Sequence alignment

Correspondence between bases of two DNA sequences, or between amino acids of two protein sequences

Alignment : 2 x k matrix (k ≥ m, n)

V = ACCTGGTAAA   n = 10

W = ACTGCGTATA   m = 10

8	matches
1	mismatches
1	deletions
1	insertions

V	A	C	C	T	G	—	G	T	A	A	A
W	A	C	—	T	G	C	G	T	A	T	A

“Goodness” of alignments

Given two sequences, there are many possible alignments

```
ATTTTCCC
                distance=2
ATTTACGC
```

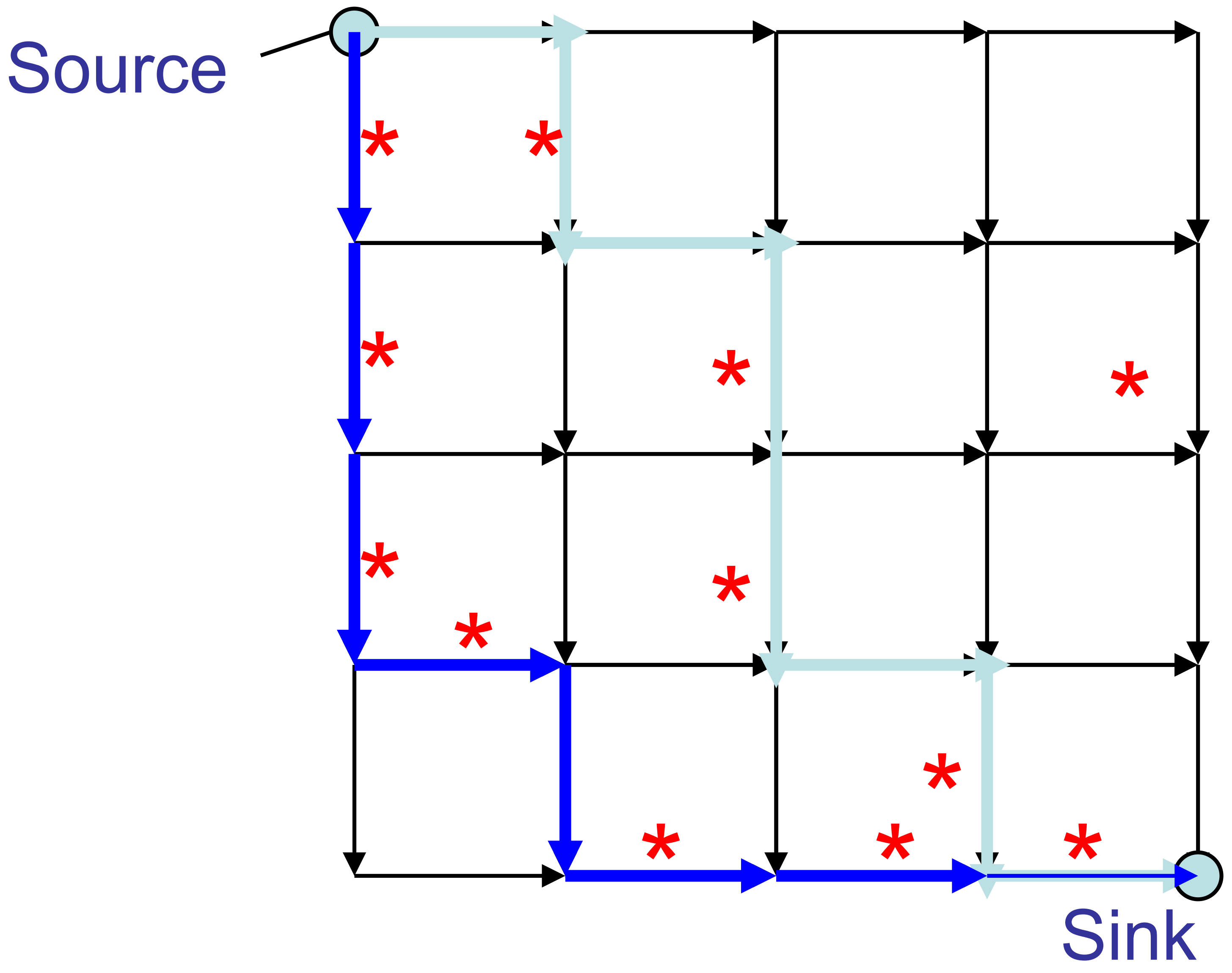
```
ATTT-TCCC
                distance=3
ATTTA-CGC
```

```
ATTTTCCC--------
                        distance=16
--------ATTTACGC
```

