CSE8803/CX4803 Machine Learning in Computational Biology

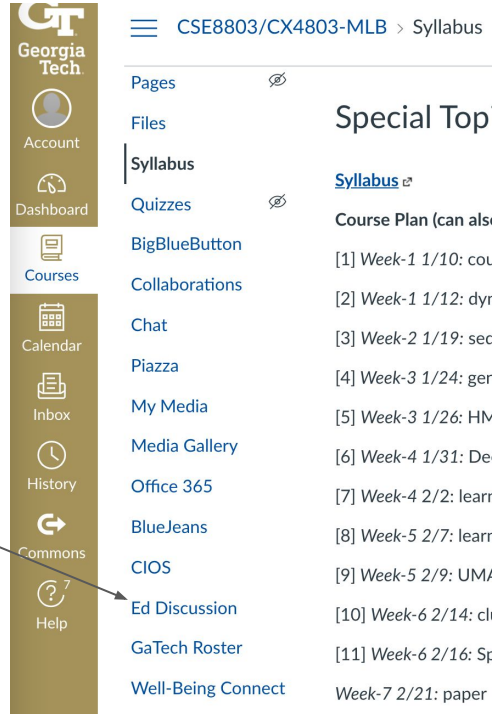
Lecture 2: Sequence Alignment I

Xiuwei Zhang

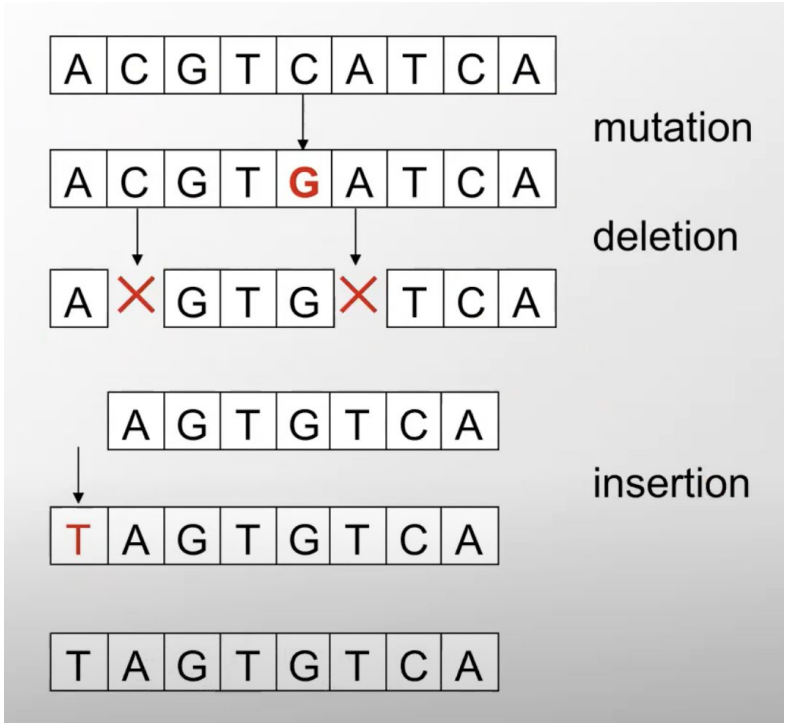
School of Computational Science and Engineering

Course logistics

- Use Ed for discussions instead of Piazza
  - https://edstem.org/us/courses/18196/discussion/
  - (can also be accessed from Canvas)
- Paper presentations: in-person
- Final exam: take-home



Genome changes over time



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

Image credit: Manolis Kellis

Why compare DNA or protein sequences?

```
H. sapiens        -EDSSDS-ENAEPDLDDNE DEEEPAVEIEPEPE----------PQPVTPA
P. troglodytes    -EDSSDS-ENAEPDLDDNE DEEEPAVEIEPEPE----------PQPVTPA
C. lupus          -EDSSDS-ENAEPDLDDNE DEEEPAVEIEPEPE----------PQPVTPA
B. taurus         -EDSSDS-ENAEPDLDDNE DEEEPAVEIEPEPE----------PQPVTPA
M. musculus       -EDSSDSEENAEPDLDDNE EEEEPAVEIEPEPE--PQPQPPPP PQPVAPA
R. norvegicus     -EDSSDS-ENAEPDLDDNE EEEEPAVEIEPEPE PQPQPQPQPQ PQPVAPA
G. gallus         -EDSSDSEENAEPDLDDNE DEEETAVEIEAEPE---------- VSAEAPA
```

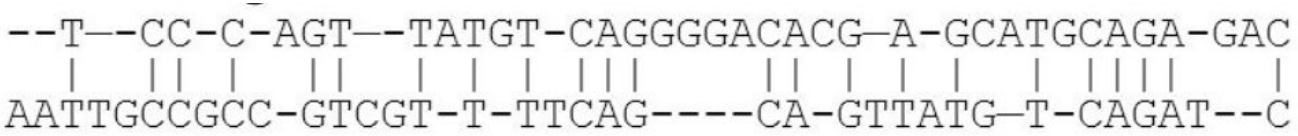
- Understand evolutionary relationships and distances.
- Identify important sequences by finding conserved regions (across species).
- Find genes similar to known genes.
- Provide hints about protein structure and function.
- Map short reads to the genome and quantify gene-expression.

