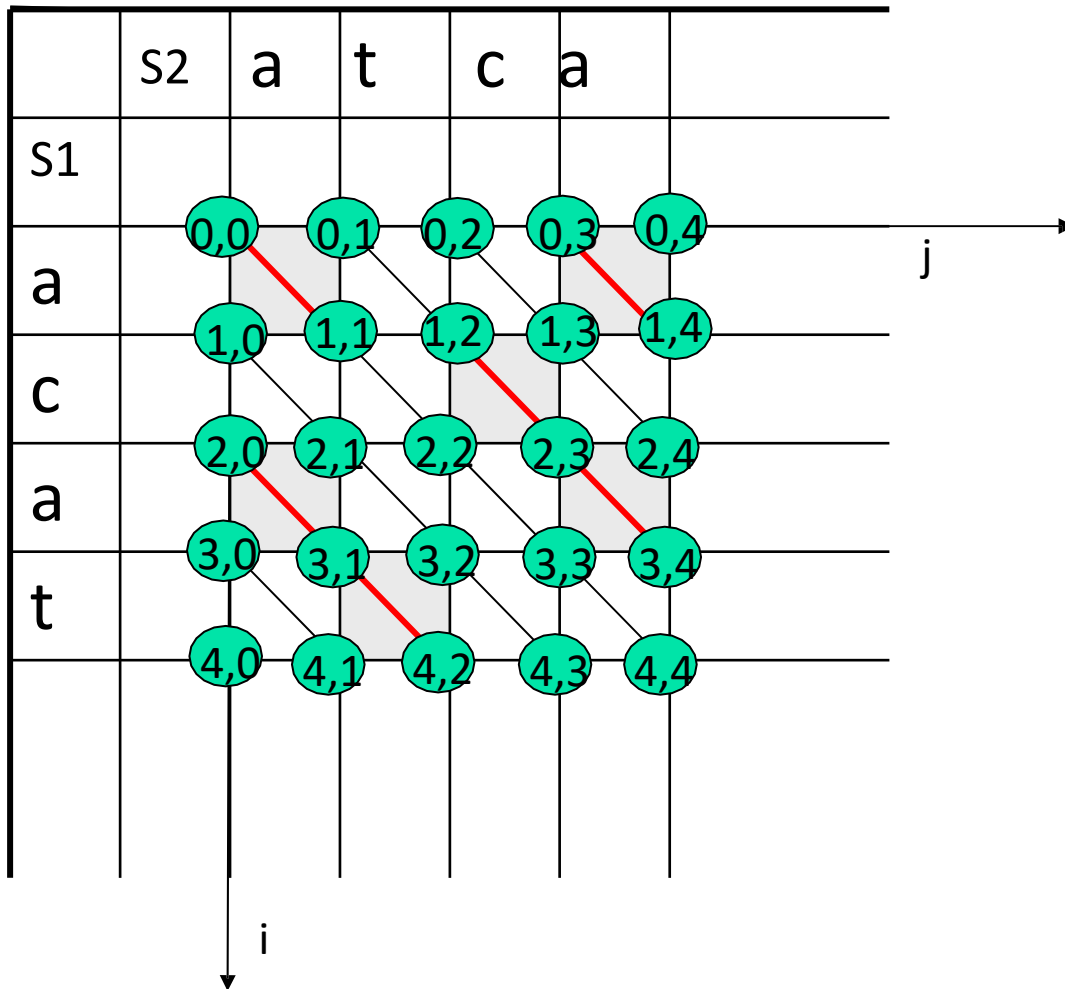


Longest Common Subsequence. String Similarity and alignments.

