
        result[1] += "-"
        return result

maxij = numpy.unravel_index(s.argmax(), s.shape)
print(maxij, s[maxij])
vsub, wsub = LocalAlignment(b,v,w,maxij[0],maxij[1])
print(v)
print(w)
print(vsub)
print(wsub)

```



The only difference here is terminating condition of the recursion. LocalAlignment() stops once a '0' (a free-ride) is reached.



s.argmax() finds the index of a maximum score in a "flattened" score array. unravel\_index() converts it to a 2D index pair.

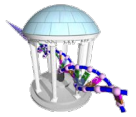
Note that LocalAlignment() only returns the "Local" part

V:    ----GCAGAGCACT  
          |  |  |  |  
W:   GCTGGAAG\_GCAT

```

(8, 11) 19
GCAGAGCACT
GCTGGAAGGCAT
GCAGAGCA
GAAG_GCA

```



# Scoring Indels: Naive Approach

ATCTTCAGCCATAAAAGATGAAGTT  
ATCTTCAGCCAAAGATGAAGTT

Reference

3 base deletion relative to the reference

ATCTTCAGCC---AAAGATGAAGTT  
ATCTTCAGCCA---AAGATGAAGTT  
ATCTTCAGCCA--A-AGATGAAGTT  
ATCTTCAGCCA-AA--GATGAAGTT  
ATCTTCAGCCA-A-A-GATGAAGTT

version 1

version 2

version 3

version 4

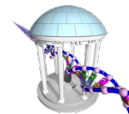
version 5

ATCTTCAGCCATA**TGTG**AAAGATGAAGTT

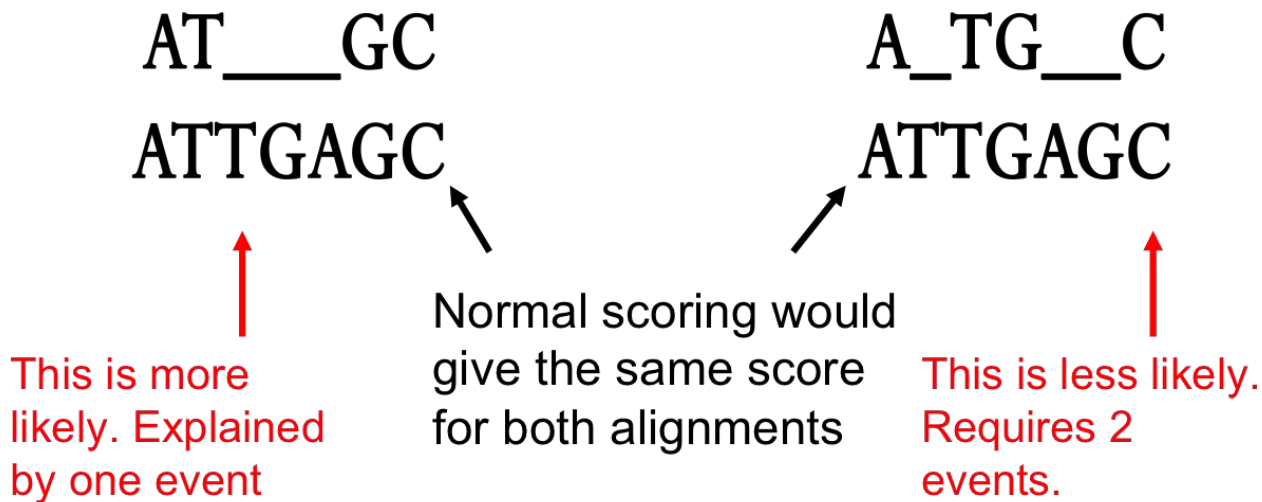
4 base insertion

- 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
  - large insertions or deletions might result from a single event

# Affine Gap Penalties



In nature, a series of  $k$  indels often come as a single, albeit rare, event rather than as a series of multiple events



# Accounting for Gaps



- Gaps- contiguous sequence of indels in a row