Global alignment vs local alignment

- Global alignment
  - Aims at aligning the entire sequences, matching as many characters as possible, from start to end
  - Sequences that are reasonably similar and have similar length
  - Needleman–Wunsch algorithm
- Local alignment
  - The sequenced may be “stretched” to align local regions that are highly similar
  - Suitable for sequences that are very similar in some parts but not similar in some other parts
  - Smith-Waterman algorithm

Global alignment vs local alignment

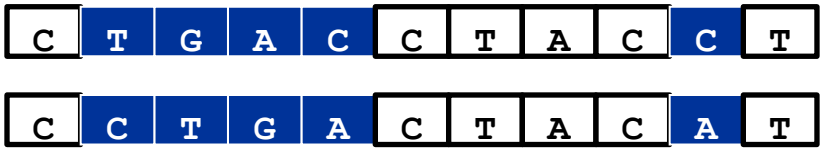
- Global alignment



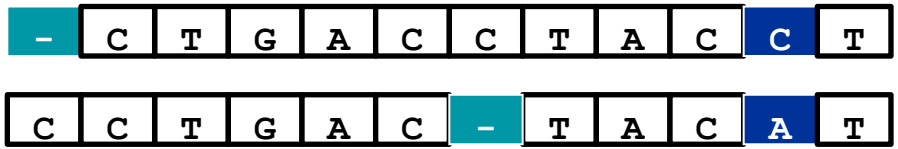
- Local alignment

Global alignment

- Goal: minimize a scoring function that penalizes insertions, deletions and mutations.
- If the penalty is the same for each *operation* -- find the least number of operations
- Most of the time the penalty is different for each operation.
- Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty δ; mismatch penalty α_{pq}.
  - Cost = sum of gap and mismatch penalties.



$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$



$2\delta + \alpha_{CA}$

Sequence alignment

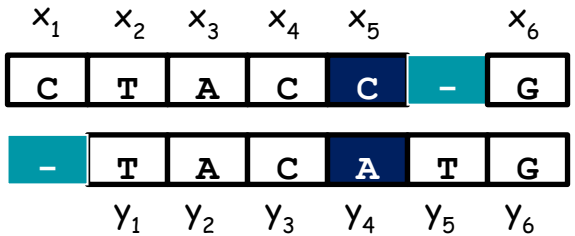
Goal. Given two strings $x_1\, x_2 \ldots x_m$ and $y_1\, y_2 \ldots y_n$ find a min-cost alignment.

Def. An alignment M is a set of ordered pairs $x_i - y_j$ such that each item occurs in at most one pair and no crossings.

$x_i - y_j$ and $x_{i'} - y_{j'}$ cross if $i < i'$, but $j > j'$

Def. The cost of an alignment M is:

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:\ x_i \text{ unmatched}} \delta + \sum_{j:\ y_j \text{ unmatched}} \delta}_{\text{gaps}}$$



an alignment of CTACCG and TACATG:

$M = \{\, x_2\text{–}y_1,\ x_3\text{–}y_2,\ x_4\text{–}y_3,\ x_5\text{–}y_4,\ x_6\text{–}y_6 \,\}$

Can we check all possible alignments?

$$x = x_1 x_2 x_3 x_4 \ldots x_m$$
$$y = y_1 y_2 y_3 y_4 \ldots y_n$$

Brute-force solution: enumerate all the possible alignments, score each alignment, and select the alignment with the maximal score.

How many possible alignments are there?

$x_1 x_2 x_3 x_4 \ldots x_m$ ---...--- (length: m+n)
---...--- $y_1 y_2 y_3 y_4 \ldots y_n$ (length: m+n)

For the *x* sequence, *m+1* choices to put the first gap, *m+1* choices to put the 2nd gap, ...
For the *y* sequence, *n+1* choices to put the first gap, ...
The total number of alignments grows exponentially with the length of sequences.