Edit distance: the total number of substitutions, insertions and deletions needed to transform one sequence to another

Manhattan tourist problem

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid



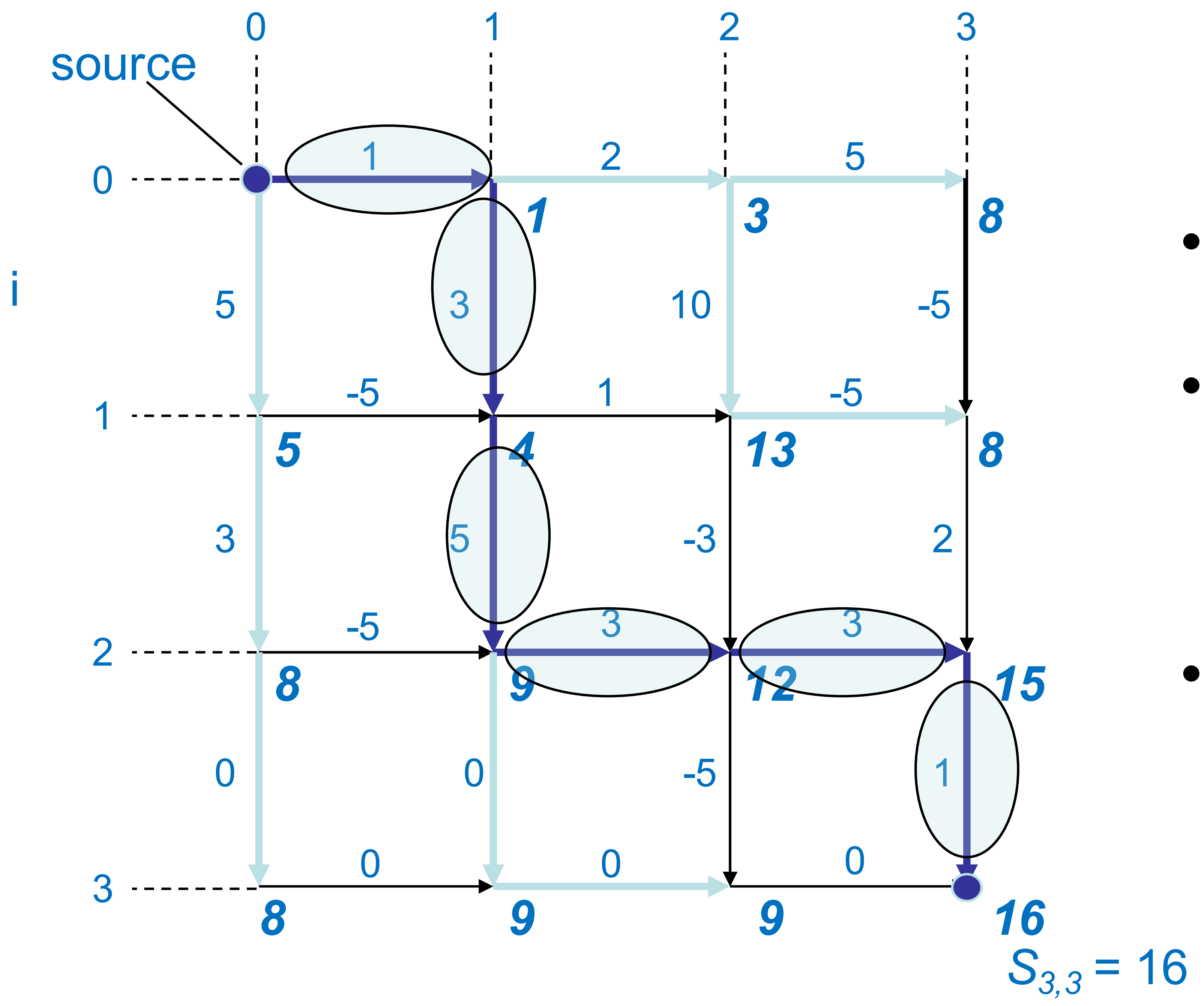
Recursive algorithm -> Dynamic programming

```
Function MT(n,m)

1.  x = MT(n-1,m)+
            weight of the edge from (n-1,m) to (n,m)
2.  y = MT(n,m-1)+
            weight of the edge from (n,m-1) to (n,m)
3.  return max{x,y}
```