- Modify the scoring for a gap of length  $x$  to be:

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

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

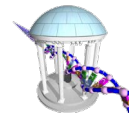
$\rho$  = gap opening penalty

- and  $\sigma$  is the cost of extending it further ( $\rho + \sigma \gg \sigma$ ):

$\sigma$  = gap extension penalty

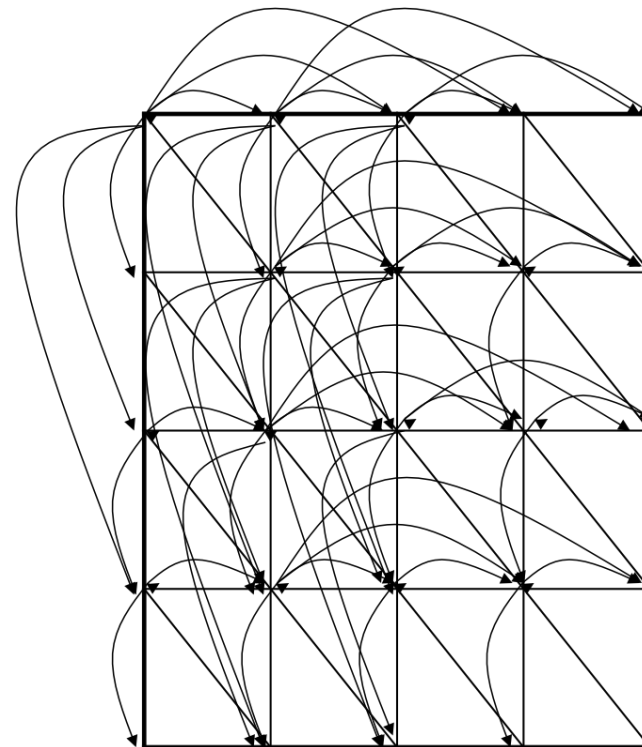
- because you do not want to add too much of a penalty for further extending the gap, once it is opened.

# Adding Affine Gap Penalties to our Graph

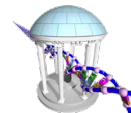


- To reflect affine gap penalties we have to add “long” horizontal and vertical edges to the edit graph.
- Each such edge of length  $x$  should have weight 
$$-\rho - x \cdot \sigma$$
- There are many such edges!
- Adding them to the graph increases the running time of the alignment algorithm by a factor of  $n$  (where  $n$  is the number of vertices)
- So the complexity increases from  $O(n^2)$  to  $O(n^3)$

Can we do it some other way?




# Adding Two More Tables



Affine Gap penalties can be more easily expressed in terms of 3 recurrences

Keep track of these intermediate values in two new tables


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

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

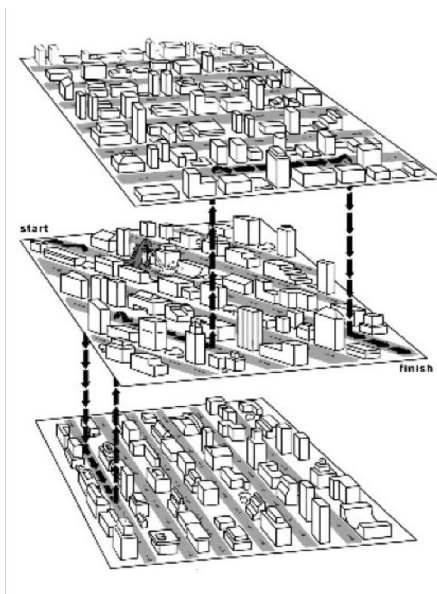
$$u_{i,j} = \max \begin{cases} u_{i,j-1} - \sigma \\ s_{i,j-1} - (\rho + \sigma) \end{cases}$$

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

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