Algorithm for Computing Edit Distance

Consider the last characters of each string:

$$x = x_1 x_2 x_3 x_4 \ldots x_m$$
$$y = y_1 y_2 y_3 y_4 \ldots y_n$$

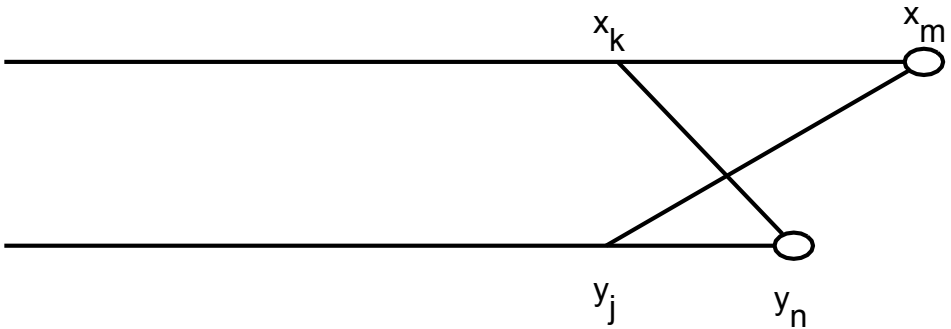
One of these possibilities must hold:

1. (x_m, y_n) are matched to each other
2. x_m is not matched (*i.e.* aligned with a gap)
3. y_n is not matched (*i.e.* aligned with a gap)
4. x_m is matched to some y_j $(j \neq n)$ and y_n is matched to some x_k $(k \neq m)$.

#4 can't happen! Why?

No Crossing Rule Forbids #4

4. x_m is matched to some y_j $(j \neq n)$ and y_n is matched to some x_k $(k \neq m)$.



So, the only possibilities for what happens to the last characters are:

1. (x_m, y_n) are matched to each other
2. x_m is not matched
3. y_n is not matched

Needleman-Wunsch algorithm

Dynamic programming. Break up a problem into a series of ***overlapping*** sub-problems, and build up solutions to larger sub-problems from smaller subproblems, (*reusing* solutions of encountered subproblems as much as possible).

Dynamic Programming

1) Show problem has optimal substructure: the optimal solution can be constructed from optimal solutions to subproblems (recurrence relation).

2) Show subproblems are overlapping, i.e., subproblems may be encountered many times but the total number of distinct subproblems is polynomial

3) Construct an algorithm that computes the optimal solution to each subproblem only once, and reuses the stored result all other times

4) Show that time and space complexity is polynomial

Systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency).

Sequence alignment: problem structure

Def. $OPT(i, j)$ = min cost of aligning prefix strings $x_1\ x_2\ \ldots\ x_i$ and $y_1\ y_2\ \ldots\ y_j$.

Goal. $OPT(m, n)$.

Case 1. $OPT(i, j)$ matches/mismatches $x_i - y_j$.

- Pay mismatch for $x_i - y_j$ + min cost of aligning $x_1\ x_2\ \ldots\ x_{i-1}$ and $y_1\ y_2\ \ldots\ y_{j-1}$.

Case 2a. $OPT(i, j)$ leaves x_i unmatched.

- Pay gap for x_i + min cost of aligning $x_1\ x_2\ \ldots\ x_{i-1}$ and $y_1\ y_2\ \ldots\ y_j$.

Case 2b. $OPT(i, j)$ leaves y_j unmatched.

- Pay gap for y_j + min cost of aligning $x_1\ x_2\ \ldots\ x_i$ and $y_1\ y_2\ \ldots\ y_{j-1}$.

