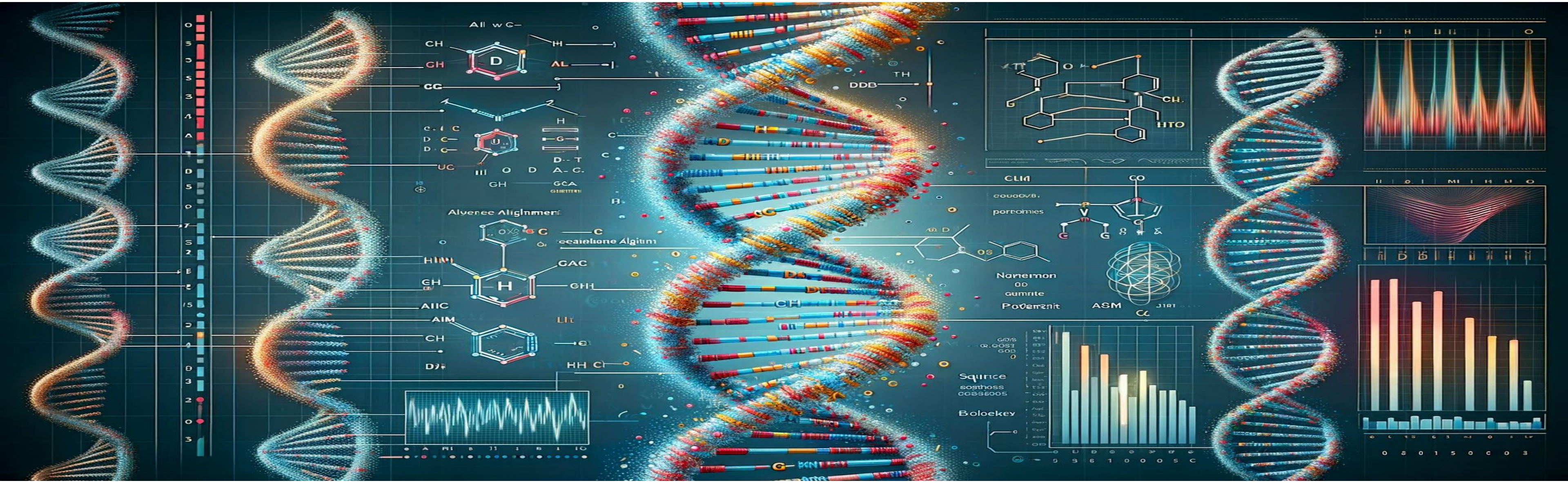
CSE7850/CX4803 Machine Learning in Computational Biology



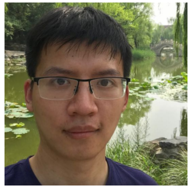
Lecture 4: Sequence Alignment II

Yunan Luo

Acknowledgements: Some of the slides were adapted from lectures of Jian Peng and Mohammed El-Kebir

Announcements: office hours

- Instructor OH
  - **Monday, 2:00 pm - 3:00 pm** on Zoom (no OH this week), or by appointment
  - Divided into 10-minutes slots
  - Visit Canvas to sign up for signing up slots
- TA OH



Fan Qu (fan.qu@gatech.edu)
CS MS student
Office hours: Thu 2-3 pm

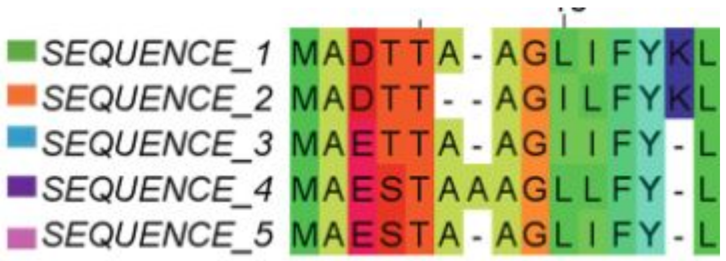
Cheng Wan (cwan39@gatech.edu)
CS PhD student
Office hours: Fri 2-3 pm

- Zoom meeting links available on Canvas

Self-paced PyTorch tutorial learning (01/24)

- **No class this Wednesday (01/24)!**
- Please watch the official PyTorch tutorial videos 1, 2, and 3.
  - After we cover neural networks in class, we will watch video 4, 5, and 6 as well.
- We will release an **exercise** (Colab notebook) on Canvas for you to get familiar with the basic PyTorch data types & operations. Please complete after watching the videos
- Consider joining **TA's OHs** if you have any questions about the PyTorch videos and the exercise.

Sequence alignment


SEQUENCE_1 MADTTA-AGLIFYKL
SEQUENCE_2 MADTT--AGILFYKL
SEQUENCE_3 MAETTA-AGIIFY-L
SEQUENCE_4 MAESTAAAGLLFY-L
SEQUENCE_5 MAESTA-AGLIFY-L

Alignment

An **alignment** between two strings **v** (of *m* characters) and **w** (of *n* characters) is a two row matrix where the first row contains the characters of **v** in order, the second row contains the characters of **w** in order, and spaces may be interspersed throughout each.

Input

v: KITTEN (*m* = 6)

w: SITTING (*n* = 7)

Output

v:	K	–	I	T	T	E	N	–
w:	S	I	–	T	T	I	N	G

Question: Is this a good alignment?

Answer: Count the number of insertion, deletions, substitutions.

Alignment

An **alignment** between two strings **v** (of *m* characters) and **w** (of *n* characters) is a two row matrix where the first row contains the characters of **v** in order, the second row contains the characters of **w** in order, and spaces may be interspersed throughout each.

Input

v: KITTEN $(m = 6)$

w: SITTING $(n = 7)$

Output

v:	K	–	I	T	T	E	N	–
w:	S	I	–	T	T	I	N	G

Question: Is this a good alignment?

Answer: Count the number of insertion, deletions, substitutions.

Recipe

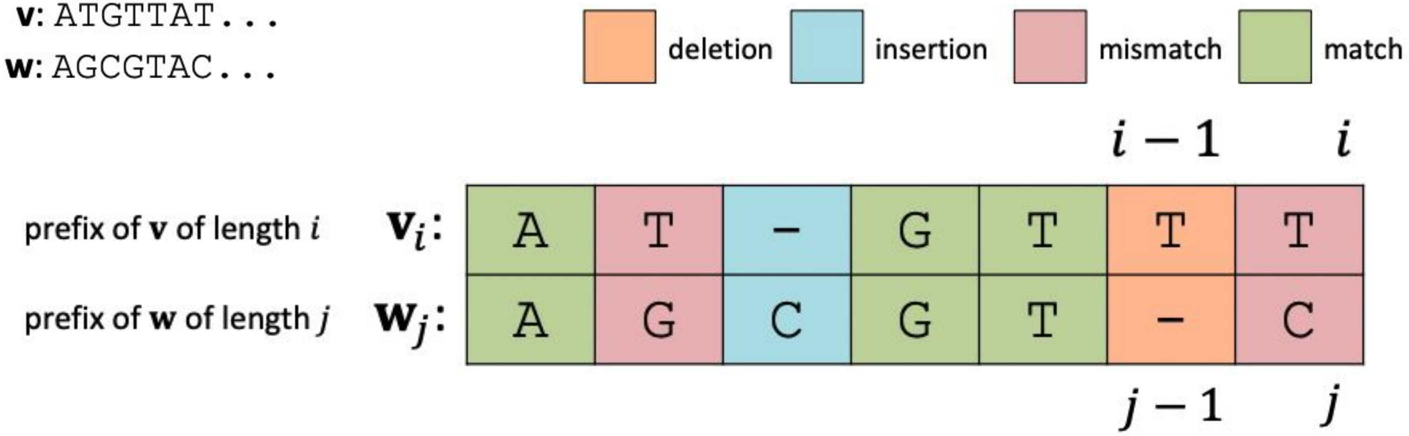
1. Identify subproblems
2. Write down recursions
3. Make it dynamic-programming!

Compute edit distance

Edit Distance problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$, compute the minimum number $d(\mathbf{v}, \mathbf{w})$ of elementary operations to transform $\mathbf{v}$ into $\mathbf{w}$.

v: `ATGTTAT...`

w: `AGCGTAC...`

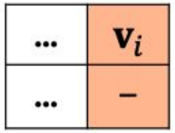


Optimal substructure:
Edit distance obtained from edit distance of prefix of string.