Match or Mismatch  
End deletion: from top  
End insertion: from left

# A 3-level Manhattan Grid



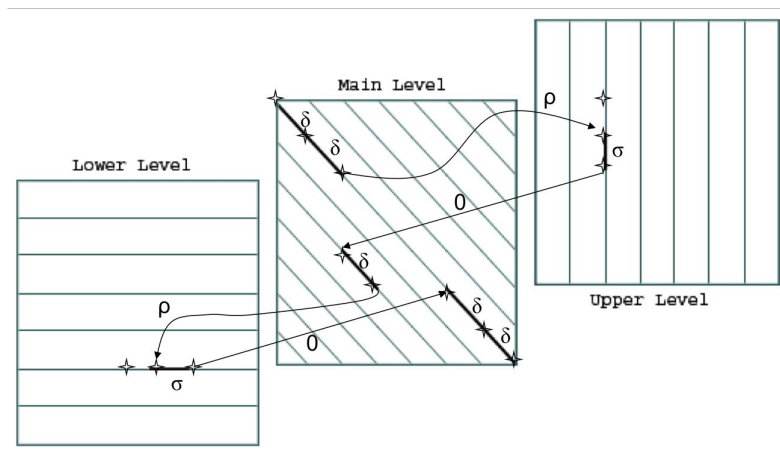
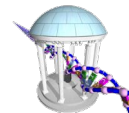
Gaps in  $w$  (t-table)

Matches/Mismatches (s-table)

Gaps in  $v$  (u-table)

- The three recurrences for the scoring algorithm creates a 3-layered graph.
- The top level creates/extends gaps in the sequence  $w$ .
- The bottom level creates/extends gaps in sequence  $v$ .
- The middle level extends matches and mismatches.

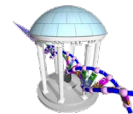
# Switching between Layers



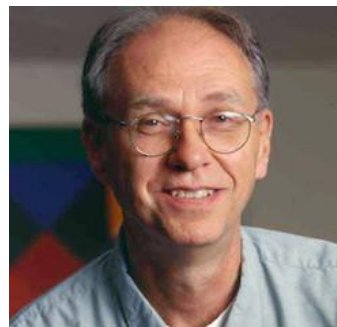
- Levels:
  - The main level is for diagonal edges
  - The lower level is for horizontal edges
  - The upper level is for vertical edges
- A jumping penalty is assigned to moving from the main level to either the upper level or the lower level ( $-\rho - \sigma$ )
- There is a gap extension penalty for each continuation on a level other than the main level ( $-\sigma$ )

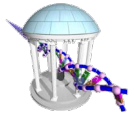


# Multiple versus Pairwise Alignment



- Up until now we have only tried to align two sequences.
- What about more than two? And why?
- A faint similarity between two sequences becomes significant if present in many
- Multiple alignments can reveal subtle similarities that pairwise alignments do not reveal





# Generalizing Pairwise Alignment

- Alignment of 2 sequences is represented as a 2-row matrix
- In a similar way, we represent alignment of 3 sequences as a 3-row matrix