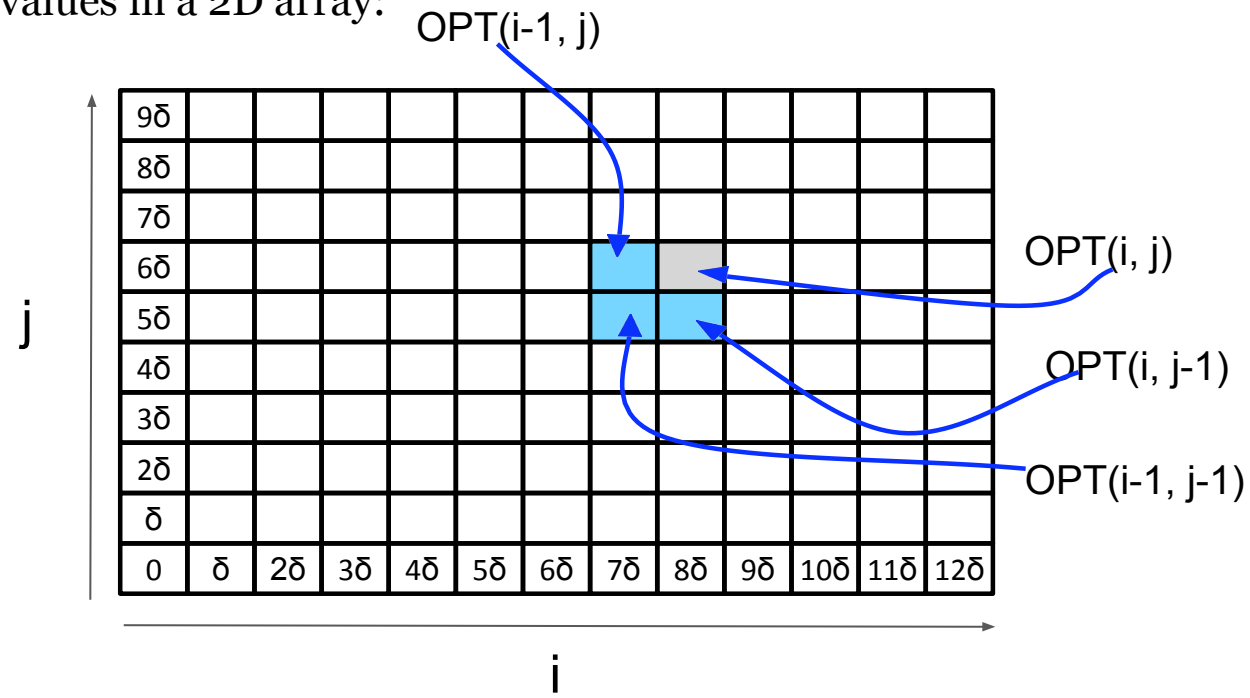
$$OPT(i,j) = \min \begin{cases} \text{cost}(a_i, b_j) + OPT(i-1, j-1) \\ \text{gap} + OPT(i-1, j) \\ \text{gap} + OPT(i, j-1) \end{cases}$$

$$OPT(i,0) = i \times \text{gap} \text{ and } OPT(0,j) = j \times \text{gap}$$

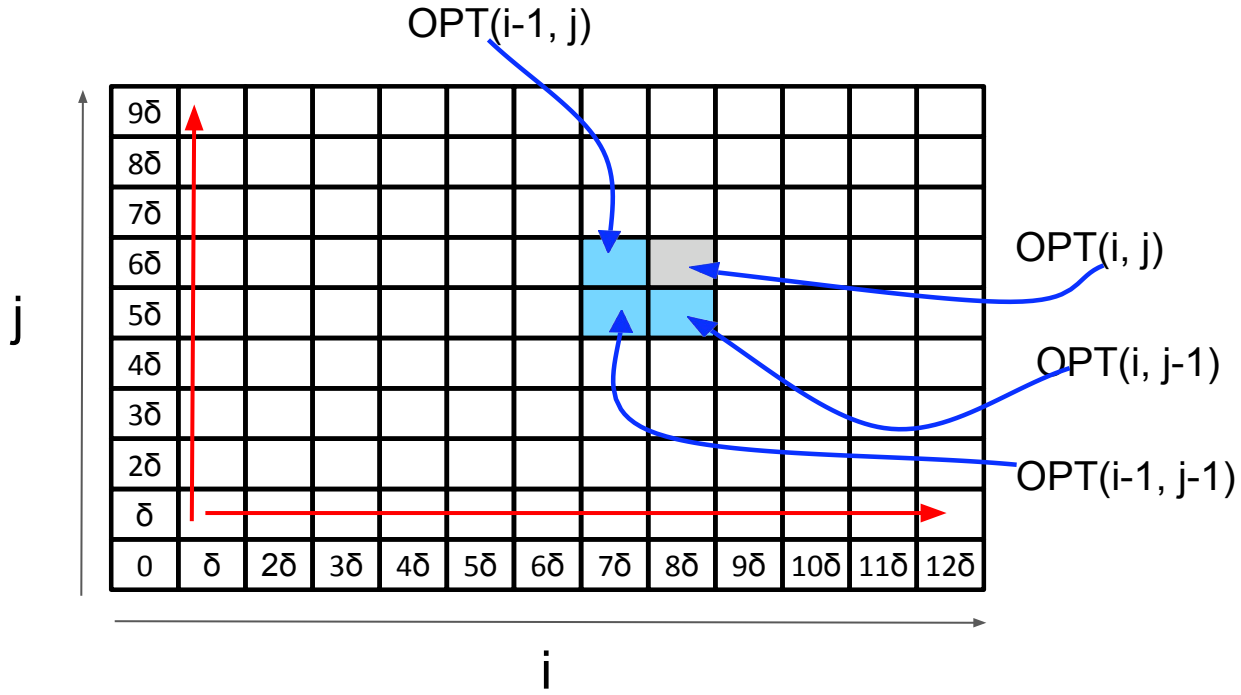
Computing OPT(i,j) Efficiently

We're ultimately interested in $OPT(n,m)$, but we will compute all other $OPT(i,j)$ ($i \le n, j \le m$) on the way to computing $OPT(n,m)$.