Lecture 5.2

by Marina Barsky

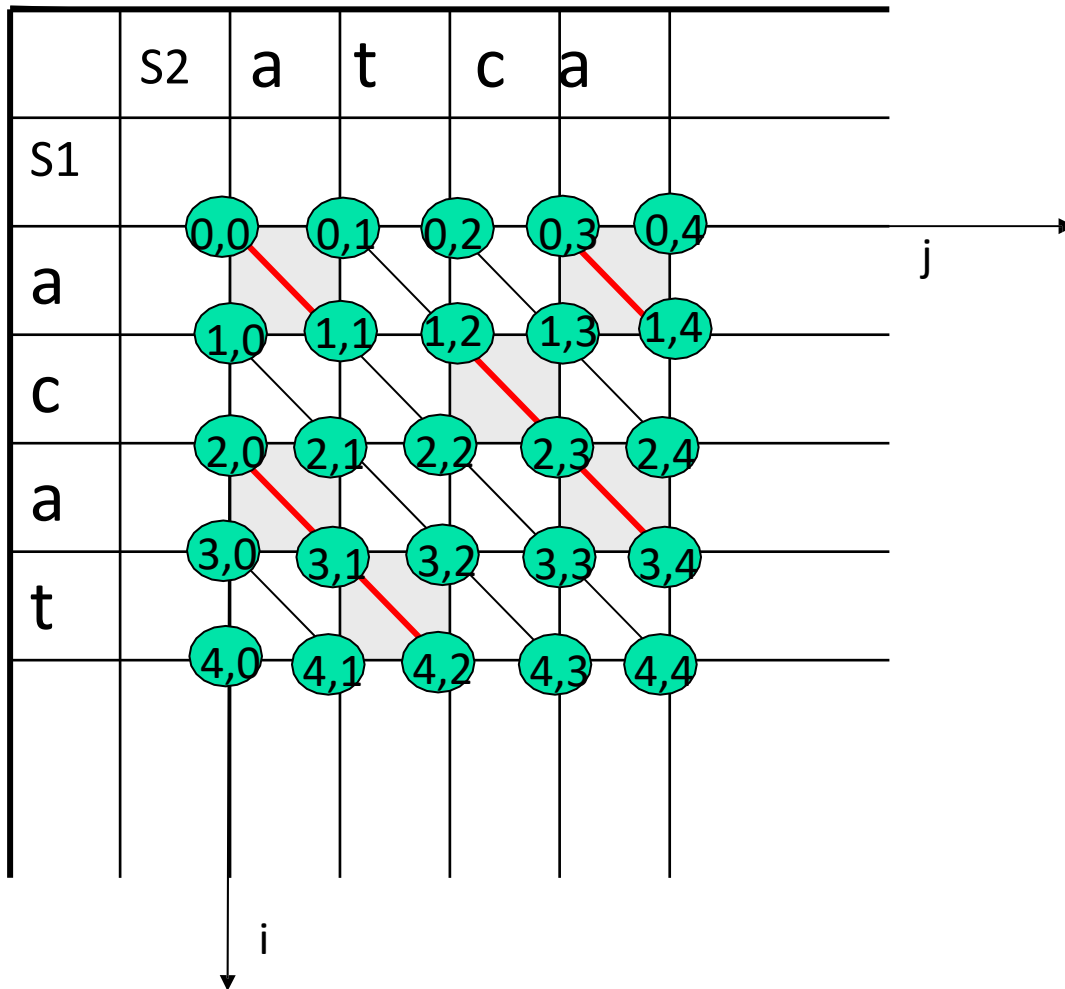
Recap: Useful abstraction: *edit graph*



An **edit graph** for a pair of strings S_1 and S_2 has $(N+1)*(M+1)$ vertices, each labeled with a corresponding pair (i,j) , $0 \leq i \leq N, 0 \leq j \leq M$

The edges are **directed** and their weight depends on the specific string problem: for the edit distance problem – red edges have cost 0, black edges have cost 1

The cheapest path in the edit graph



The cost of a **cheapest path** from vertex $(0,0)$ to vertex (N,M) in this edit graph corresponds to the **edit distance** between S1 and S2, and the path itself represents a series of edit operations and an optimal alignment of S1 with S2

Motivation: sequence similarity

- ❑ Life is based on a repertoire of successful structural and interrelated building blocks which are passed around
- ❑ Biological universality occurs at many levels of details, so we can compare not only the sequence data, but 3D shapes, chemical pathways, morphological features etc.

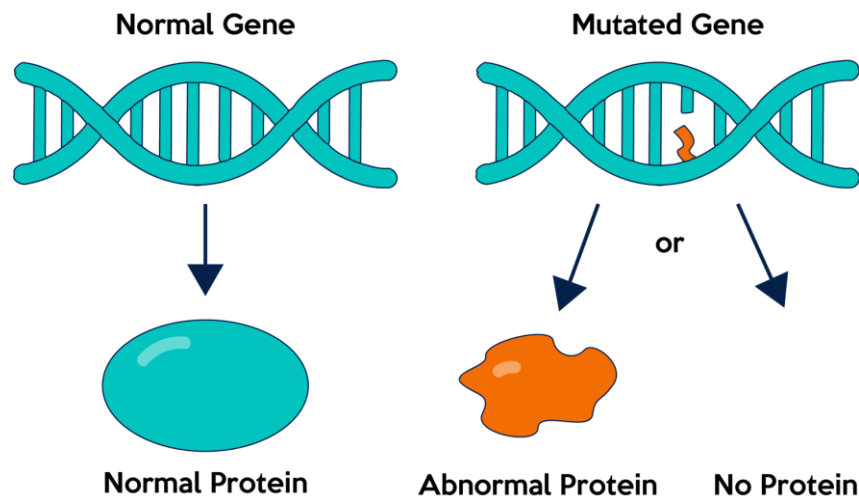
“Everything in life is so similar that the same genes that work in flies are the ones that work in humans” (Wieschaus, 1995)

Why compare biosequences

- ❑ The biological sequences (DNA, RNA or protein) encode and reflect the higher-level molecular structures and mechanisms
- ❑ **High sequence similarity usually implies significant structural and functional similarity**
- ❑ A tractable, though partly heuristic way to infer structure and function of an unknown protein is to search for the similar known proteins at the sequence level: **similar but not identical!**

Note of caution in interpreting sequence similarity

- ❑ There is not a one-to-one correspondence between similar sequences and similar structures or between sequences and functions:
 - ❑ Quite similar structures can be obtained from completely unrelated sequences
 - ❑ Very similar sequences can produce very different structures depending on the location of a change



Edit distance as a similarity metric

<i>S1</i>	<i>a</i>	-	<i>c</i>	<i>a</i>	<i>t</i>
<i>S2</i>	<i>a</i>	<i>t</i>	<i>c</i>	<i>a</i>	-

If the number of basic evolutionary events is small, we infer that the divergence between *S1* and *S2* happened not so long time ago, and that the two strings are still *similar*

The **smaller** is the **edit distance** between 2 strings, the **more similar** they are

Optimal alignment

<i>S1</i>	<i>a</i>	-	<i>c</i>	<i>a</i>	<i>t</i>
<i>S2</i>	<i>a</i>	<i>t</i>	<i>c</i>	<i>a</i>	-

Evolutionary explanation:

S_2 evolved from S_1 by a series of the following mutations:

Insertion of nucleotide *t* at position 2

Deletion of nucleotide *t* at position 5

An optimal alignment is not unique

<i>S1</i>	-	<i>a</i>	<i>t</i>	<i>t</i>	<i>a</i>	<i>a</i>	<i>g</i>
<i>S2</i>	<i>t</i>	<i>a</i>	-	<i>t</i>	<i>c</i>	<i>a</i>	<i>g</i>

<i>S1</i>	-	<i>a</i>	<i>t</i>	<i>t</i>	<i>a</i>	<i>a</i>	<i>g</i>
<i>S2</i>	<i>t</i>	<i>a</i>	<i>t</i>	<i>c</i>	<i>a</i>	-	<i>g</i>

2 different alignments with the optimal minimal cost 3

The exact sequence of changes (mutations) cannot be determined

The edit-distance based similarity metric

S	a	c	c	g	c
S1	a	c		g	c

Edit distance: 1

S	a	c	c	g	c
S2		c	c	g	t

Edit distance: 2

The smaller is the edit distance, the larger is the similarity.