```
A T _ G C G _  
A _ C G T _ A  
A T C A C _ A
```

- Score: more conserved columns, better alignment

# Three-D Alignment Paths



- An alignment of 3 sequences: ATGC, AATC, ATGC

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

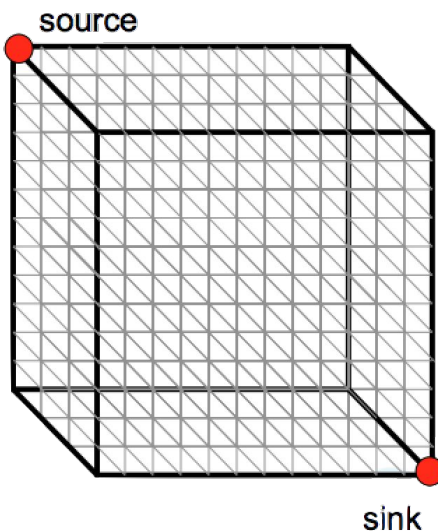
x coordinate

y coordinate

z coordinate

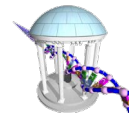
- Resulting path in (x,y,z) space:  
(0,0,0) → (1,1,0) → (1,2,1) → (2,3,2) → (3,3,3) → (4,4,4)
- Is there a better one?

# Aligning Three Sequences

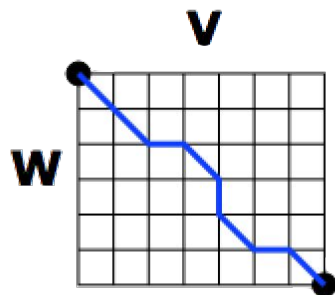


- Same strategy as aligning two sequences
- Use a 3-D “Manhattan Cube”, with each axis representing a sequence to align
- For global alignments, go from source to sink

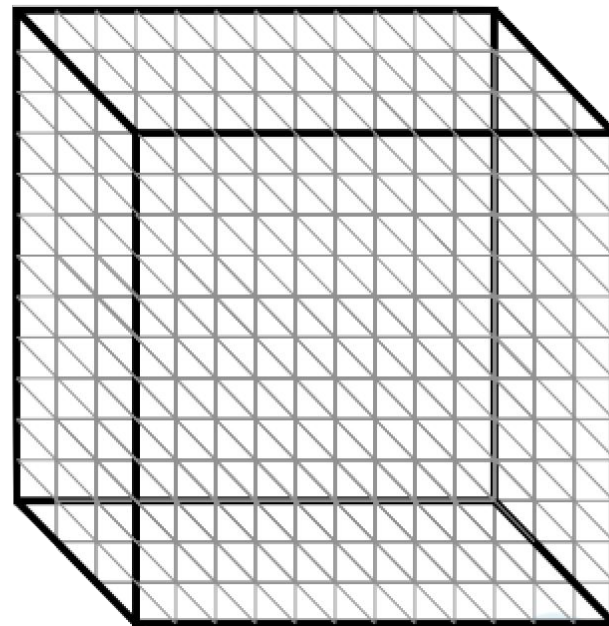
# 2-sequence vs 3-sequence Alignment



- In a 2-D grid there are 3 approaches to each intersection
- I'm now ignoring
  - Free-passes
  - Affine jumps
- How about 3-D?
- How does this impact our recurrence relations?



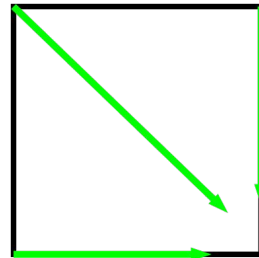
2-D edit graph



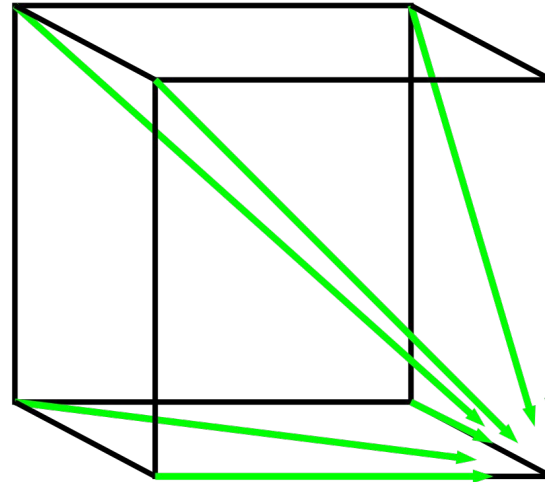
3-D edit graph



# A 2-D versus a 3-D neighborhood



In **2-D**, 3 edges  
lead to each  
interior vertex



In **3-D**, 7 edges lead to  
each interior vertex

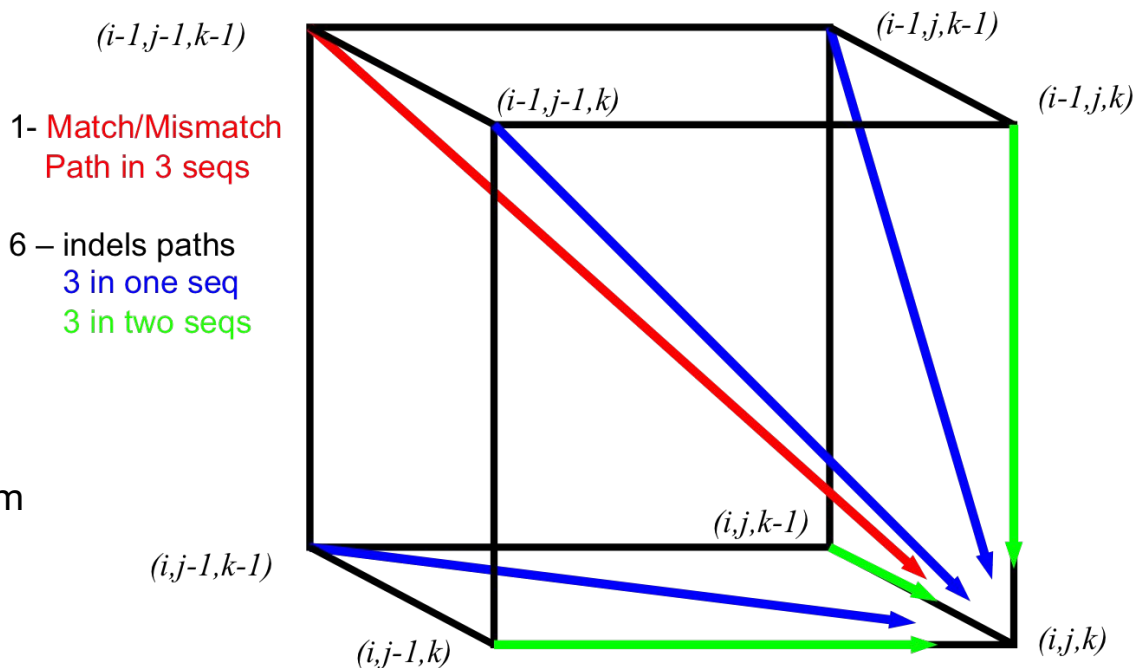
- 2-D  $[(i-1,j-1), (i-1,j), (i,j-1)] \rightarrow (i,j)$  (3 directions)
- 3-D  $[(i-1,j-1,k-1), (i-1,j,k), (i,j-1,k), (i,j,k-1), (i,j-1,k-1), (i-1,j,k-1), (i-1,j-1,k),] \rightarrow (i,j,k)$  (7 directions)
- N-D ( $2^N - 1$  directions)