Store those values in a 2D array:



In what order do we fill in this array?



Running Time

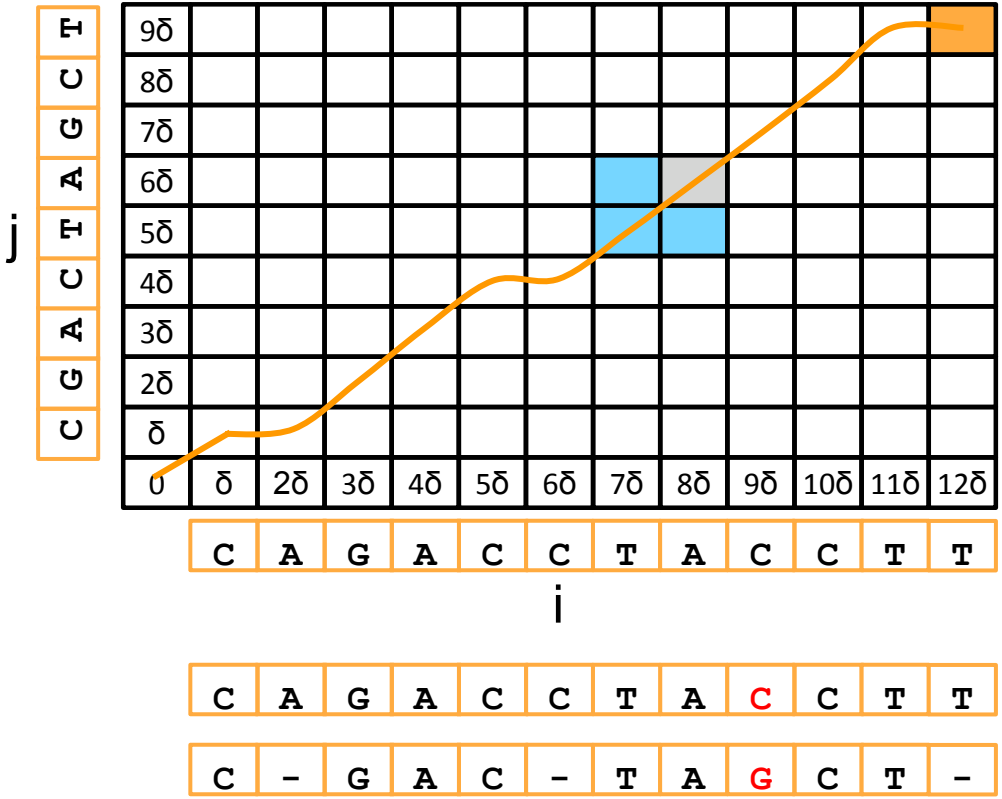
Number of entries in array = $O(m \times n)$, where m and n are the lengths of the 2 strings.

Filling in each entry takes constant $O(1)$ time.

Total running time is $O(mn)$.

Other ways to optimize time and space.

Finding the actual alignment



Outputting the Alignment

Build the alignment from right to left.

Follow the backtrack pointers starting from entry (n,m).

- If you follow a diagonal pointer, add both characters to the alignment,
- If you follow a left pointer, add a gap to the y-axis string and add the x- axis character
- If you follow a down pointer, add the y-axis character and add a gap to the x-axis string.
- There may be multiple optimal alignments.