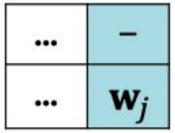
Compute edit distance - Optimal substructure

$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

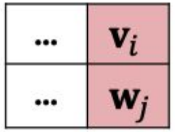
Deletion: $d[i, j] = d[i-1, j] + 1$
Extend by a character in $\mathbf{v}$



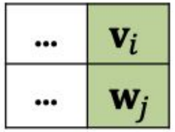
Insertion: $d[i, j] = d[i, j-1] + 1$
Extend by a character in $\mathbf{w}$



Mismatch: $d[i, j] = d[i-1, j-1] + 1$
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$



Match: $d[i, j] = d[i-1, j-1]$
Extend by a character in $\mathbf{v}$ and $\mathbf{w}$



Compute edit distance - Recurrence

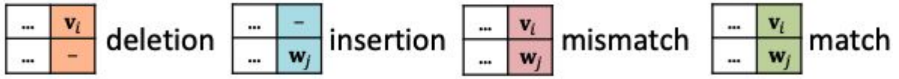
$d[i, j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

Compute edit distance - Dynamic Programming

Example:

w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

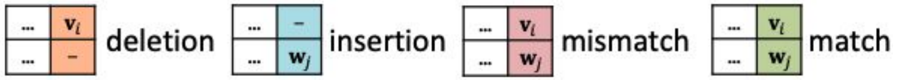
	j	0	1	2	3	4
i	***w*** ***v***		A	T	C	G
0						
1	A					
2	T					
3	G					
4	T					

Compute edit distance - Dynamic Programming

Example:

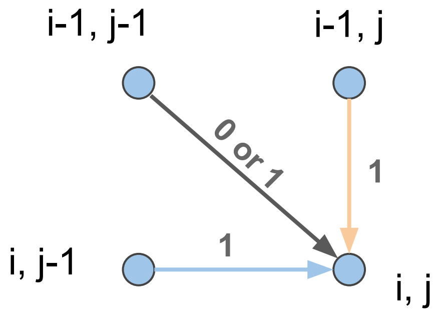
w = ATCG
v = ATGT

	j	0	1	2	3	4
i	*w* *v*		A	T	C	G
0		0				
1	A					
2	T					
3	G					
4	T					



... v_i ... – deletion
... – ... w_j insertion
... v_i ... w_j mismatch
... v_i ... w_j match

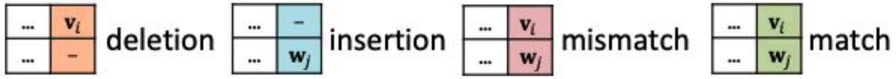
$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



Compute edit distance - Dynamic Programming

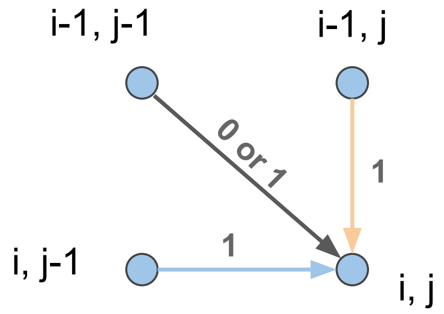
Example:

w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

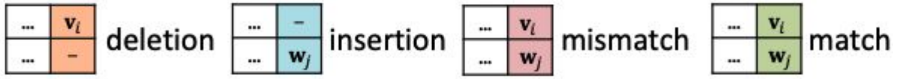
j		0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0		0 →	1 →	2 →	3 →	4
1	A	↓ 1				
2	T	↓ 2				
3	G	↓ 3				
4	T	↓ 4				



Compute edit distance - Dynamic Programming

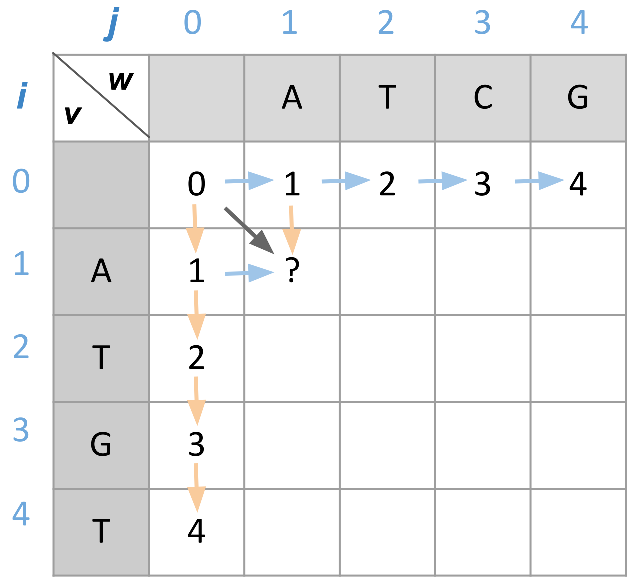
Example:

w = ATCG
v = ATGT

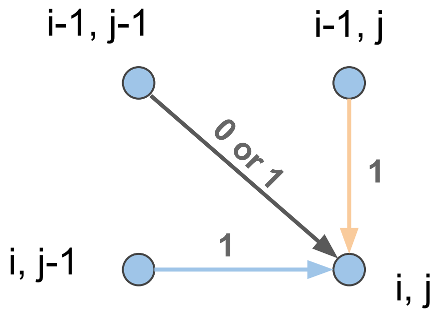


... v_i ... – deletion
... – ... w_j insertion
... v_i ... w_j mismatch
... v_i ... w_j match

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



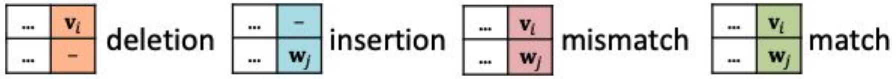
i \ *j*		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	?			
2	T	2				
3	G	3				
4	T	4				



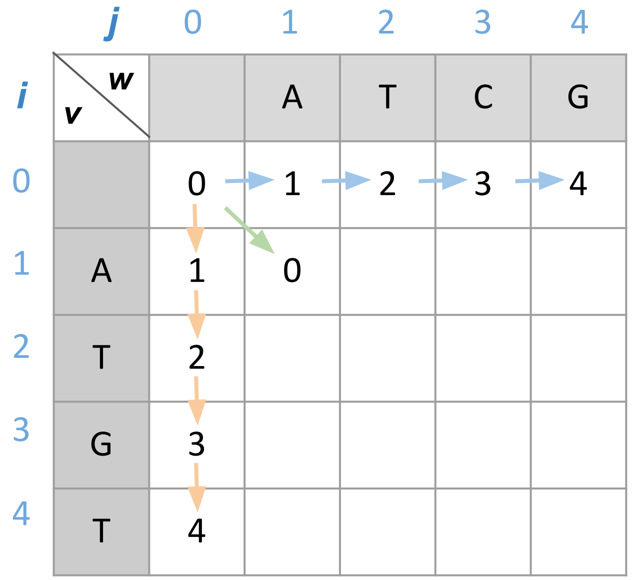
Compute edit distance - Dynamic Programming

Example:

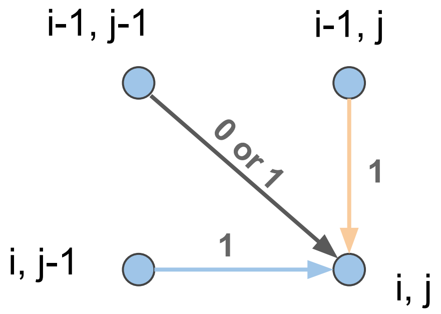
w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



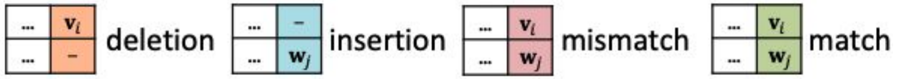
i \ j	v \ w	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2				
3	G	3				
4	T	4				



Compute edit distance - Dynamic Programming

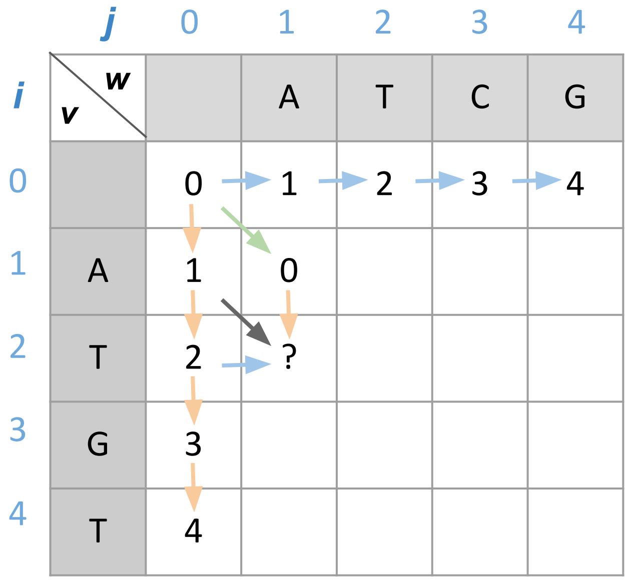
Example:

w = ATCG
v = ATGT

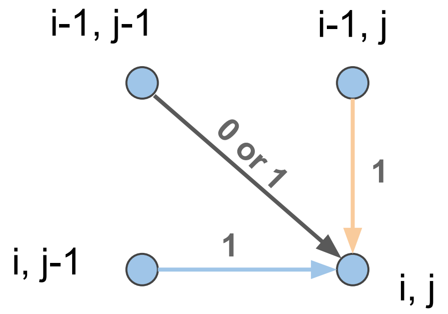


... v_i ... – deletion
... – ... w_j insertion
... v_i ... w_j mismatch
... v_i ... w_j match

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



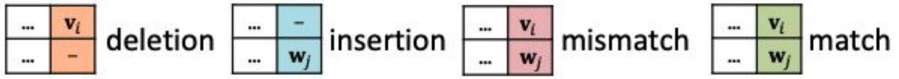
j		0	1	2	3	4
i	***v*** \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2	?			
3	G	3				
4	T	4				