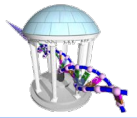
# Structure of a 3-D Alignment Cell



There are three path types

1. Consume a character in all 3 sequences (diagonal "red" edge)
2. Consumes characters in 2 of 3 sequences ("blue" diagonals on faces)
3. Consumes a character from only one sequence ("green" edges)





# Multiple Alignment: Recursion Relation

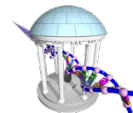
- $s_{i,j,k} = \max \left\{ \begin{array}{l} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \end{array} \right\}$ 
  - cube diagonal: no indels
  - face diagonal: one indel
  - Lattice edge: two indels

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

Scoring matrix has  $5^3$  entries



# Multiple Alignment: Running Time



- For 3 sequences of length  $n$ , the run time is  $7n^3$ ;  $O(n^3)$
- For  $k$  sequences, build a  $k$ -dimensional Manhattan, with run time  $(2^k-1)(n^k)$ ;  $O(2^k n^k)$
- Conclusion: dynamic programming approach for alignment between two sequences is easily extended to  $k$  sequences but it is impractical due to exponential running time

## Example:

To align 6, 100-base sequences, there are 63 directions to consider and  $10^{12}$  cells to compute

Compare to aligning all  $6(5-1)/2 = 15$  pairs, each with 3 directions and 10,000 cells

# Multiple Alignment Induces Pairwise Alignments

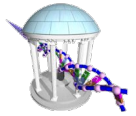


Every multiple alignment induces pairwise alignments

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

Induces:

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



