CMSC 423: Sequence Alignment

Part4

	-	A	G	C	G	T	A	G
-	0	-2	-4	-6	-8	-10	-12	-14
G	-2	-4	8	6				
T	-4	-6	6	4				
C	-6	-8	4	16				
A	-8							
G	-10							
A	-12							

v = AGCGTAG

w = GTCAGA

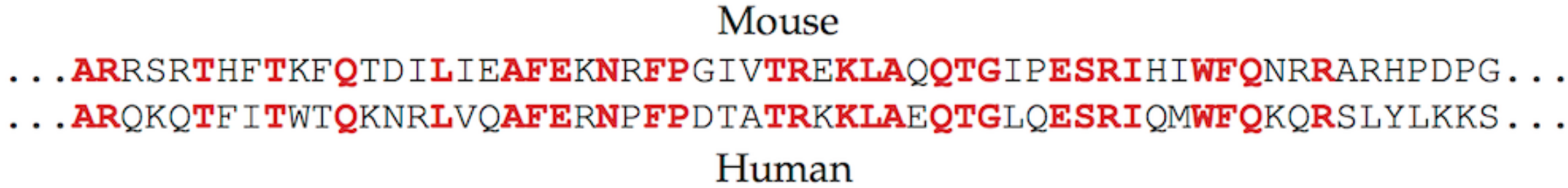
$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + \text{Value}[v[i], w[j]] \\ s_{i-1,\,j} + \text{Value}[v[i], -] \\ s_{i,\,j-1} + \text{Value}[-, w[j]] \end{cases}$$

Match = 10 [A,A], …

Mismatch = -5 [A,G], [A,C], [A,T], …

Gap = -2 [A,-], [-,A], …

Local vs. global alignment



Local vs. global alignment

- Can we change the algorithm to allow *w* to be a substring of *v*?

```
ACAGTTGACCCGTGCAT
----TG-CC-G------
```

- Key idea: gaps at the end of *w* are free
- Simply change the first row in the dynamic programming table to 0s
- Answer is no longer Score[n, m], rather the largest value in the last row

Sub-string alignment

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0							
C								
G								
T								

$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + \text{Value}[v[i], w[j]] \\ s_{i-1,\,j} + \text{Value}[v[i], -] \\ s_{i,\,j-1} + \text{Value}[-, w[j]] \end{cases}$$

Value[Match] = 10
Value[Mismatch] = -5
Value[Gap]= -2

Sub-string alignment

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0							
C	-2							
G	-4							
T	-6							

$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + \text{Value}[v[i], w[j]] \\ s_{i-1,\,j} + \text{Value}[v[i], -] \\ s_{i,\,j-1} + \text{Value}[-, w[j]] \end{cases}$$

Value[Match] = 10
Value[Mismatch] = -5
Value[Gap]= -2

Sub-string alignment

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0 →	0 →	0 →	0 →	0 →	0 →	0 →	0
C	-2							
G	-4							
T	-6							

$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + \text{Value}[v[i], w[j]] \\ s_{i-1,\,j} + \text{Value}[v[i], -] \\ s_{i,\,j-1} + \text{Value}[-, w[j]] \end{cases}$$

Value[Match] = 10
Value[Mismatch] = -5
Value[Gap]= -2

Sub-string alignment

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0	0	0	0	0	0	0	0
C	-2			10	8			
G	-4			8	20			
T	-6			6	18	30	28	26

$$s_{i,j} = \max \begin{cases} s_{i-1,\,j-1} + \text{Value}[v[i], w[j]] \\ s_{i-1,\,j} + \text{Value}[v[i], -] \\ s_{i,\,j-1} + \text{Value}[-, w[j]] \end{cases}$$

Value[Match] = 10
Value[Mismatch] = -5
Value[Gap]= -2

Sub-string alignment

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0	0	0	0	0	0	0	0
C	-2			10	8			
G	-4			8	20			
T	-6			6	18	30	28	26

AGCGTAG

CGT

Local alignment

- What if we just want a region of similarity?
- Change the first row and column in the dynamic programming table to 0s
- Allow the alignment to start anywhere