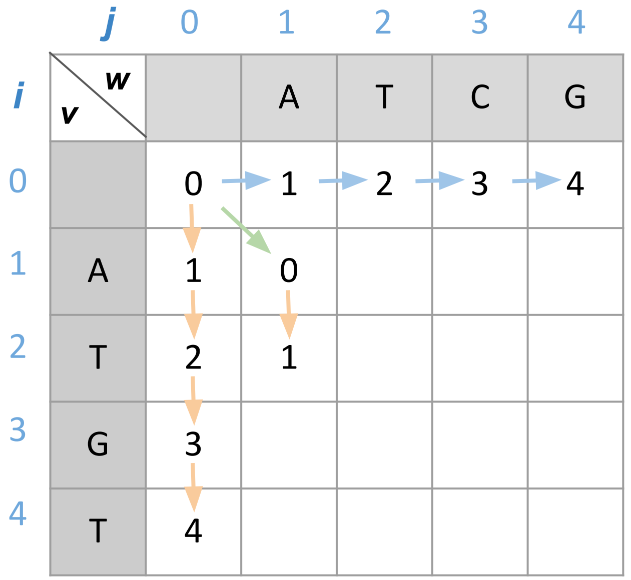
Compute edit distance - Dynamic Programming

Example:

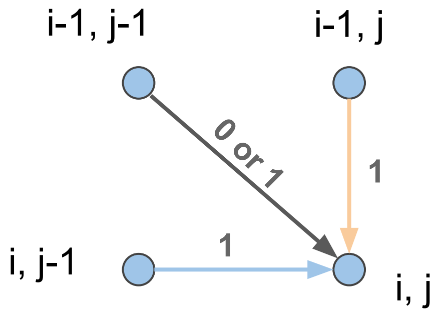
w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



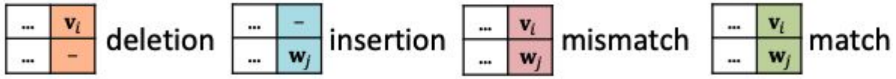
i \ ***j***		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0			
2	T	2	1			
3	G	3				
4	T	4				



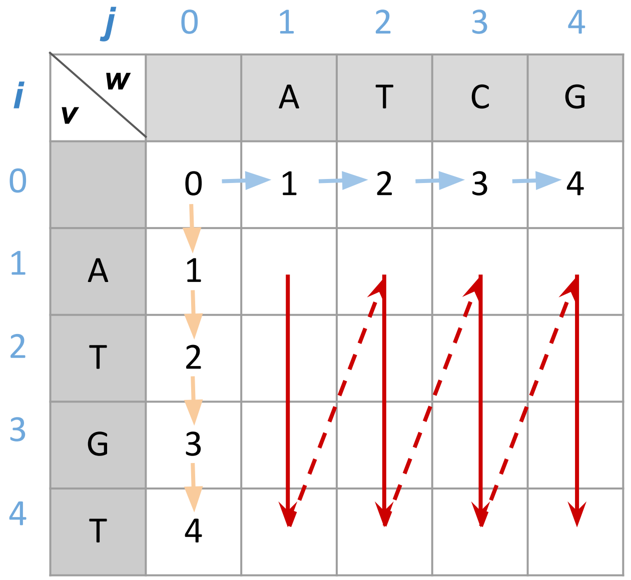
Compute edit distance - Dynamic Programming

Example:

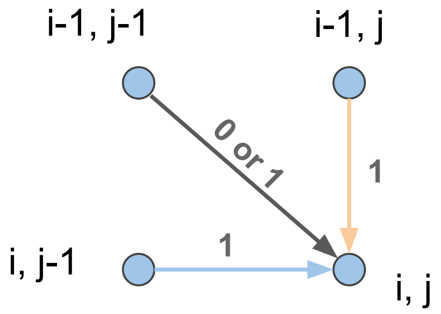
w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



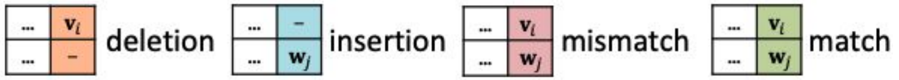
i \ ***j***		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



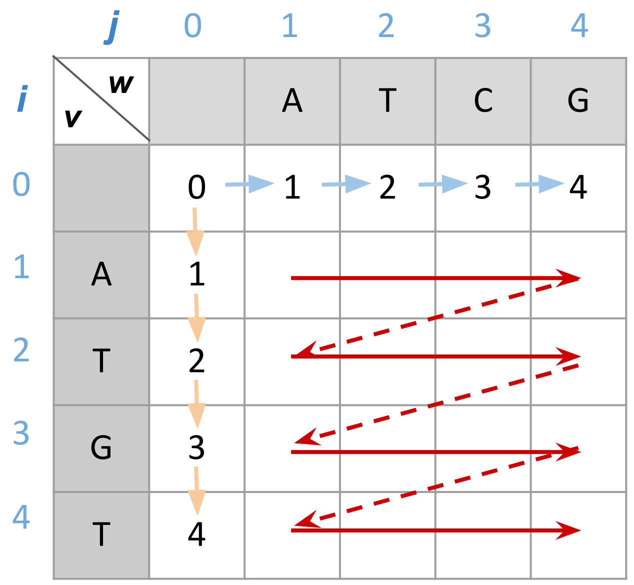
Compute edit distance - Dynamic Programming

Example:

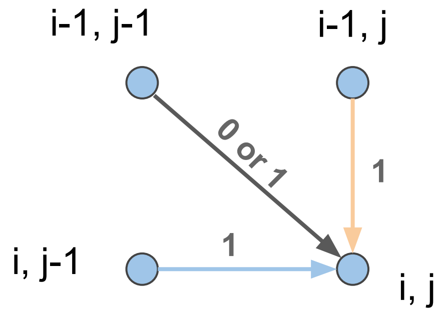
w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



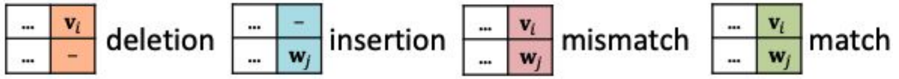
i \ *j*		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



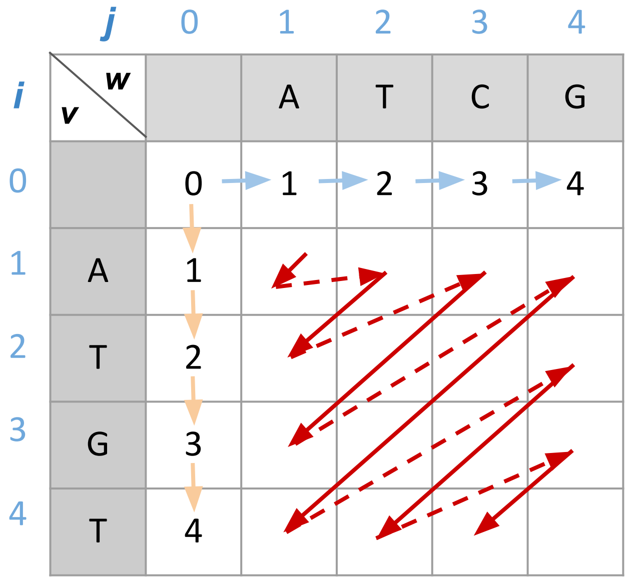
Compute edit distance - Dynamic Programming

Example:

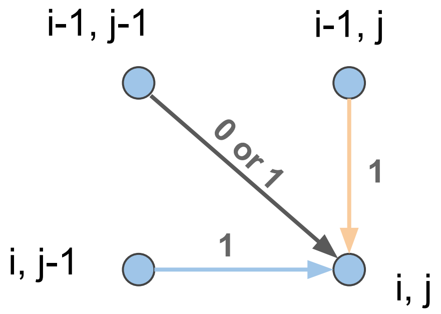
w = ATCG
v = ATGT



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



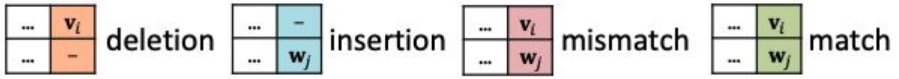
j		0	1	2	3	4
i	*v* \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1				
2	T	2				
3	G	3				
4	T	4				



Compute edit distance - Dynamic Programming

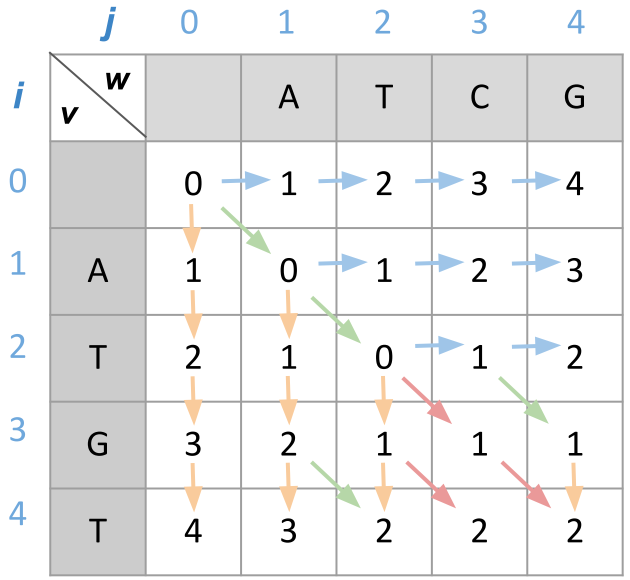
Example:

w = ATCG
v = ATGT

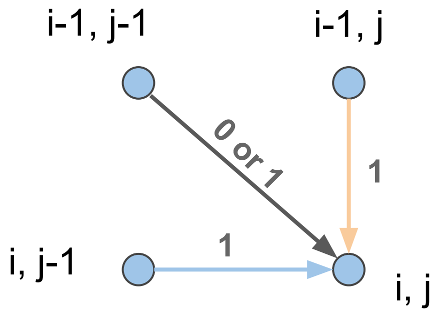


... v_i ... – deletion
... – ... w_j insertion
... v_i ... w_j mismatch
... v_i ... w_j match

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

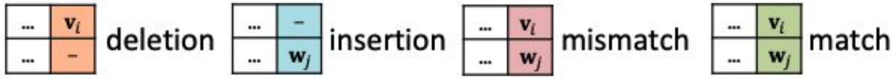


i \ *j*	v \ w	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



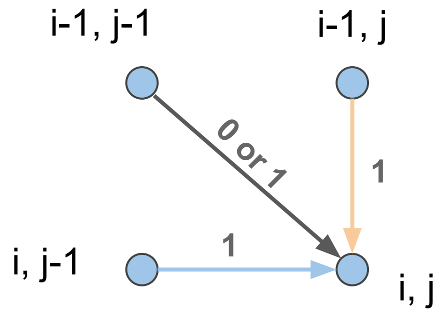
Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?