MT(x, y) returns the “most weighted” path from point (x, y) to the “sink”.

How to find the optimal path



- Start from Sink.
- Find which of the two edges gave the "max". Take it.
- Repeat.

Recipe

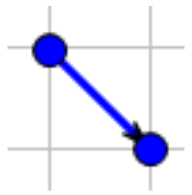
1. Identify subproblems

2. Write down recursions

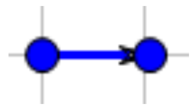
3. Make it dynamic-programming!

The edit distance problem

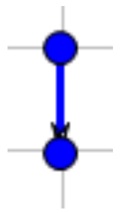
Match

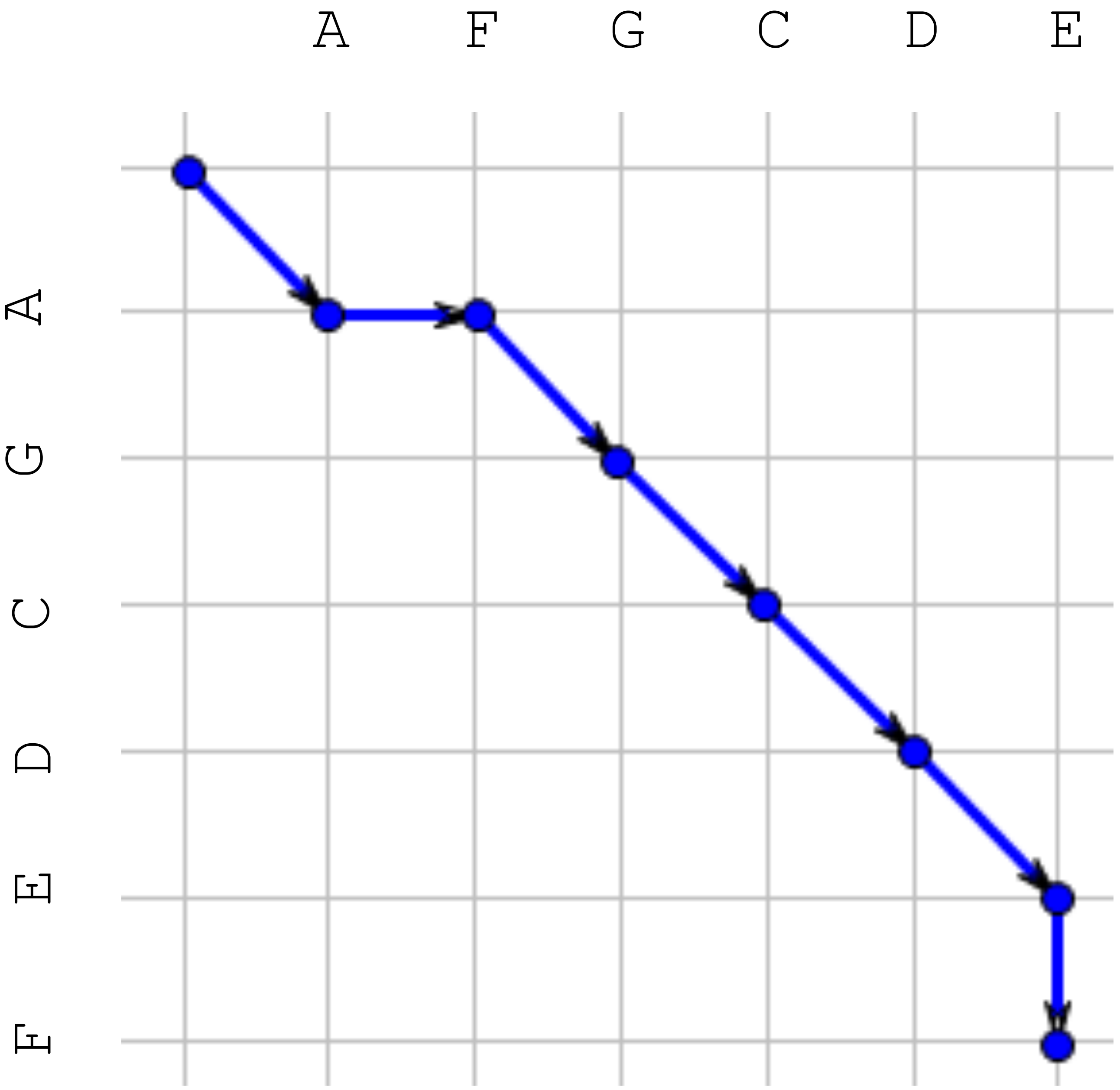


Insertion_X



Insertion_Y





A-GCDEF
AFGCDE-

Minimum Edit Distance

For sequence X and Y

- Initialization

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

- Recurrence Relation:

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

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

- Termination:

```
D(N,M) is distance
```

Optimal alignment

- Base conditions:

  `D(i,0) = i`　　　`D(0,j) = j`

- Recurrence Relation:

Termination:

`D(N,M) is distance`

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

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

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

Complexity

- Time:

  O(nm)

- Space:

  O(nm)

- Backtrace

  O(n+m)

Is the edit distance the best way?

For sequence X and Y

- Initialization

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

- Recurrence Relation:

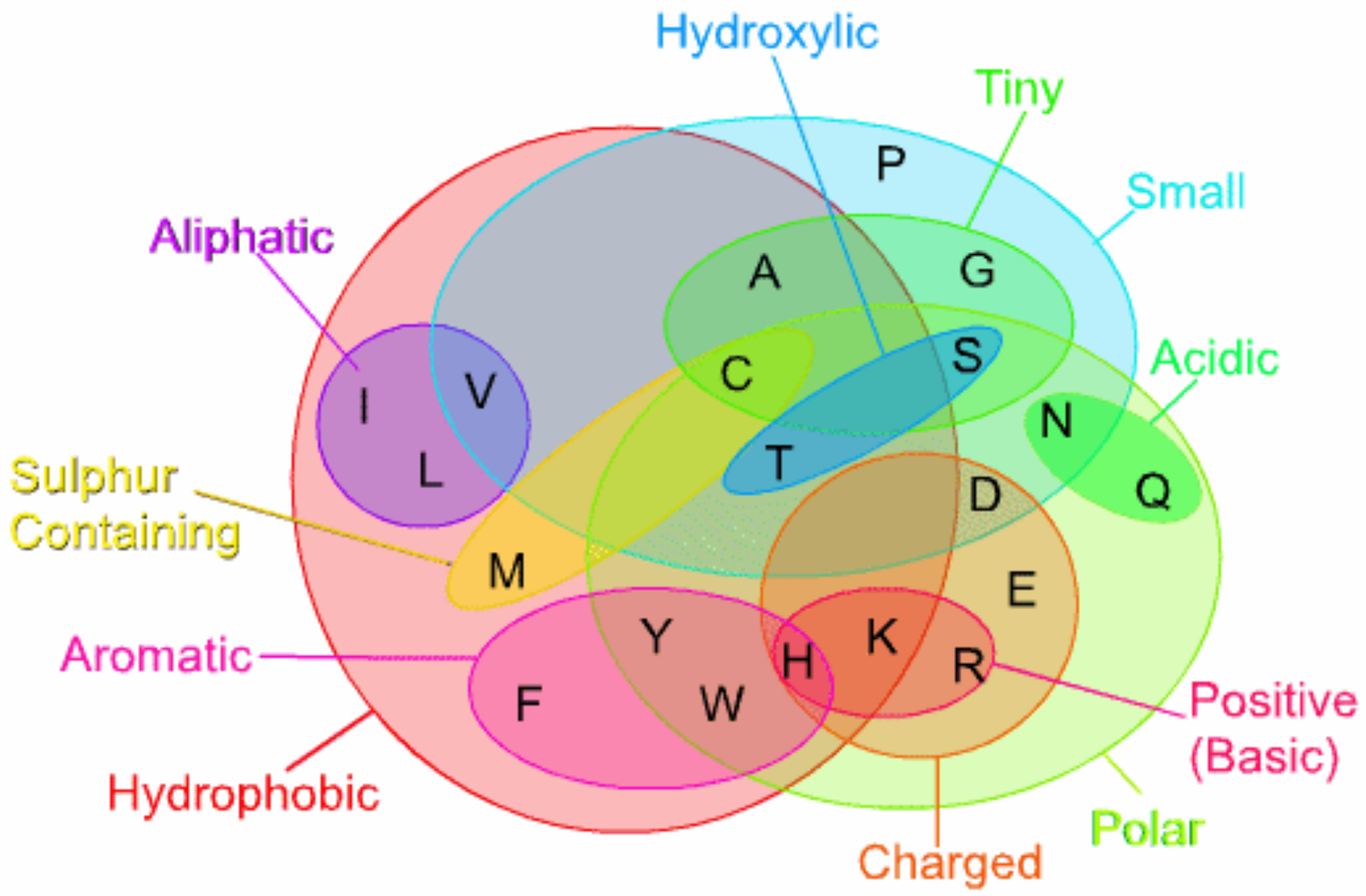
```
For each  i = 1…M
     For each  j = 1…N
```

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

- Termination:

```
D(N,M) is distance
```

Amino acids can share similar properties



Amino Acids

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)

Weighted edit distance

- To generalize scoring for DNA/RNA, consider a 4x4 scoring matrix **S**.
- In the case of an amino acid sequence alignment, the scoring matrix would be a 20x20 size.
- The addition of d is to include the score for comparison of a gap character “-”.
- Two questions:
  - (a) What should **S** be?
  - (b) How do we find optimal scoring alignment?

Weighted edit distance

- To generalize scoring for DNA/RNA, consider a (4+1) x(4+1) scoring matrix **S**.
- In the case of an amino acid sequence alignment, the scoring matrix would be a (20+1)x(20+1) size.
- The addition of d is to include the score for comparison of a gap character “-”.
- Two questions:
    - (a) What should **S** be?
    - (b) How do we find optimal scoring alignment?

Traditionally, people tend to maximize the alignment score with a negative gap penalty score

BLOcks SUbstitution Matrix (BLOSUM)

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
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

amino acids

BLOcks SUbstitution Matrix (BLOSUM)

Introduced by Henikoff & Henikoff in 1992

Start with the BLOCKS database

1. Look for conserved (gapless, >=62% identical) regions in alignments.

2. Count all pairs of amino acids in each column of the alignments.

3. Use amino acid pair frequencies to derive “score” for a mutation/replacement

Recursion for generalized edit distance

$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i-1, j) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$$

Complexity?

Gap score/penalty

```
AAAGAATTCA          AAAGAATTCA
A-A-A-T-CA    vs.   AAA----TCA
```

These have the same score, but the second one is often more plausible.

A single insertion of "GAAT" into the first string could change it into the second — Biologically, this is much more likely as **x** could be transformed into **y** in "one fell swoop".

- Currently, the score of a run of k gaps is $s_{gap} \times k$
- It might be more realistic to support general gap penalty, so that the score of a run of k gaps is $|\mathbf{gscore}(k)| < |(s_{gap} \times k)|$.
- Then, the optimization will prefer to group gaps together.