S is more similar to S1 than to S2

The edit-distance based similarity metric: not enough

S	a	c	c	g	c
S1	a	c		t	c

S	a		c	c		g	c
S2	a	c	c	c	t	g	c

The edit distance alone is not always a sufficient metric to characterize similarity between strings

In these 2 examples, the edit distance between S and S1 is the same as an edit distance between S and S2, but it is intuitively clear that S is more similar to S2 than to S1, since they share more identical characters

To infer similarity – we want to evaluate what was preserved rather than what changed

The Longest Common **Substring**

- The longest substring, common to both strings: the longest sequence of consecutive characters which occur in both strings

The longest sequence of ***consecutive matches***

S	a	c	c	g	c
S1	a	c		t	c

S	a		c	c		g	c
S2	a	c	c	c	t	g	c

Not very helpful

The Longest Common *Subsequence*

- A **subsequence** of a string S is a subset of characters of S in their original relative order. A subsequence does not need to consist of the consecutive characters of S
- Given 2 strings $S1$ and $S2$, a **common subsequence** for 2 strings is a subsequence which appears both in $S1$ and $S2$
- **The longest common subsequence** is a longest between all possible subsequences of $S1$ and $S2$

Substring vs subsequence

w	i	n	t	e	r	s
---	---	---	---	---	---	---

w	i	n	t	e	r	s
---	---	---	---	---	---	---

its – a subsequence of *winters*

w	i	n	t	e	r	s
---	---	---	---	---	---	---

inter – both substring and subsequence of *winters*

Longest Common **Subsequence** (LCS)

m	a	d	b	u	n	n	y
b	a	d	m	o	n	e	y

Common subsequence of length 3

m	a	d	b	u	n	n	y
b	a	d	m	o	n	e	y

Common subsequence of length 4

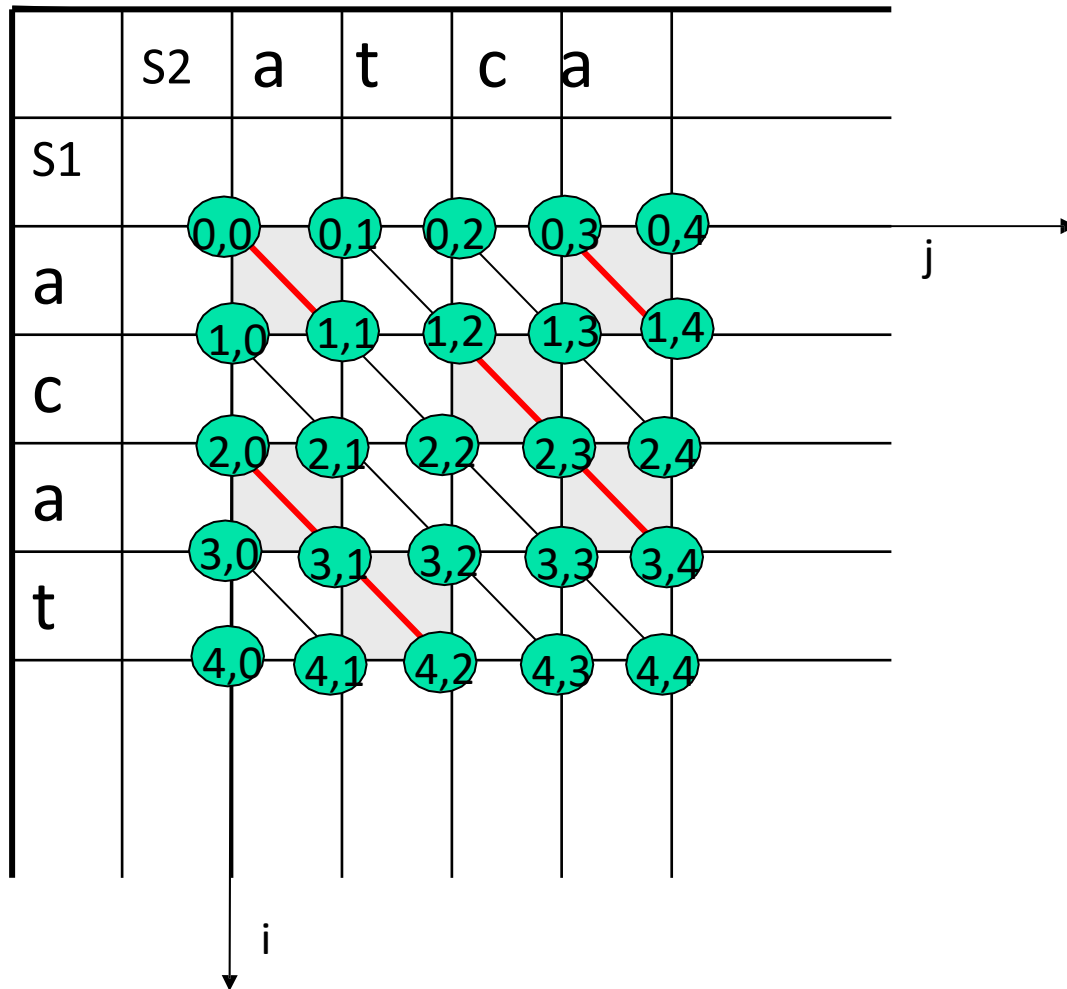
How can we be sure that ***adny*** is the **longest** common subsequence?