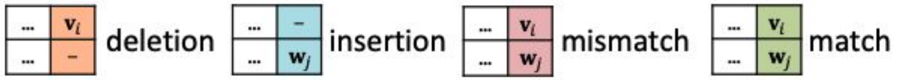
$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ \boxed{d[i-1,j]+1,} & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ \boxed{d[i-1,j-1]+1,} & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



Output the alignment -- Finding optimal path(s)

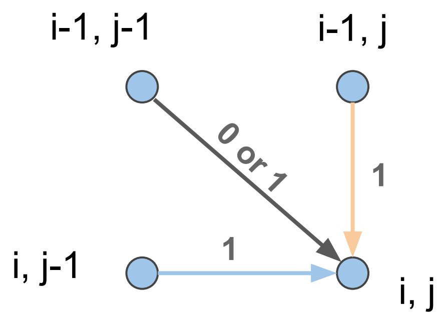
Key idea: where does the score of the current cell come from?



... v_i / ... – deletion
... – / ... w_j insertion
... v_i / ... w_j mismatch
... v_i / ... w_j match

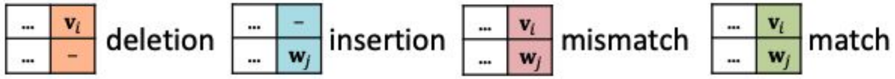
$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + 1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

	j	0	1	2	3	4
i	*w* *v*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



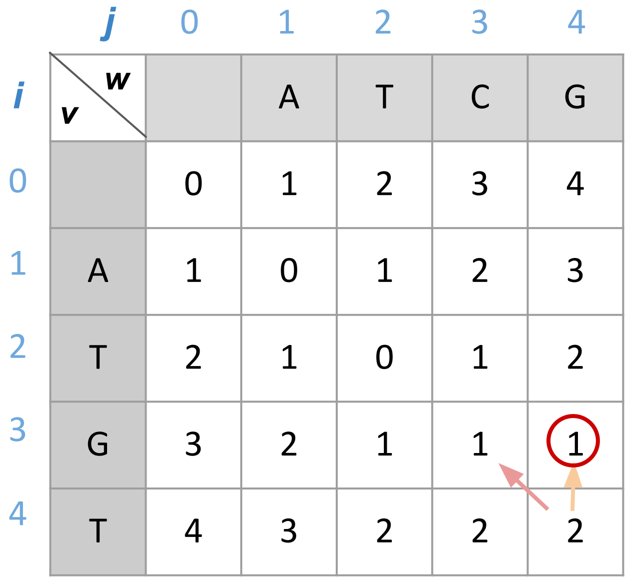
Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?

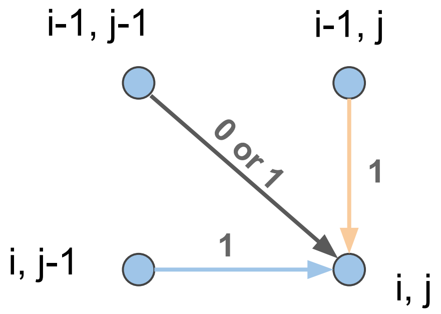


... v_i / ... – deletion
... – / ... w_j insertion
... v_i / ... w_j mismatch
... v_i / ... w_j match

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ \boxed{d[i-1,j-1],} & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



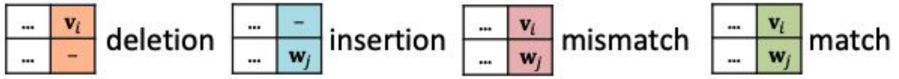
	j	0	1	2	3	4
i	*v* \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



$$\begin{tikzcd} i-1,\ j-1 \arrow[dr, "0 \text{ or } 1"] & i-1,\ j \arrow[d, "1"] \\ i,\ j-1 \arrow[r, "1"] & i,\ j \end{tikzcd}$$

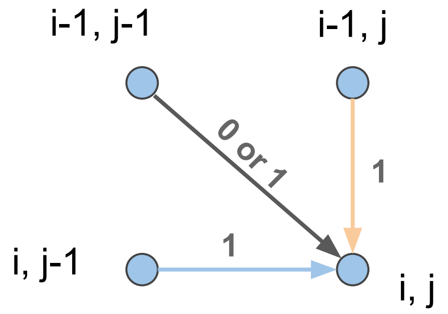
Output the alignment -- Finding optimal path(s)

Key idea: where does the score of the current cell come from?



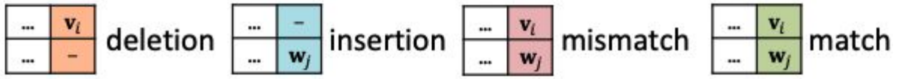
$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ \boxed{d[i-1,j-1],} & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

i \ *j*		0	1	2	3	4
	v \ ***w***		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

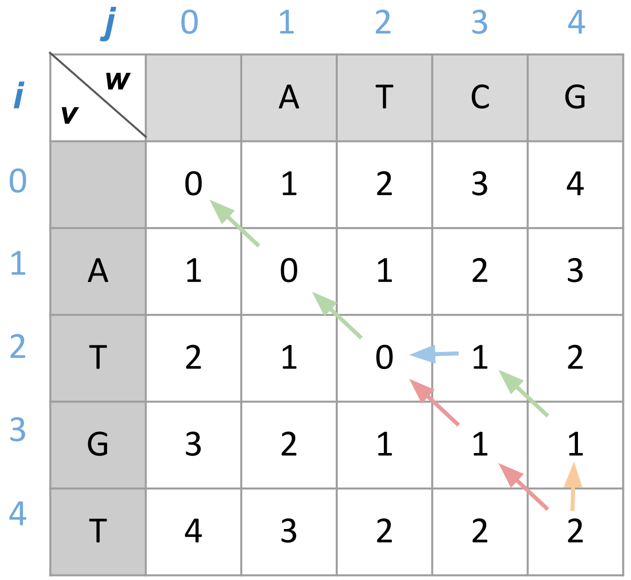


Output the alignment -- Finding optimal path(s)

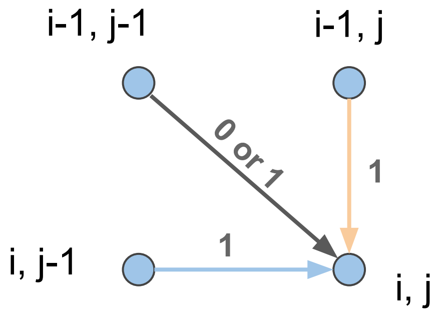
Key idea: where does the score of the current cell come from?



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



i \ *j*	v \ w	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2



Backtrace algorithm

- Base conditions:

  `D(i,0) = i` `D(0,j) = j`

  Termination:

  `D(N,M) is distance`

- Recurrence Relation:

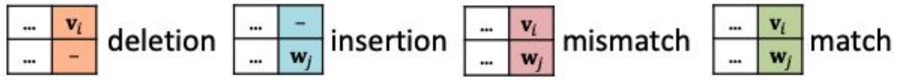
```
For each  i = 1…M
    For each  j = 1…N
```

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } X(i) \neq Y(j) \quad \text{substitution} \\ 0; & \text{if } X(i) = Y(j) \quad \text{match} \end{cases} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{RIGHT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \quad \text{match} \end{cases}$$

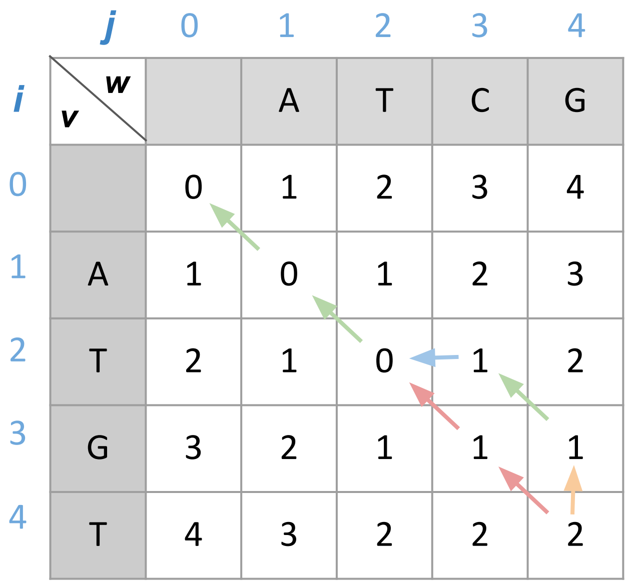
Output the alignment

Key idea: where does the score of the current cell come from?