# Inverse Problem

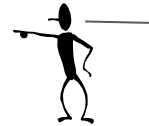
## Do Pairwise Alignments imply a Multiple Alignment?

- Given 3 arbitrary pairwise alignments:

x: ACGCTGGC;	x: AC-GCTGGC;	y: AC-GC-GAG
y: ACGC-GAG;	z: GCCGCAGAG;	z: GCCGCAGAG

- Can we construct a multiple alignment that induces them?

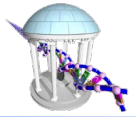
x: AC-GCTGGC
y: AC-GC-GAG
z: GCCGCAGAG



The x/y alignment has an extra "aligned" INDEL in both x and y. The x/z and y/z alignments are unchanged.

SOMETIMES, BUT NOT ALWAYS

- Why? Because pairwise alignments can be arbitrarily inconsistent



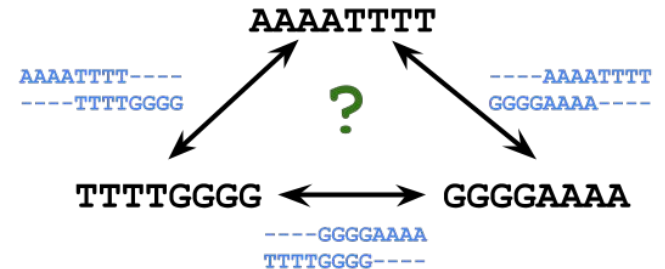
# Combining Optimal Pairwise Alignments

- In some cases we can combine pairwise alignments into a single multiple alignment
- But, in others we cannot because one alignment makes a choice that is inconsistent with the overall best choice

AAAATTTT-----                      -----AAAATTTT-----  
 ----TTTTGGGG-----                      -----TTTTGGGG-----  
 -----GGGAAAA                                      GGGAAAA-----

-OR-

- Is there another way?



# Multiple Alignment from Pairwise Alignments



- From an optimal multiple alignment, we can infer pairwise alignments between all pairs of sequences, but they are not necessarily optimal
- It is difficult to infer a “good” multiple alignment from optimal pairwise alignments between all sequences
- Are we stuck, or is there some other trick?



# Multiple Alignment using a Profile Scores

- We used profile scores like those we used earlier when Motif finding