The LCS problem

Input: 2 strings S_1 and S_2

Output: the length of *the longest subsequence common to* both strings along with the subsequence itself

Edit Graph for LCS problem



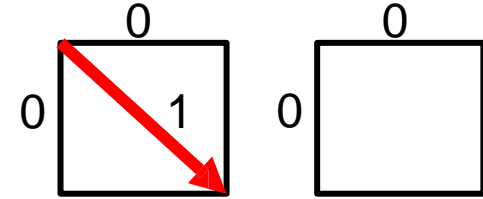
An *edit graph* for a pair of strings S_1 and S_2 can be used to solve the LCS problem

We need to change edge weights: in the LCS problem we are only interested in a sequence of matches – red edges have cost 1, black edges have cost 0

Dynamic Programming solution for LCS.

Edit graph

		<i>b</i>	<i>a</i>	<i>d</i>	<i>m</i>	<i>o</i>	<i>n</i>	<i>e</i>	<i>y</i>
	0								
<i>m</i>									
<i>a</i>									
<i>d</i>									
<i>b</i>									
<i>u</i>									
<i>n</i>									
<i>n</i>									
<i>y</i>									



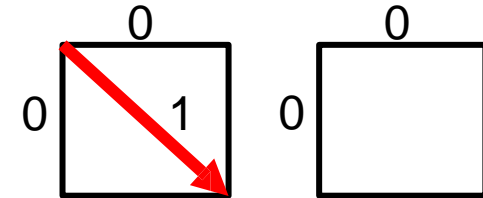
Since we are interested in a longest sequence of matches, we give to the red edges cost 1 and to all the other edges cost 0

Since aligning 2 different characters does not contribute to the total score we do not consider the diagonal edges in case of mismatch

Dynamic Programming solution for LCS.

Greedy path

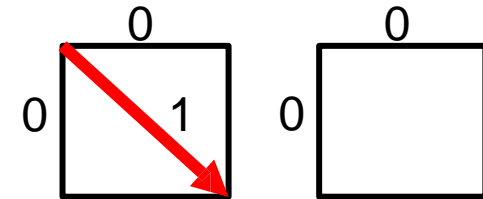
		<i>b</i>	<i>a</i>	<i>d</i>	<i>m</i>	<i>o</i>	<i>n</i>	<i>e</i>	<i>y</i>
	0								
<i>m</i>									
<i>a</i>									
<i>d</i>									
<i>b</i>									
<i>u</i>									
<i>n</i>									
<i>n</i>									
<i>y</i>									



The LCS problem can be reduced to finding the greediest (the longest) path through matches - **the path with the largest cost**

Base condition

		<i>b</i>	<i>a</i>	<i>d</i>	<i>m</i>	<i>o</i>	<i>n</i>	<i>e</i>	<i>y</i>
	0	0	0	0	0	0	0	0	0
<i>m</i>	0								
<i>a</i>	0								
<i>d</i>	0								
<i>b</i>	0								
<i>u</i>	0								
<i>n</i>	0								
<i>n</i>	0								
<i>y</i>	0								



All the black edges are of cost 0, so moving strictly right or down gives paths of a total cost 0

LCS. Recurrence relation

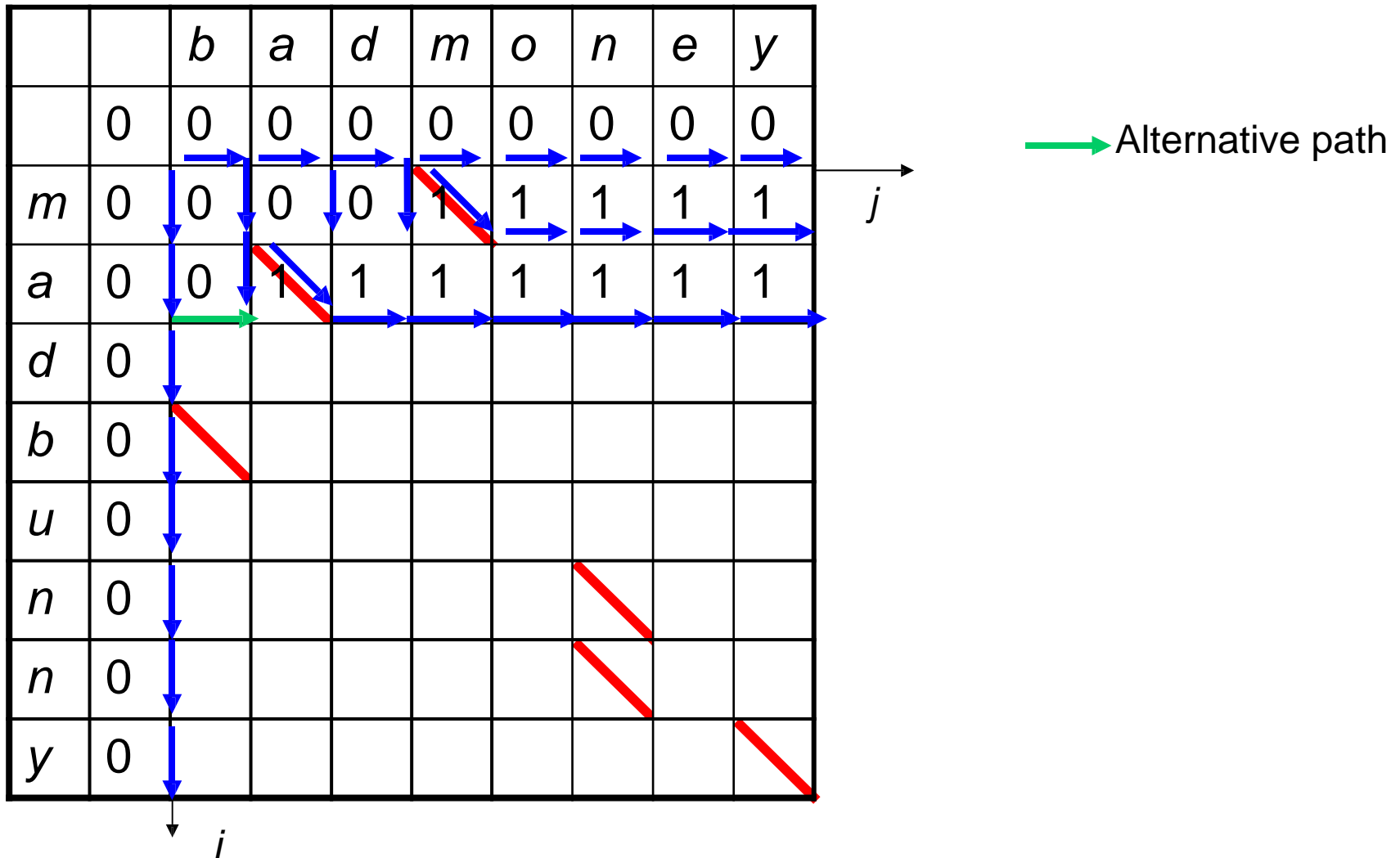
$$\text{COST}(i,j)=\max \begin{cases} \text{COST}(i-1,j) \\ \text{COST}(i,j-1) \\ \text{COST}(i-1,j-1)+1 \text{ if } S1[i]=S2[j] \end{cases}$$