... v_i ... – deletion
... – ... w_j insertion
... v_i ... w_j mismatch
... v_i ... w_j match

$$
d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}
$$

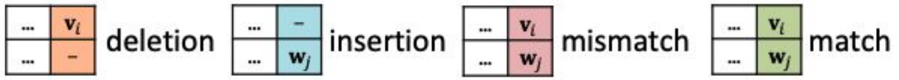


i \ *j*		0	1	2	3	4
	v \ *w*		A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

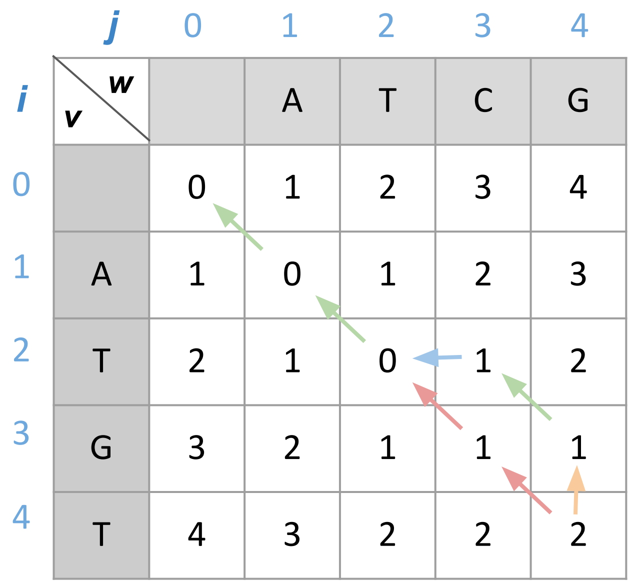
A	T	–	G	T
A	T	C	G	–

A	T	G	T
A	T	C	G

Computing edit distance -- Running time



$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$



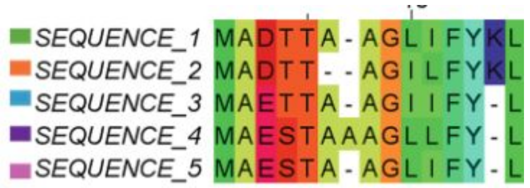
i \ *j*	v \ w	0	1	2	3	4
			A	T	C	G
0		0	1	2	3	4
1	A	1	0	1	2	3
2	T	2	1	0	1	2
3	G	3	2	1	1	1
4	T	4	3	2	2	2

For each $(m+1) \times (n+1)$ entry:
- 3 addition operations
- 1 comparison operation
- 1 minimum operation

Running time: $O(mn)$ time

Summary

- Sequence alignment

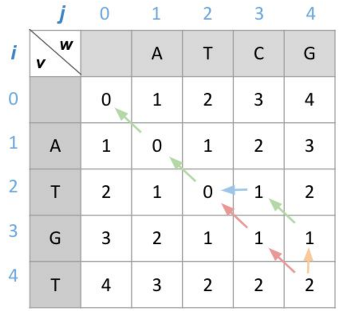


🟩 SEQUENCE_1	MADTTA-AGLIFYKL
🟧 SEQUENCE_2	MADTT--AGILFYKL
🟦 SEQUENCE_3	MAETTA-AGIIFY-L
🟪 SEQUENCE_4	MAESTAAAGLLFY-L
🟪 SEQUENCE_5	MAESTA-AGLIFY-L

- Algorithm for alignment
  - Edit distance
  - Dynamic programming

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

- Find the optimal alignment
  - Backtrace

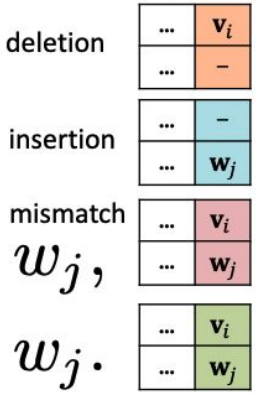


j	0	1	2	3	4
i \ v w		A	T	C	G
0	0	1	2	3	4
1 A	1	0	1	2	3
2 T	2	1	0	1	2
3 G	3	2	1	1	1
4 T	4	3	2	2	2

Is edit distance the best way?

$d[i,j]$ is the edit distance of $\mathbf{v}_i$ and $\mathbf{w}_j$,
where $\mathbf{v}_i$ is prefix of $\mathbf{v}$ of length i and $\mathbf{w}_j$ is prefix of $\mathbf{w}$ of length j

$$d[i,j] = \min \begin{cases} d[i-1,j] + \boxed{1,} \\ d[i,j-1] + \boxed{1,} \\ d[i-1,j-1] + \boxed{1,} & \text{if } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } v_i = w_j. \end{cases}$$



Replace +1 with different penalties for different types of edits.

Weighted edit distance

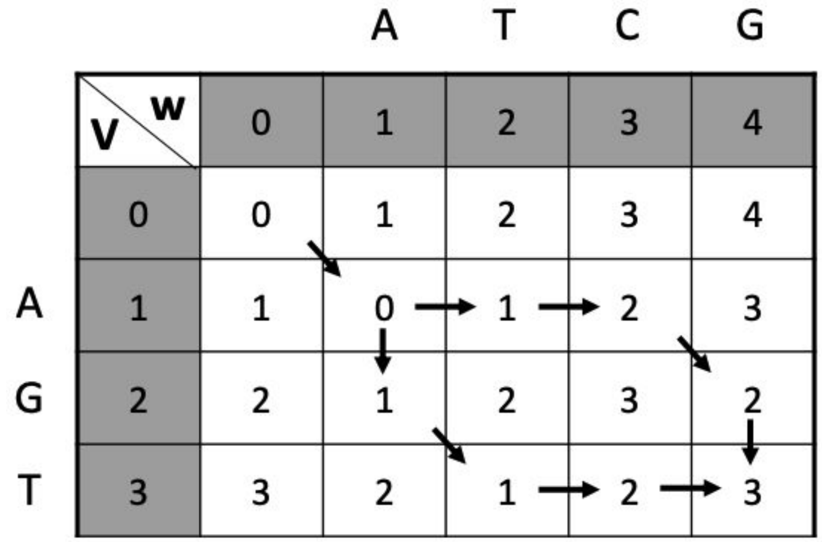
- Compute weighted edit distance between $\mathbf{v}$ = AGT and $\mathbf{w}$ = ATCT.

			A	T	C	G
	V \ W	0	1	2	3	4
	0					
A	1					
G	2					
T	3					

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j] + 1, & \text{if } i > 0, \\ d[i,j-1] + 1, & \text{if } j > 0, \\ d[i-1,j-1] + \boxed{2,} & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

Weighted edit distance

- Compute weighted edit distance between $\mathbf{v} = \mathrm{AGT}$ and $\mathbf{w} = \mathrm{ATCT}$.



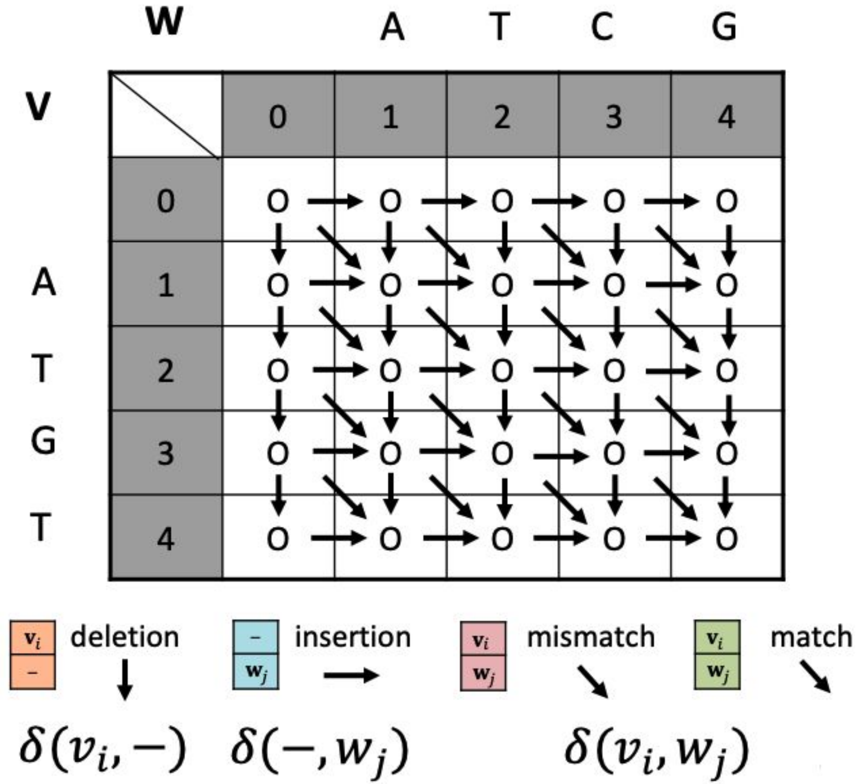
			A	T	C	G
	v \ w	0	1	2	3	4
	0	0	1	2	3	4
A	1	1	0	1	2	3
G	2	2	1	2	3	2
T	3	3	2	1	2	3

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+\boxed{2,} & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

Biological sequence alignment

- Weighted edit distance: find alignment with minimum distance

- Sequence alignment: find alignment with maximum similarity
  - Score function $\delta : (\Sigma \cup \{-\})^2 \to \mathbb{R}$
  - E.g.: δ[i, j] = -d[i,j]



Edit distance vs sequence alignment

- **Edit distance**

$$d[i,j] = \min \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ d[i-1,j]+1, & \text{if } i > 0, \\ d[i,j-1]+1, & \text{if } j > 0, \\ d[i-1,j-1]+1, & \text{if } i > 0,\ j > 0 \text{ and } v_i \neq w_j, \\ d[i-1,j-1], & \text{if } i > 0,\ j > 0 \text{ and } v_i = w_j. \end{cases}$$

- **Sequence alignment**

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j]+\delta(v_i,-), & \text{if } i > 0, & \text{deletion} \\ s[i,j-1]+\delta(-,w_j), & \text{if } j > 0, & \text{insertion} \\ s[i-1,j-1]+\delta(v_i,w_j), & \text{if } i > 0 \text{ and } j > 0. & \text{match/ mismatch} \end{cases}$$

Question: What is an example of δ?