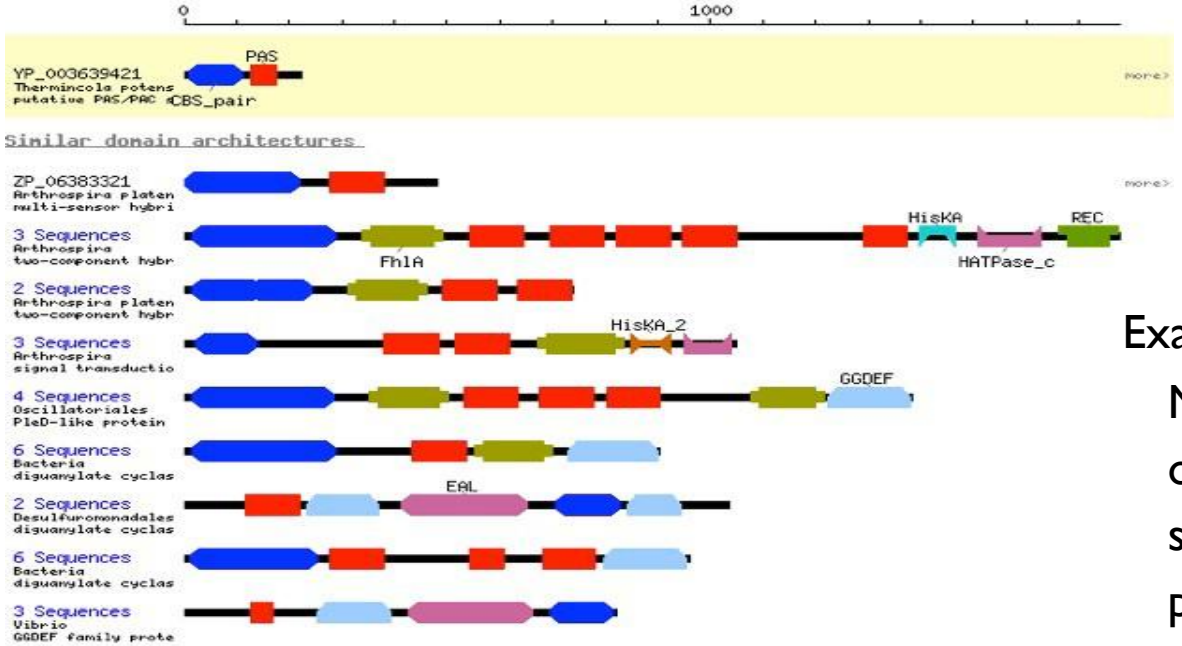
Sequence Alignment: bottom-up algorithm

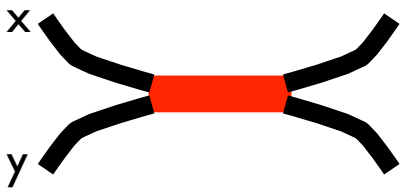
```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
  for i = 0 to m
    M[i, 0] = iδ
  for j = 0 to n
    M[0, j] = jδ

  for i = 1 to m
    for j = 1 to n
      M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                    δ + M[i-1, j],
                    δ + M[i, j-1])
  return M[m, n]
}
```

Local Alignment

Local alignment between x and y: Best alignment between a subsequence of x and a subsequence of y.





Example:

Many genes are composed of *domains*, which are subsequences that perform a particular function.

Local alignment

- **Local** alignment is much more common than **global** alignment
  - Example: aligning two protein sequences that have a common domain but are otherwise different
  - Mapping short reads to the genome
- Compared to **global** alignment, the **local** alignment problem appears to be significantly more complex
- **Naïve approach**:
  - Given that we know how to compute the global alignment between two sequences in O(*mn*) time
  - We can take all possible combinations of substrings of ***x*** and substrings of ***y***
  - How many all possible combinations of substrings?

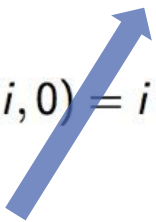
The running time will be $O(m^3n^3)$

Maximization vs. Minimization

Global alignment:

$$OPT(i,j) = \min \begin{cases} \text{cost}(a_i, b_j) + OPT(i-1, j-1) \\ \text{gap} + OPT(i-1, j) \\ \text{gap} + OPT(i, j-1) \end{cases}$$

$$OPT(i,0) = i \times \text{gap} \text{ and } OPT(0,j) = j \times \text{gap}$$



Sequence Similarity: replace *min* with a *max* and *negate* the parameters.