  Score[i,j] = max{0, case 1, case 2, case 3}

- Answer is the location in the matrix with the highest score

Local alignment

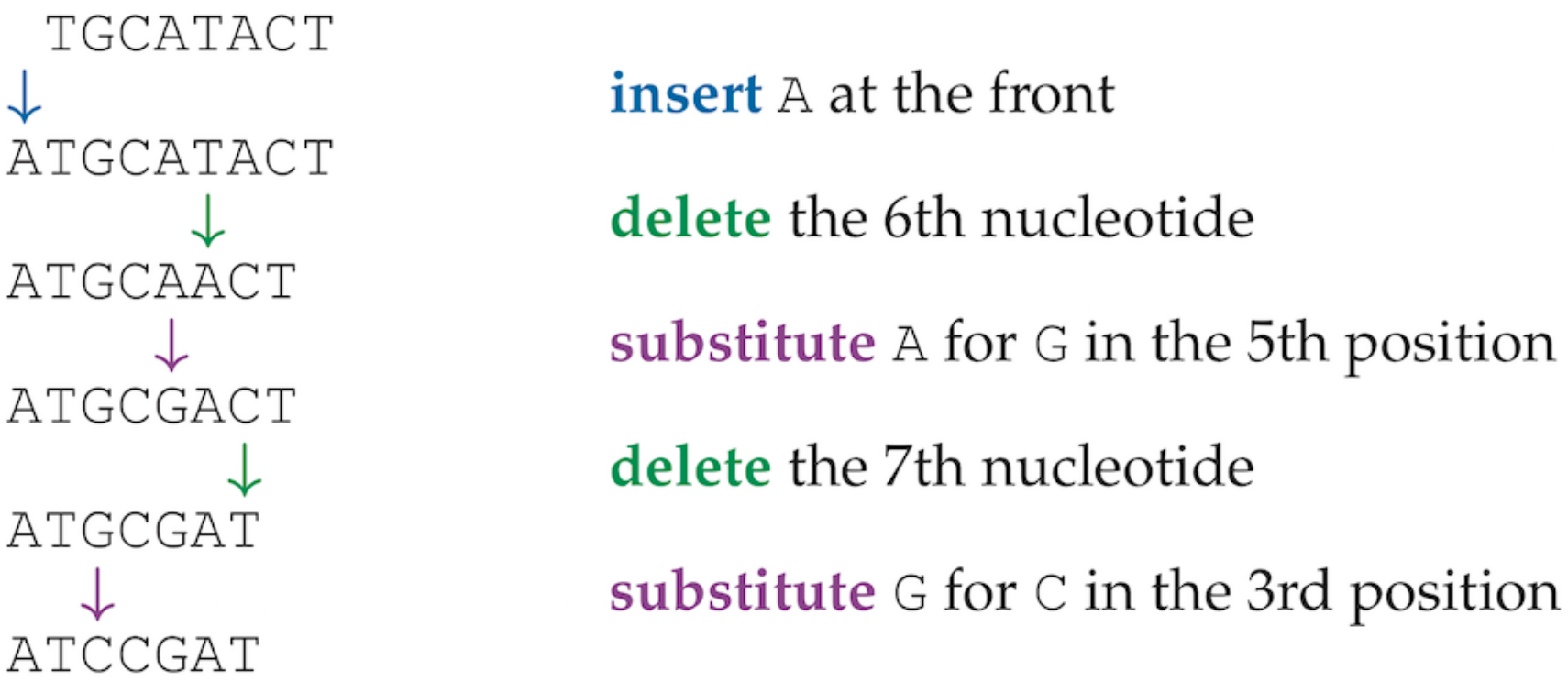
Match = 10
Mismatch = -5
Gap = -2

	-	**A**	**G**	**C**	**G**	**T**	**A**	**G**
-	0	0	0	0	0	0	0	0
C	0							
T	0		0					
C	0			10				
G	0				20			
T	0					30		
C	0							

```
AGCGTAG
|||
CTCGTC
```

Various flavors of alignment

- Alignment problem also called "edit distance" – how many changes do you have to make to a string to convert it into another one.



Various flavors of alignment

- Alignment problem also called "edit distance" – how many changes do you have to make to a string to convert it into another one.

- Edit distance is also called the Levenshtein distance
- Local alignment – Smith-Waterman
- Global alignment – Needleman-Wunsch

Running times

- All these algorithms run in O(mn) – quadratic time
- Note – this is significantly worse than exact matching
- Next week we'll talk about speed-up opportunities
- BTW, how much space is needed?
  - If we only need to find the best score (not the exact alignment as well) – O(min(m,n))
  - If we need to find the best alignment – elegant divide and conquer algorithm leads to linear space solution.