We only consider the diagonal edge if the characters match

Tabular computation. Row 1

		<i>b</i>	<i>a</i>	<i>d</i>	<i>m</i>	<i>o</i>	<i>n</i>	<i>e</i>	<i>y</i>
	0	0	0	0	0	0	0	0	0
<i>m</i>	0	0	0	0	1	1	1	1	1
<i>a</i>	0								
<i>d</i>	0								
<i>b</i>	0								
<i>u</i>	0								
<i>n</i>	0								
<i>n</i>	0								
<i>y</i>	0								

Tabular computation. Row 2

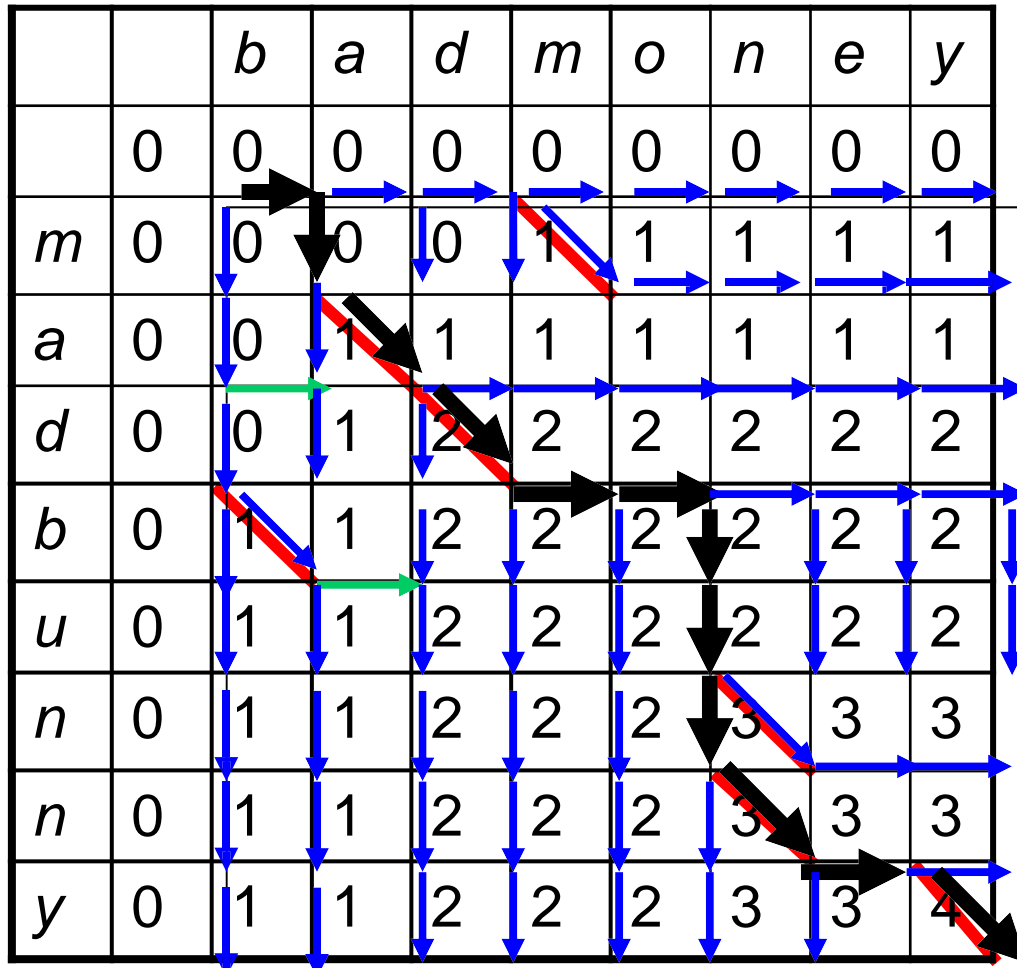


Tabular computation. Row 3

		<i>b</i>	<i>a</i>	<i>d</i>	<i>m</i>	<i>o</i>	<i>n</i>	<i>e</i>	<i>y</i>
	0	0	0	0	0	0	0	0	0
<i>m</i>	0	0	0	0	1	1	1	1	1
<i>a</i>	0	0	1	1	1	1	1	1	1
<i>d</i>	0	0	1	2	2	2	2	2	2
<i>b</i>	0								
<i>u</i>	0								
<i>n</i>	0								
<i>n</i>	0								
<i>y</i>	0								

i ↓ → *j*

LCS. Alignment



j

i

S1	-	m	a	d	-	-	b	u	n	n	-	y
S2	b	-	a	d	m	o	-	-	-	n	e	y

Note, that **only the matches are aligned**, since the problem we are solving – find the longest sequence of matches