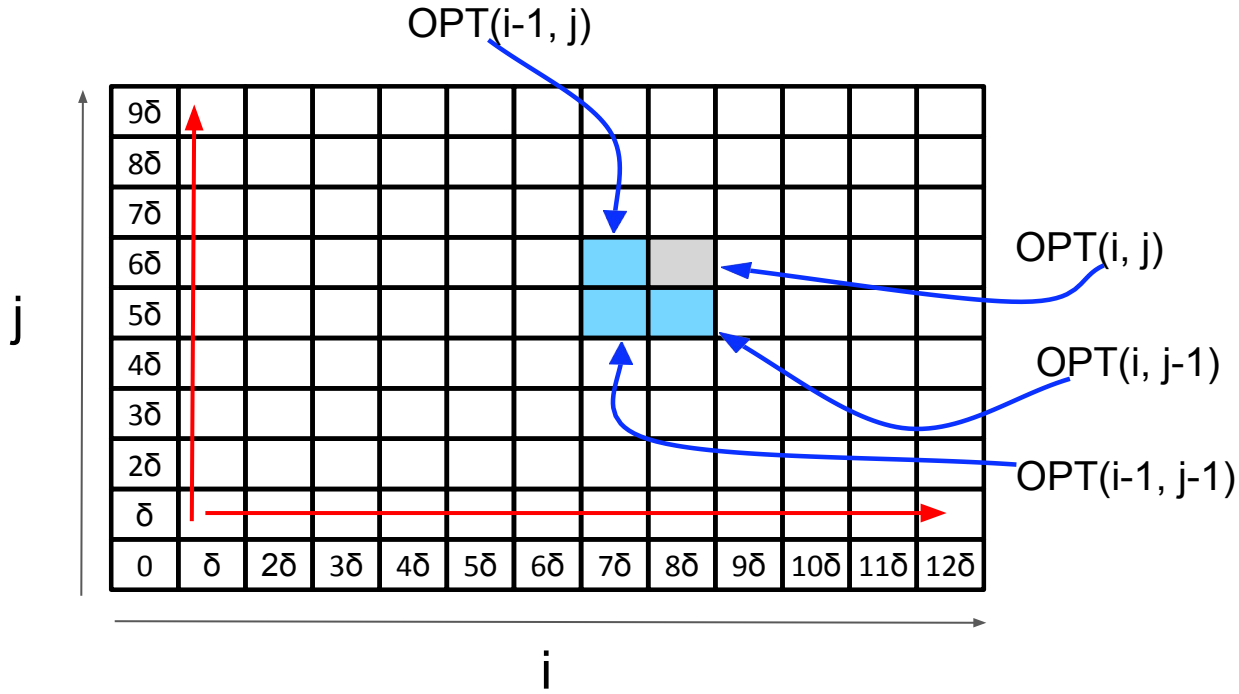
gap penalty → gap benefit (probably negative)
cost → score
Minimization → maximization

Recall: Global Alignment Matrix

OPT(*i*,*j*) contains the score for the best alignment between:

the first *i* characters of string *x* [prefix *i* of *x*]

the first *j* character of string *y* [prefix *j* of *y*]



Local Alignment

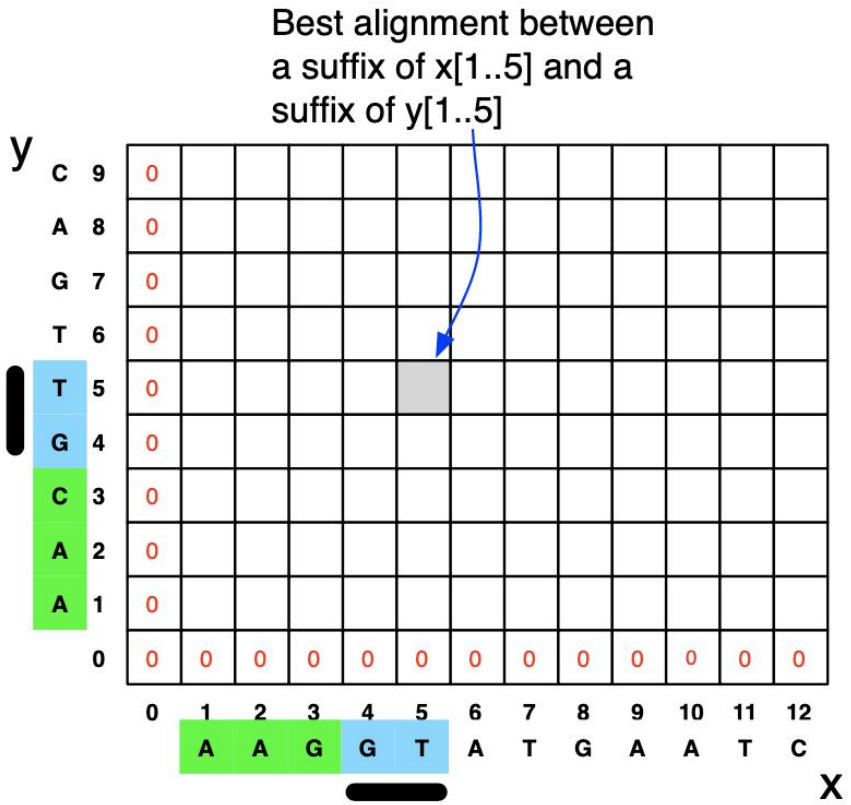
New meaning of entry of matrix entry:

A[i, j] = best score between:

some suffix of *x*[1...*i*]

and

some suffix of *y*[1...*j*]



How do we fill in the local alignment matrix?