Scoring matrices

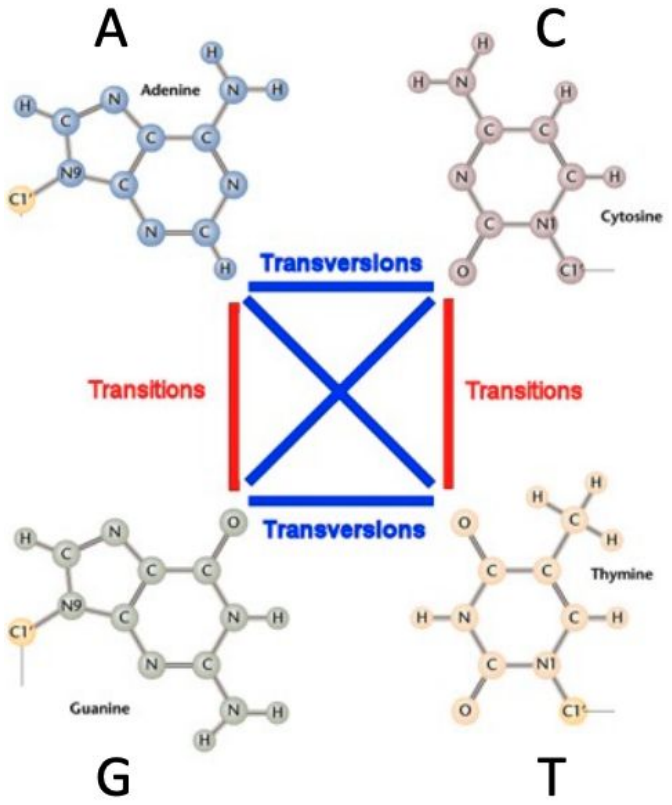
Transitions: interchanges among purines (two rings) or pyrimidines (one ring)

- A <--> G
- C <--> T

Transversions: interchanges between purines (two rings) and pyrimidines (one ring)

- A <--> C, A <--> T
- G <--> C, G <--> T

Transitions more likely than transversions!



Scoring matrices

Transitions: interchanges among purines (two rings) or pyrimidines (one ring)

- A <--> G
- C <--> T

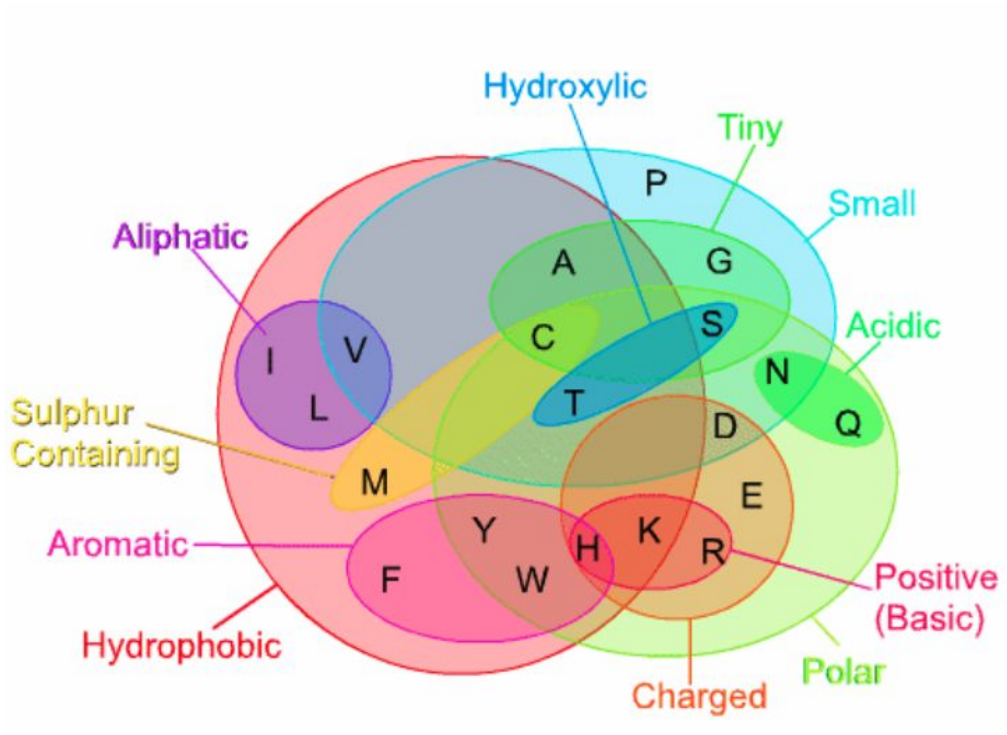
Transversions: interchanges between purines (two rings) and pyrimidines (one ring)

- A <--> C, A <--> T
- G <--> C, G <--> T

Transitions more likely than transversions!

δ	A	T	C	G	-
A	1	-2	-2	-1	-1
T	-2	1	-1	-2	-1
C	-2	-1	1	-2	-1
G	-1	-2	-2	1	-1
-	-1	-1	-1	-1	$-\infty$

Amino acids can share similar properties



<u>Amino Acids</u>

A alanine (ala)
R arginine (arg)
N asparagine (asn)
D aspartic acid (asp)
C cysteine (cys)
Q glutamine (gln)
E glutamic acid (glu)
G glycine (gly)
H histidine (his)
I isoleucine (ile)
L leucine (leu)
K lysine (lys)
M metioneine (met)
F phenyalanine (phe)
P proline (pro)
S serine (ser)
T threonine (thr)
W trytophan (trp)
Y tyrosine (tyr)

BLOcks SUbstitution Matrix (BLOSUM)

Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
	Ala	**Arg**	**Asn**	**Asp**	**Cys**	**Gln**	**Glu**	**Gly**	**His**	**Ile**	**Leu**	**Lys**	**Met**	**Phe**	**Pro**	**Ser**	**Thr**	**Trp**	**Tyr**	**Val**

amino acids

Recursion for generalized edit distance

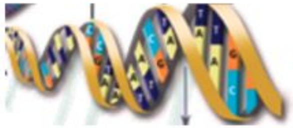
$$
s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, & \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, & \text{deletion} \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, & \text{insertion} \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. & \text{match/ mismatch} \end{cases}
$$

- Known as the **Needleman-Wunsch Algorithm** (1970)
- Used to solve the **global** pairwise alignment problem

Biological sequence alignment problems

- Global alignment ✅
  - Needleman-Wunsch Algorithm
- **Fitting alignment**
- Local alignment

NGS Characterized by Short Reads



Genome
Millions -billions nucleotides

Next-generation DNA sequencing

... CATTCAGTAG ...

... AGCCATTAG ...

... GGTAGTTAG ...

... GGTAAACTAG ...

... TATAATTAG ...

... CGTACCTAG ...

10-100's million ***short reads***
Short read*:* 100 nucleotides

Allow for inexact matches due to:

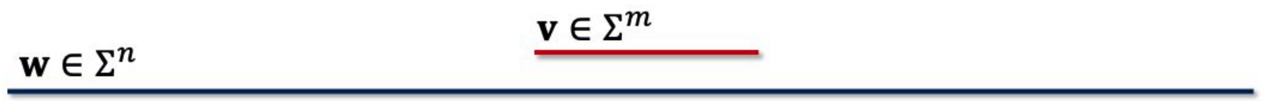
- Sequencing errors
- Polymorphisms/mutations in reference genome

Human reference genome is 3,300,000,000 nucleotides, while a short read is 100 nucleotides. Global sequence alignment will not work!

Question: How to account for discrepancy between lengths of reference and short read?

Fitting alignment

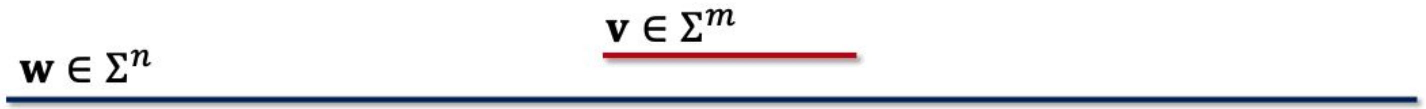
For short read alignment, we want to align complete short read $\mathbf{v} \in \Sigma^m$ to substring of reference genome $\mathbf{w} \in \Sigma^n$. Note that $m \ll n$.



Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$

Fitting Alignment – Naive Approach

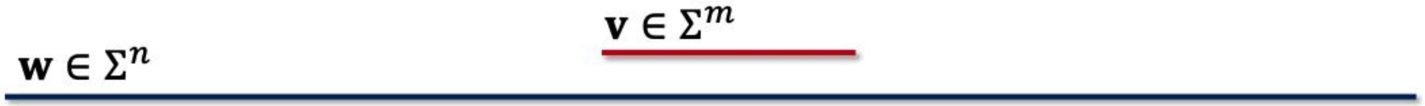
Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$



- Consider all contiguous non-empty substrings of $\mathbf{w}$, defined by $1 \leq i \leq j \leq n$
- How many?

Fitting Alignment – Naive Approach

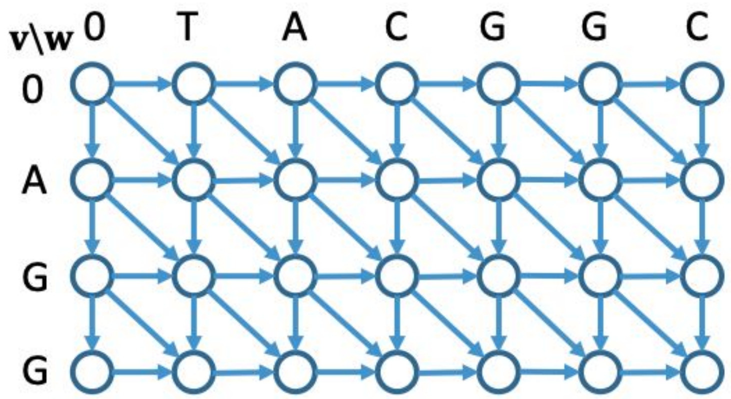
Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$



- Consider all contiguous non-empty substrings of $\mathbf{w}$, defined by $1 \leq i \leq j \leq n$
- How many? Answer: $n + \binom{n}{2}$
- What are their total lengths?
- What is the running time?

Fitting Alignment – Dynamic Programming

Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$

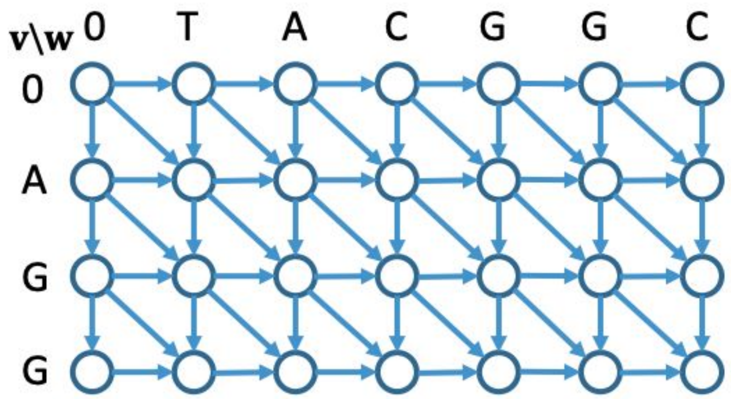


$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0, \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } i > 0 \text{ and } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$$s^* = \max\{s[m,0], \ldots, s[m,n]\}$$

Fitting Alignment – Dynamic Programming

Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$

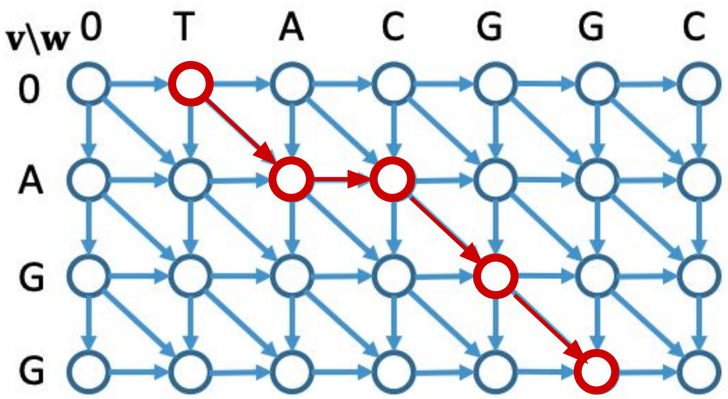


$$s[i,j] = \max \begin{cases} \boxed{0, \text{ Start anywhere on first row} \quad \text{if } i = 0,} \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } i > 0 \text{ and } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$$s^* = \boxed{\max\{s[m,0], \ldots, s[m,n]\} \quad \text{End anywhere on last row}}$$

Fitting Alignment – Dynamic Programming

Fitting Alignment problem: Given strings **v** ∈ Σ^m and **w** ∈ Σ^n and scoring function δ, find an alignment of **v** and a substring of **w** with maximum global alignment score s^* among *all* global alignments of **v** and *all* substrings of **w**



$$s[i,j] = \max \begin{cases} \boxed{0, \text{ Start anywhere on first row } \text{if } i = 0,} \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } i > 0 \text{ and } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$$s^* = \boxed{\max\{s[m,0], \ldots, s[m,n]\} \quad \text{End anywhere on last row}}$$

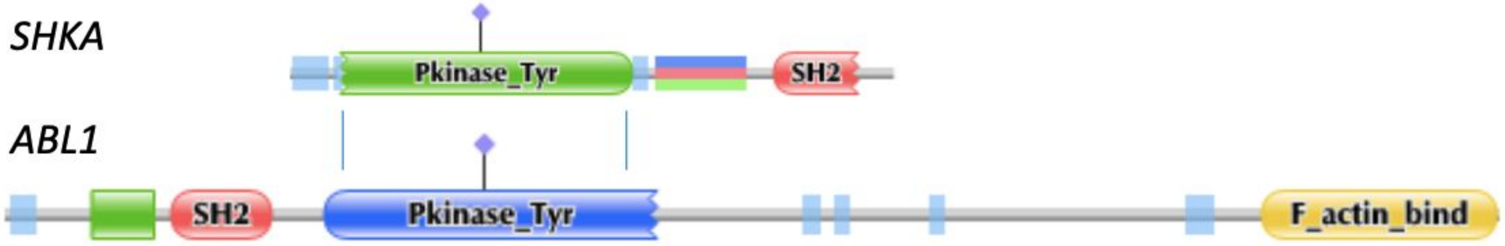
v	A	–	G	G
w	A	C	G	G

Biological sequence alignment problems

- Global alignment ✅
  - Needleman-Wunsch Algorithm
- Fitting alignment ✅
- **Local alignment**

Local Alignment – Biological Motivation

Proteins are composed of functional units called domains. Such domains may occur in different proteins even across species.



From Pfam database (http://pfam.sanger.ac.uk/)

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

Global, Fitting and Local Alignment

Global Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find alignment of $\mathbf{v}$ and $\mathbf{w}$ with maximum score.

Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

Local Alignment – Naive Algorithm

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

Local Alignment – Naive Algorithm

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

Brute force:

1. Generate all pairs $(\mathbf{v}', \mathbf{w}')$ of substrings of $\mathbf{v}$ and $\mathbf{w}$
2. For each pair $(\mathbf{v}', \mathbf{w}')$, solve global alignment problem.

Question: There are $\binom{m}{2}\binom{n}{2}$ pairs of substrings.
But they have different lengths. What is the running time?

Key idea

Global alignment:

- Start at $(0,0)$ and end at (m,n)

Local alignment:

- Start and end anywhere

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

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

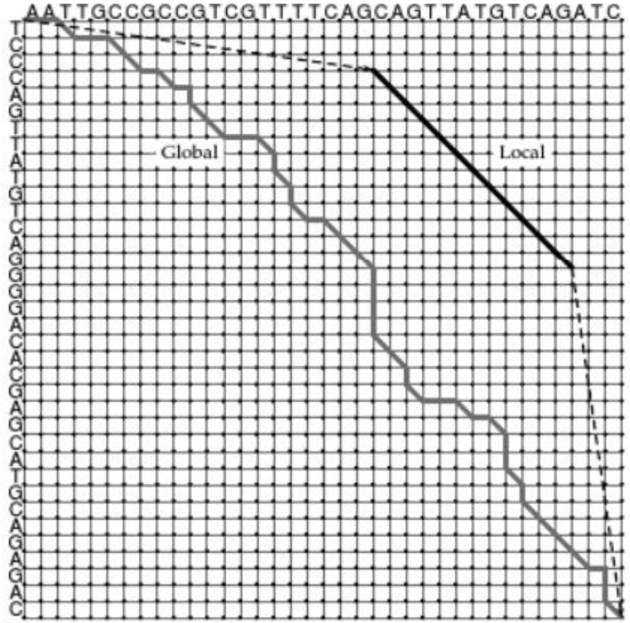

AATTGCCGCCGTCGTTTTCAGCAGTTATGTCAGATC
TCCCAGTTATGTCAGGGGACACGAGCATGCAGAGAC
Global
Local

Figure 6.16 (a) Global and (b) local alignments of two hypothetical genes that each have a conserved domain. The local alignment has a much worse score according to the global scoring scheme, but it correctly locates the conserved domain.