We don't count the number of edit operations, since their cost in this model is 0

The edit-distance based similarity metric: not enough

S	a	c	c	g	c
S1	a	c		t	c

S	a		c	c		g	c
S2	a	c	c	c	t	g	c

In these 2 examples, the edit distance between S and S1 is the same as an edit distance between S and S2, but it is intuitively clear that S is more similar to S2 than to S1, since they share **more identical characters**

The LCS-based similarity metric: not enough

S	a	c	c	c
S1	a	c	-	c

S	a	-	c	c	-	-	c
S2	a	t	c	-	t	g	c

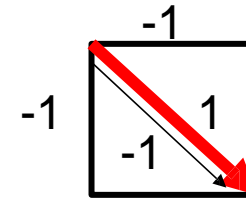
The longer is the LCS, the more similar are two strings

The LCS alone is not a sufficient similarity metric

In these 2 examples, the LCS of S and S1 is the same as the LCS of S and S2, but it is intuitively clear that S is more similar to S1 than to S2, since they have **less different characters**

We want to score both the matches and the differences

Basic optimal alignment scores



	S2	t	g	c	a	t	a
S1							
a							
t							
c							
t							
g							
a							
t							

Let us set the simplest weights of the edges:

For a match: award of 1

For a mismatch: penalty of -1

For a gap (insertion/deletion):

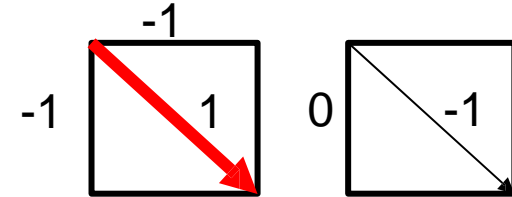
penalty of -1

Then the maximum cost of the path in the edit graph will give a numerical score of the similarity between S1 and S2: large positive values – two strings are similar, negative or low positive values – the strings are different

Everything else is exactly the same

Optimal alignment. Base condition

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1						
t	-2						
c	-3						
t	-4						
g	-5						
a	-6						
t	-7						



Since moving from point (0,0) strictly to the right or to the bottom corresponds to a series of gaps, we initialize the 0-column and 0-row with consecutive negative integers

Optimal alignment.

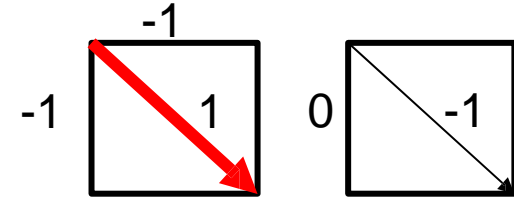
Recurrence relation

$$\text{COST}(i,j)=\max \begin{cases} \text{COST}(i-1,j) - 1 \\ \text{COST}(i,j-1) - 1 \\ \text{COST}(i-1,j-1)+\text{diagonal}(i,j) \end{cases}$$

$$\text{diagonal}(i,j)= \begin{cases} 1 \text{ if } S1[i]=S2[j] \\ -1 \text{ if } S1[i]\neq S2[j] \end{cases}$$

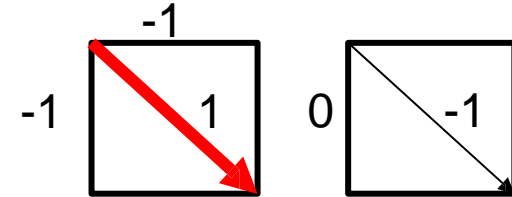
Optimal alignment. Row 1

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2						
c	-3						
t	-4						
g	-5						
a	-6						
t	-7						



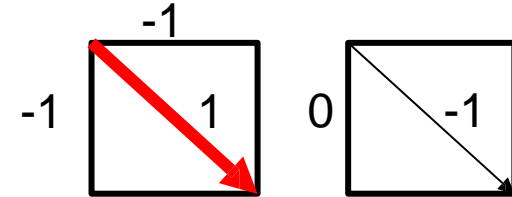
Optimal alignment. Row 2

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3						
t	-4						
g	-5						
a	-6						
t	-7						



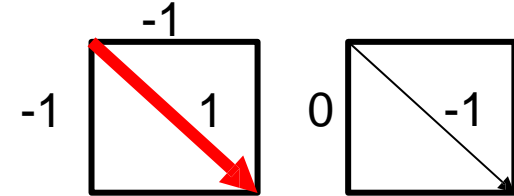
Optimal alignment. Row 3

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3	-1	-1	0	-1	-2	-2
t	-4						
g	-5						
a	-6						
t	-7						



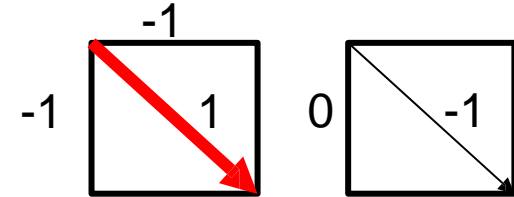
Optimal alignment. Row 4

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3	-1	-1	0	-1	-2	-2
t	-4	0	-1	-1	-1	0	-1
g	-5						
a	-6						
t	-7						



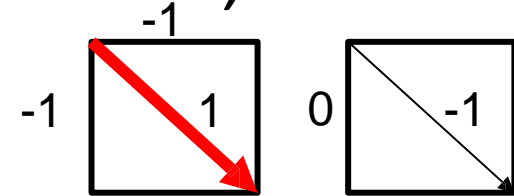
Optimal alignment. Row 5

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3	-1	-1	0	-1	-2	-2
t	-4	-2	-1	-1	-1	0	-1
g	-5	-1	-1	-2	-2	-1	-1
a	-6						
t	-7						