Affine gap penalty

- We encourage spaces to group together using a special case of general penalties called *affine gap penalties*:

  g_{start} = the cost of starting a gap

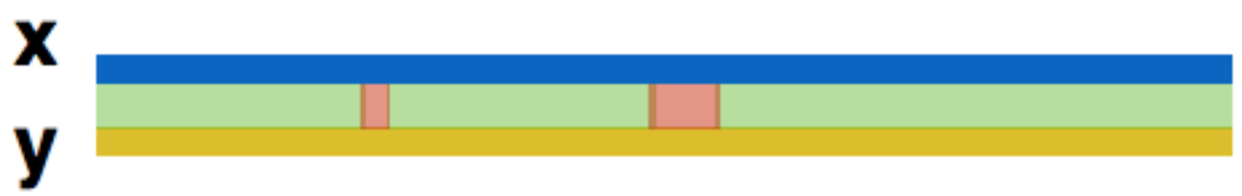
  g_{extend} = the cost of extending a gap by one more space

  $gscore(k) = g_{start} + (k-1) \times g_{extend}$

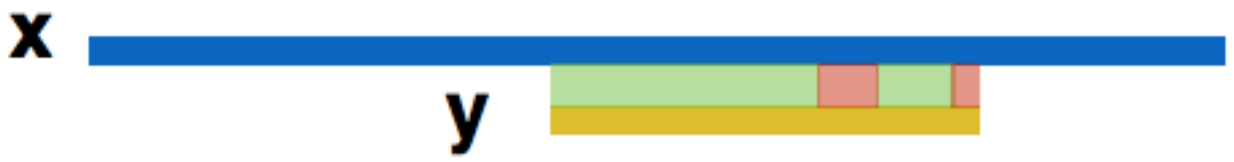
Question: How to develop an efficient dynamic programming algorithm for affine gap penalties?

Categories of pairwise alignments

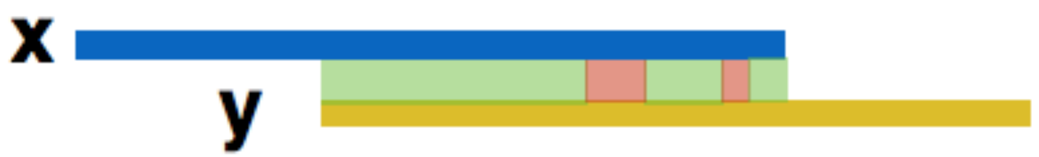
Global: Require an end-to-end alignment of **x**,**y**



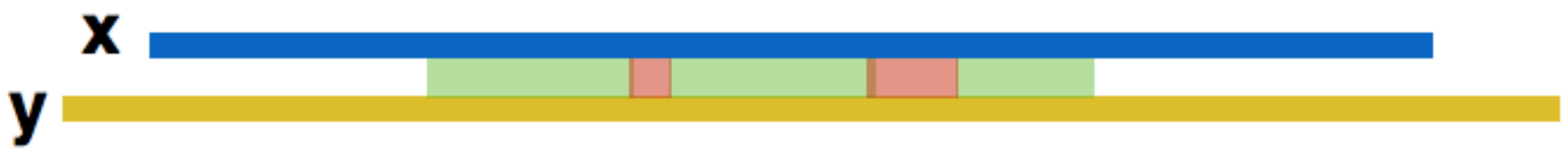
Semi-global (glocal): Gaps at the beginning or end of **x** or **y** are free — useful when one string is significantly shorter than the other or for finding overlaps between strings



or



Local: Find the highest scoring alignment between **x'** a substring of **x** and **y'** a substring of **y** — useful for finding similar regions in strings that may not be globally similar



Semi-global alignment

The recurrence remains the *same*, we only change the base case of the recurrence and the origin of the backtrack

1) Ignore gaps before x $\longrightarrow$ change base case; OPT(0,j) = 0

2) Ignore gaps after x $\longrightarrow$ change traceback; start from $\max_{0<j\leq m}$ OPT(n,j)