Local Alignment Recurrence

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

$$s[i,j] = \max \begin{cases} 0, & \cancel{\text{if } i = 0 \text{ and } j = 0,} \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$$s^* = \max_{i,j} s[i,j]$$

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

                  tccCAGTTATGTCAGggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

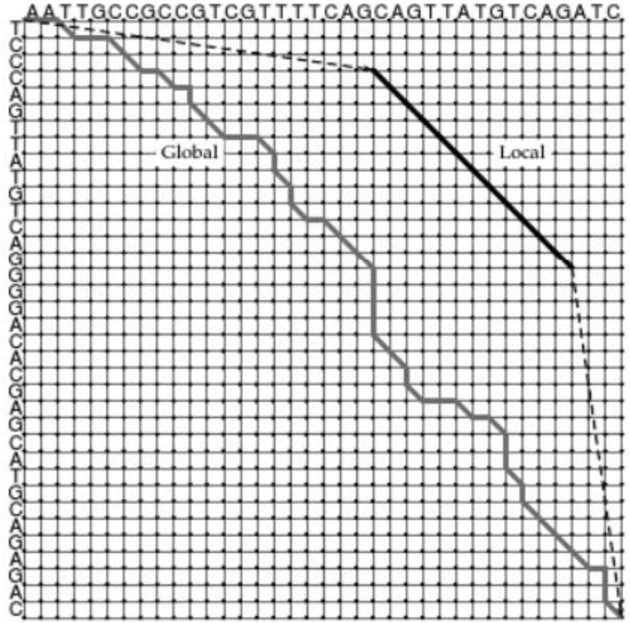

AATTGCCGCCGTCGTTTCAGCAGTTATGTCAGATC
TCCCAGTTATGTCAGGGGACACGAGCATGCAGAGAC
Global
Local

Figure 6.16 (a) Global and (b) local alignments of two hypothetical genes that each have a conserved domain. The local alignment has a much worse score according to the global scoring scheme, but it correctly locates the conserved domain.

Local Alignment Recurrence

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$

$$s[i,j] = \max \begin{cases} \boxed{0, \quad \text{Start anywhere} \qquad \cancel{\text{if } i = 0 \text{ and } j = 0,}} \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$$\boxed{s^* = \max_{i,j} s[i,j] \qquad \text{End anywhere}}$$

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

                  tccCAGTTATGTCAGggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

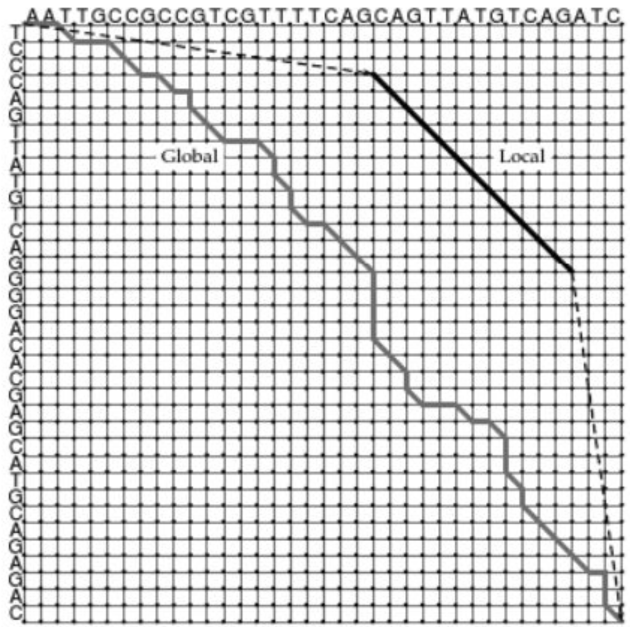
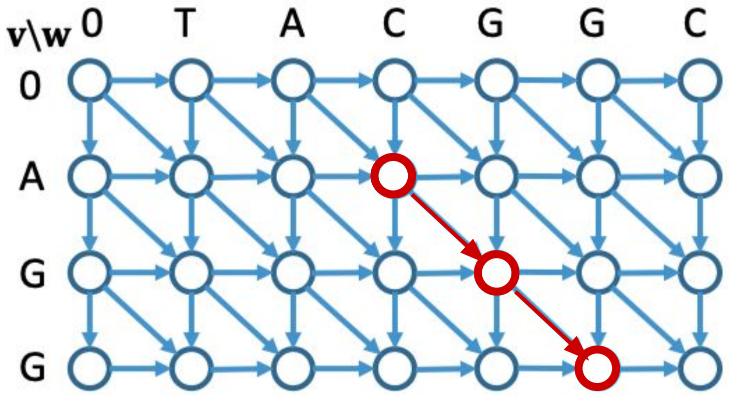

AATTGCCGCCGTCGTTTTCAGCAGTTATGTCAGATC
TCCCAGTTATGTCAGGGACACGAGCATGCAGAGAC
Global
Local

Figure 6.16 (a) Global and (b) local alignments of two hypothetical genes that each have a conserved domain. The local alignment has a much worse score according to the global scoring scheme, but it correctly locates the conserved domain.

Local Alignment – Dynamic Programming



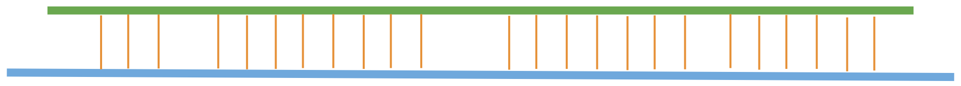
v	G	G
w	G	G

$$s[i,j] = \max \begin{cases} 0, & \text{~~if } i = 0 \text{ and } j = 0,\text{~~} \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

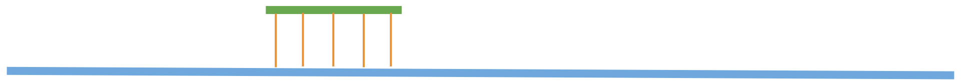
$$s^* = \max_{i,j} s[i,j]$$

Global, Fitting and Local Alignment

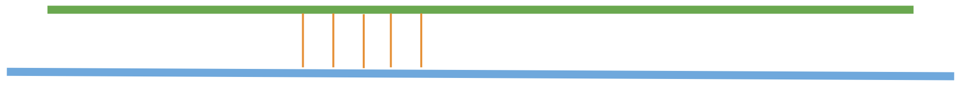
Global alignment



Fitting alignment



Local alignment



Scoring gaps

Let $\mathbf{v}$ = AAC and $\mathbf{w}$ = ACAGGC

Match $\delta(c, c) = 1$;
Mismatch $\delta(c, d) = -1$ (where $c \neq d$); Indel $\delta(c, -) = \delta(-, c) = -2$

v	A	-	-	A	C
w	A	C	A	A	C

v	A	-	A	-	C
w	A	C	A	A	C

Both alignments have 3 matches and 2 indels.
Score: $(3 * 1) + (2 * -2) = -1$

***Question**: Which alignment is better?*

Scoring gaps

Let $\mathbf{v}$ = AAC and $\mathbf{w}$ = ACAGGC

Match $\delta(c, c) = 1$;
Mismatch $\delta(c, d) = -1$ (where $c \neq d$); Indel $\delta(c, -) = \delta(-, c) = -2$

v	A	-	-	A	C
w	A	C	A	A	C

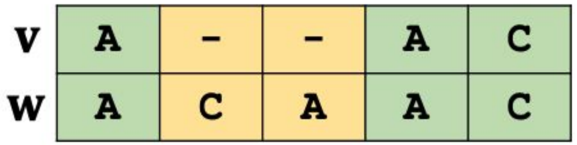
v	A	-	A	-	C
w	A	C	A	A	C

Both alignments have 3 matches and 2 indels.
Score: $(3 * 1) + (2 * -2) = -1$

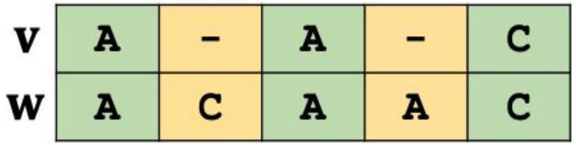
Both have the same score, but the first one is often more plausible (biologically)

Scoring Gaps – Affine Gap Penalties

Desired: Lower penalty for consecutive gaps than interspersed gaps.



v	A	-	-	A	C
w	A	C	A	A	C



v	A	-	A	-	C
w	A	C	A	A	C

- A single insertion of “CA” into the first string could change it into the second -- Biologically, this is much more likely as **v** could be transformed into **w** in “one fell swoop”
- **Why**: Consecutive gaps are more likely due to slippage errors in DNA replication (2-3 nucleotides), codons for protein sequences, etc.

Affine gap penalty

v	A	-	-	A	C
w	A	C	A	A	C

v	A	-	A	-	C
w	A	C	A	A	C

Affine gap penalty: Two penalties: (i) gap open penalty $\rho \geq 0$ and (ii) gap extension penalty $\sigma \geq 0$. Stretch of k consecutive gaps has score $-(\rho + \sigma k)$.

Let $\rho = 10$ and $\sigma = 1$. Left: $(3 * 1) - (10 + 1 * 2) = -9$.
Right: $(3 * 1) - (10 + 1 * 1) - (10 + 1 * 1) = -19$.

- The alignment problem under affine gap penalty is called gapped alignment

Conclusions

1. Edit distance
2. Global alignment
3. Fitting alignment
4. Local alignment
5. Gapped alignment
6. BLOSUM substitution matrix

Edit distance: minimize cost

Global alignment: maximize (generalized) score

Small tweaks enable different extensions

Self-paced PyTorch tutorial learning (01/24)

- **No class this Wednesday (01/24)!**
- Please watch the official PyTorch tutorial videos 1, 2, and 3.
  - After we cover neural networks in class, we will watch video 4, 5, and 6 as well.
- We will release an **exercise** (Colab notebook) on Canvas for you to get familiar with the basic PyTorch data types & operations. Please complete after watching the videos
- Consider joining **TA's OHs** if you have any questions about the PyTorch videos and the exercise.