```
- 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    0  5  0  0  0  0  5  0  0  4  0  0  0  0
C    3  0  0  0  5  0  0  2  5  0  3  1  0  0
G    0  0  5  1  0  0  0  0  0  1  0  0  2  5
T    1  0  0  0  0  5  0  3  0  0  0  0  1  0
-    1  0  0  4  0  0  0  0  0  0  2  4  2  0
```

- Thus far we have aligned sequences against other sequences
- Can we align a sequence against a profile?
- Can we align a profile against a profile?

# Aligning Alignments



A more general version of the multi-alignment problem:

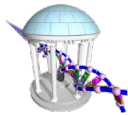
- Given two alignments, can we align them?

```
x: GGGCACTGCAT
y: GGTTACGTC--      Alignment 1
z: GGGA ACTGCAG
```

```
w: GGACGTACC--      Alignment 2
v: GGACCT-----
```

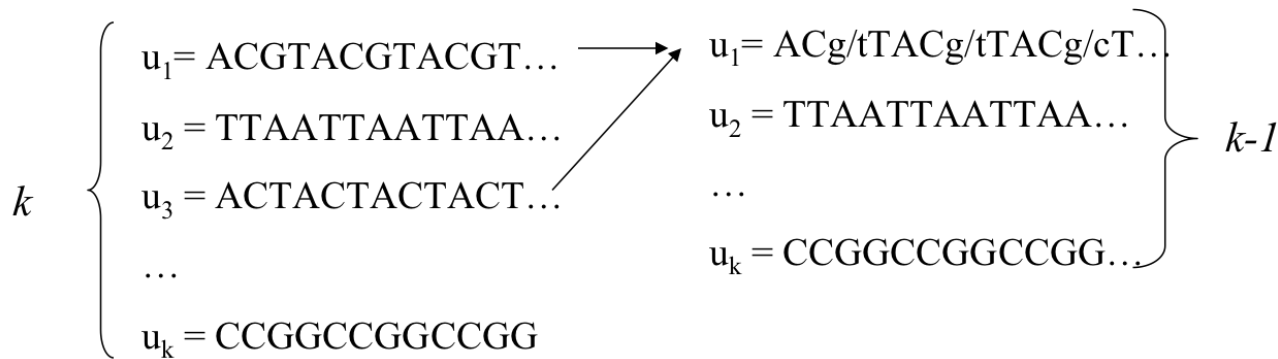
- Idea: don't use the sequences, but align their profiles

```
x: GGGCAC=TGCAT
y: GGTTAC=GTC--
z: GGGAAC=TGCAG      Combined Alignment
  ||  ||  |  |
w: GG==ACGTACC--
v: GG==ACCT-----
```



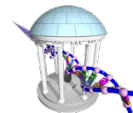
# Profile-Based Multiple Alignment: A Greedy Approach

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





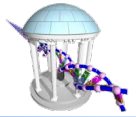
# Example



- Consider these 4 sequences

s1: GATTCA  
s2: GTCTGA  
s3: GATATT  
s4: GTCAGC

- with the scoring matrix: {Match = 1, Mismatch = -1, Indel = -1}



# Example (continued)

- There are  $4 \text{ choose } 2 = 6$  possible pairwise alignments

$s_2$ : GTCTGA

$s_1$ : GATTCA--

$s_4$ : GTCAGC (score = 2)

$s_4$ : G-T-CAGC (score = 0)

$s_1$ : GAT-TCA

$s_2$ : G-TCTGA

$s_2$ : G-TCTGA (score = 1)

$s_3$ : GATAT-T (score = -1)

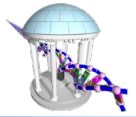
$s_1$ : GAT-TCA

$s_3$ : GAT-ATT

$s_3$ : GATAT-T (score = 1)

$s_4$ : G-TCAGC (score = -1)

- The best pairwise score, 2, is between  $s_2$  and  $s_4$



# Example (continued)

- Combine s2 and s4:

```

s2:  G T C T G A
      | | | |
s4:  G T C A G C
  
```

→

```

s2,4:  G T C t/a G a/c
  
```

- Giving a set of three sequences:

```

s1  :   G A T T C A
s3  :   G A T A T T
s2,4:   G T C t/a G a/c
  
```

- Repeat** for 3 choose 2 = 3 possible pairwise alignments

```

s1  : GAT-TCA
s3  : GATAT-T (score = 1 + 1 + 1 - 1 + 1 - 1 - 1 = 1)
s1  : GAT-TCA
s2,4: G-TCtGa (score = 2 - 2 + 2 - 2 + ½ - 1 + ½ = 0)
s3  : GATAT-T
s2,4: G-TCtGa (score = 2 - 2 + 2 - 2 + ½ - 1 - 1 = -1½)
  
```



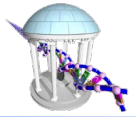
This is the best pairwise score with consensus:

s1,3: G A T -/a T -/c a/t

Next we'd align:

```

s1,3: G A T -/a T -/c a/t
s2,4: G T C t/a G a/c
  
```



# Next Time

- Dynamic Programming saved time when considering all alignments, but at a cost of space (i.e. align 2 sequences each 10,000 bp long)
- Alignment algorithm strategies that save space
- Divide and Conquer