Optimal alignment. Rows 6,7

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3	-1	-1	0	-1	-2	-2
t	-4	-2	-1	-1	-1	0	-1
g	-5	-1	-1	-2	-2	-1	-1
a	-6	-2	-2	-2	-1	-2	0
t	-7	-3	-3	-3	-2	0	-1



Optimal alignment. Alignment

	S2	t	g	c	a	t	a
S1	0	-1	-2	-3	-4	-5	-6
a	-1	-1	-2	-3	-2	-3	-2
t	-2	0	-1	-2	-3	-1	-2
c	-3	-1	-1	0	-1	-2	-2
t	-4	-2	-1	-1	-1	0	-1
g	-5	-1	-1	-2	-2	-1	-1
a	-6	-2	-2	-2	-1	-2	0
t	-7	-3	-3	-3	-2	0	-1

S1	a	t	c	t	g	-	a	t	-
S2	-	t	-	-	g	c	a	t	a

General scoring schemes

$$\text{COST}(i,j)=\max \begin{cases} \text{COST}(i-1,j) + \text{gapCost} \\ \text{COST}(i,j-1) + \text{gapCost} \\ \text{COST}(i-1,j-1)+\text{score}(S1[i], S2[j]) \end{cases}$$

Here the *gapCost* is the cost of aligning each character with a gap, and it should be negative in order to penalize

score depends on the characters placed opposite to each other. It is always positive for a pair of matching characters

The total score is a summative score of aligning the characters in S1 and S2, maximized over all the combinations of possible alignments

The scoring matrix

For an alphabet Σ of size σ add one more artificial character '-'.
Then the scoring matrix is a $(\sigma+1)*(\sigma+1)$ table, where for each character of Σ plus '-' there is a cost of aligning this character with each other character.

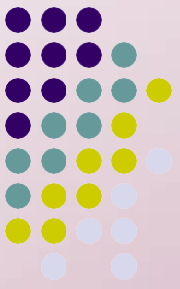
If an optimal alignment has been computed according to a given scoring matrix, the total score of an alignment is the sum of scores of the columns of an alignment table

Our scoring matrix

	a	c	g	t	-
a	1	-1	-1	-1	-1
c	-1	1	-1	-1	-1
g	-1	-1	1	-1	-1
t	-1	-1	-1	1	-1
-	-1	-1	-1	-1	n/a

S1	a	t	c	t	g	-	a	t	-
S2	-	t	-	-	g	c	a	t	a
	-1	1	-1	-1	1	-1	1	1	-1

Total score is -1



The sequence of mutations

S1	a	t	c	t	g	-	a	t	-
S2	-	t	-	-	c	c	a	t	a

This alignment suggests that S1 was transformed into S2 by the following sequence of evolutionary events:

Deletion of nucleotide a

Deletion of nucleotides c and t

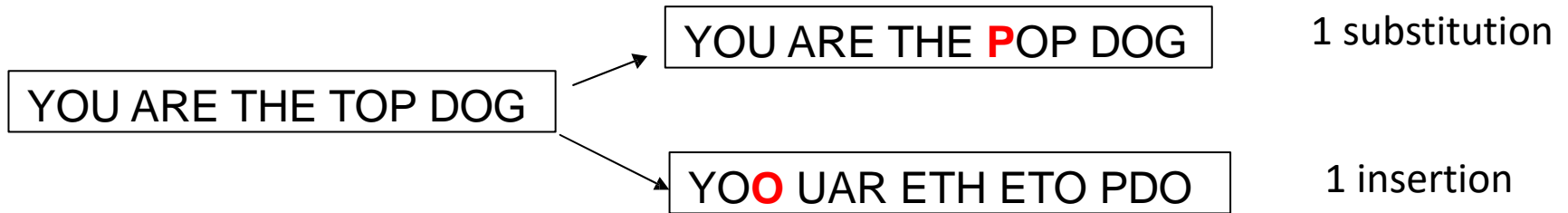
Substitution of nucleotide g by c

Insertion of nucleotide c

Deletion of nucleotide a

Since an optimal alignment is not unique, this sequence of mutations is only one of many possible explanations