3) Ignore gaps before y $\longrightarrow$ change base case; OPT(i,0) = 0

4) Ignore gaps after y $\longrightarrow$ change traceback; start from $\max_{0<i\leq n}$ OPT(i,m)

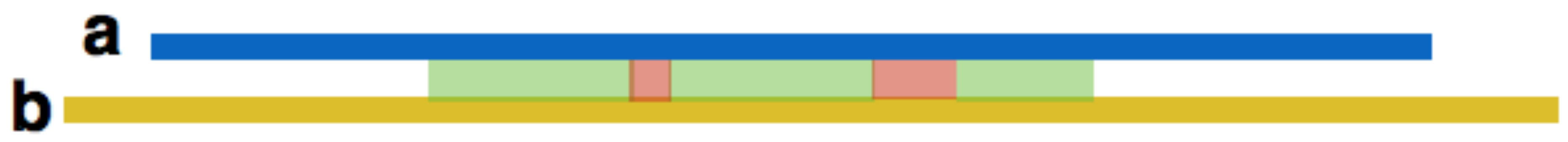
Semi-global alignment

What is the same and different between the "global" and semi-global alignment problems?

*assuming |y| < |x| and we are "fitting" y into x

Global	Semi-global ("fitting")
$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i-1, j) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$	$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i-1, j) \\ s_{\mathrm{gap}} + \mathrm{OPT}(i, j-1) \end{cases}$
Base case: OPT(i,0) = i x s_{gap}	Base case: OPT(i,0) = 0
Traceback starts at OPT(n,m)	Traceback starts at $\max_{0<j\leq n}$ OPT(j,m)

Local alignment: naive algorithm



Local alignment between a and b: Best alignment between a subsequence of a and **a** subsequence of **b.**

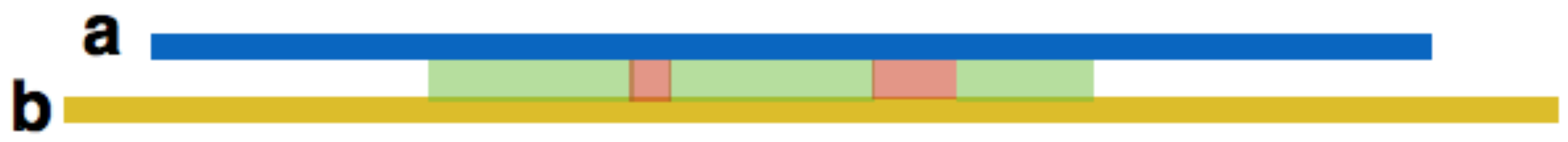
- Long run time $O(n^4)$:

  - In the grid of size n x n there are n^2 vertices (i,j) that may serve as a source.

  - For each such vertex computing alignments from (i,j) to (i',j') takes $O(n^2)$ time.

- This can be remedied by allowing every point to be the starting point

Local alignment: Smith-Waterman algorithm



Local alignment between a and b: Best alignment between a subsequence of a and **a** subsequence of **b**.

$$\mathrm{OPT}(i,j) = \max \begin{cases} \mathrm{score}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i-1, j) \\ \mathrm{s_{gap}} + \mathrm{OPT}(i, j-1) \\ 0 \end{cases}$$

Idea: start over from any entry!

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

	*	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

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

local_align(“catdogfish”, “dog”)

	*	c	a	t	d	o	g	f	i	s	h
*	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	10	3	0	0	0	0	0
o	0	0	0	0	3	20	13	6	0	0	0
g	0	0	0	0	0	13	30	23	16	9	2

local_align(“mississippi”, “issp”)

	*	m	i	s	s	i	s	s	i	p	p	i
*	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	10	3	0	10	3	0	10	3	0	10
s	0	0	3	20	13	6	20	13	6	5	0	3
s	0	0	0	13	30	23	16	30	23	16	9	2
p	0	0	0	6	23	25	18	23	25	33	26	19