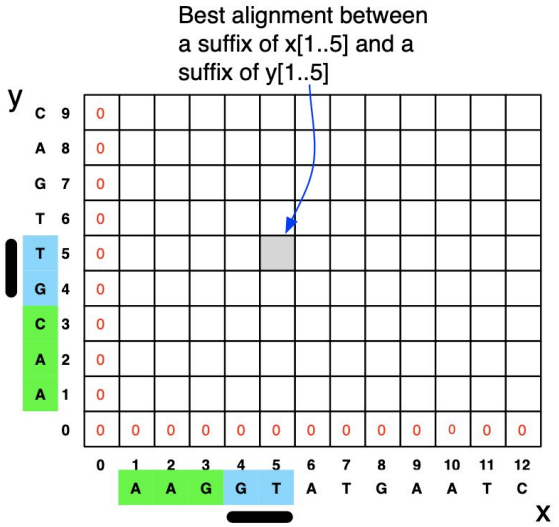
$$A[i, j] = \max \begin{cases} A[i, j-1] + \text{gap} & (1) \\ A[i-1, j] + \text{gap} & (2) \\ A[i-1, j-1] + \text{match}(i, j) & (3) \\ 0 \end{cases}$$

(1), (2), and (3): same cases as before: gap in x, gap in y, match x and y

New case: 0

allows you to say the best alignment between a suffix of *x* and a suffix of *y* is the empty alignment.

Lets us "start over"



Local Alignment

- Initialize first row and first column to be 0.
- The score of the best local alignment is the largest value in the entire array.
- To find the actual local alignment:
  - start at an entry with the maximum score
  - traceback as usual
  - stop when we reach an entry with a score of 0

Local Alignment Example #1

X = AG**CGT**AG

Y = CT**CGT**C

Score(match) = 10

Score(mismatch) = -5

Score(gap) = -7

	*	A	G	C	G	T	A	G
*	0	0	0	0	0	0	0	0
C	0	0	0	10	3	0	0	0
T	0	0	0	3	5	13	6	0
C	0	0	0	10	3	6	8	1
G	0	0	10	3	20	13	6	18
T	0	0	3	5	13	**30**	23	16
C	0	0	0	13	6	23	25	18

Note: this table written top-to-bottom instead of bottom-to-top

Local Alignment Example #2

X = best**oft**imes
Y = s**oft**en

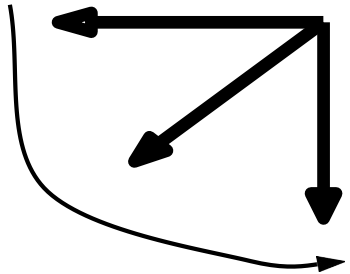
Score(match) = 10
Score(mismatch)= -5
Score(gap) = -7

	*	b	e	s	t	o	f	t	i	m	e	s
*	0	0	0	0	0	0	0	0	0	0	0	0
s	0	0	0	10	3	0	0	0	0	0	0	10
o	0	0	0	3	5	13	6	0	0	0	0	3
f	0	0	0	0	0	6	23	16	9	2	0	0
t	0	0	0	0	10	3	16	33	26	19	12	5
e	0	0	10	3	3	5	9	26	28	21	29	22
n	0	0	3	5	0	0	2	19	21	23	22	24

Note: this table written top-to-bottom instead of bottom-to-top

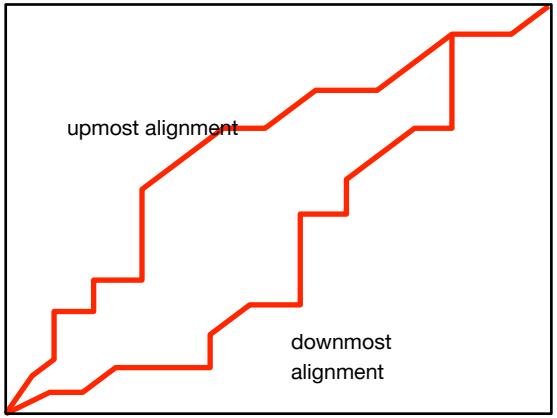
Upmost and Downmost Alignments

When there are ties in the max{}, we have a choice about which arrow to follow.



If we prefer arrows higher in the matrix, we get the *upmost* alignment.

If we prefer arrows lower in the matrix, we get the *downmost* alignment.



Slide credit: Carl Kingsford

Local / Global Recap

- Alignment score sometimes called the “edit distance” between two strings.
- Algorithm for local alignment is sometimes called “Smith-Waterman”
- Algorithm for global alignment is sometimes called “Needleman-Wunsch”
- Same basic algorithm, however.

Further reading

- Global alignment, local alignment, scoring matrix, gap penalty functions:
  Durbin, R., Eddy, S. R., Krogh, A. & Mitchison, G. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. (Cambridge University Press, 1998), Chapter 2.