Point Mutations

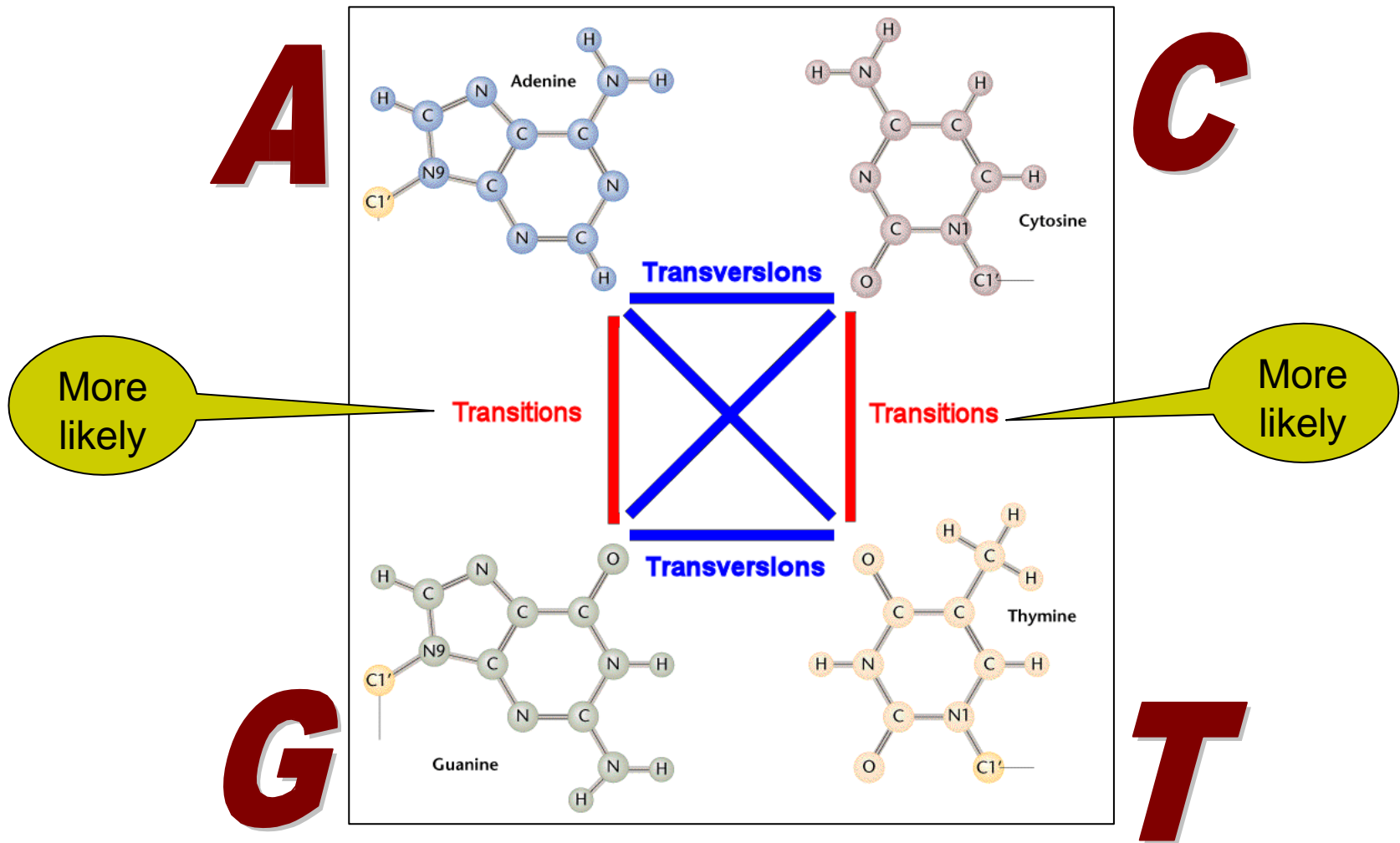


- Mutagenesis (causes of mutations)
 - Wrong base-pairing during replication - point
 - Damage from the environmental agents - point
 - Unequal crossing-over - macromutations
 - Insertions from mobile genes (transposons) – macromutations
- Point mutations can be as deleterious as the macromutations, since they can break the reading frame or introduce a stop codon in the middle of the reading frame

Mutations

- Regulatory mechanisms of DNA repair try to undo the mutations
- Despite this, all cells possess a spontaneous mutation rate defined as a number of mutations which normally occur in each genome over a particular time
- This allows to infer the evolutionary distance between species diverged from a common ancestor

Some mutations are more likely than the others



The first scoring matrix for a real DNA

A, G – 2-ring bases

T, C – 1-ring bases

Mutation which preserves rings number is much more likely than changing the number of rings.

	a	c	g	t	-
a	3	0	2	0	-1
c	0	3	0	2	-1
g	2	0	3	0	-1
t	0	2	0	3	-1
-	-1	-1	-1	-1	

The score of exact matches: + 3

The score of transitions A->G, G->A and T->C, C->A: + 2

The score of any other mismatch (transversions) is 0

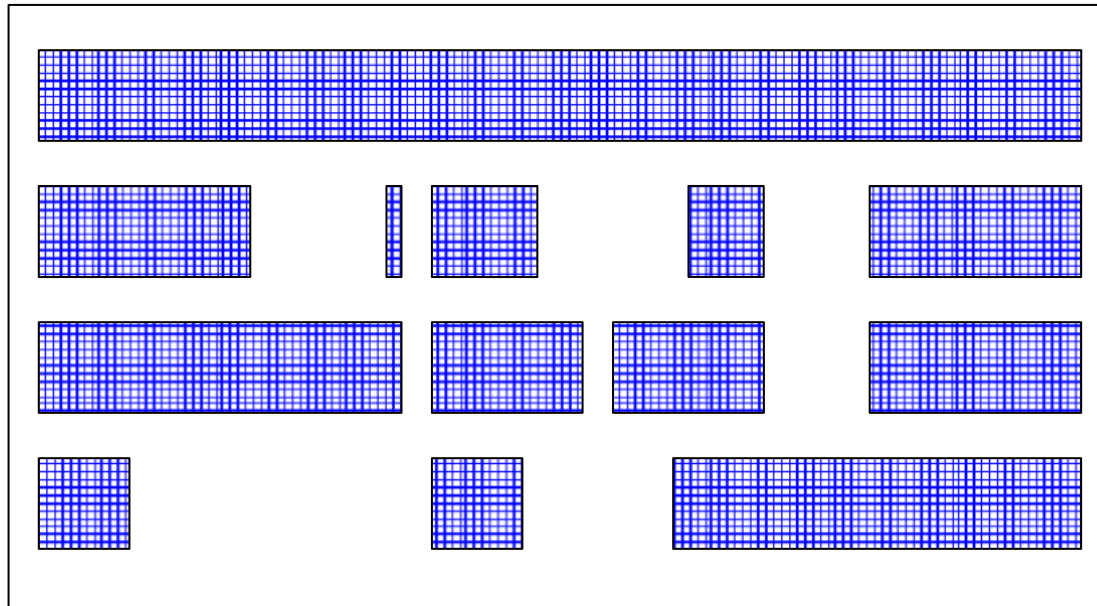
Gaps

- The deletion or insertion of a single nucleotide is often called *indel* (insertion/deletion)
- In real molecular life, the insertions/ deletions occur in a consecutive block, rather than at the level of single nucleotides
- The deletion/insertion of an entire substring occurs as a single mutational event
- The sequence of consecutive insertions/deletions is called a *gap*

Scoring gaps

Each row represents a part of the genomic sequence of a different strain of HIV viruses. 3 bottom rows represent mutated genotypes with an ancestral sequence in the top row.

How many evolutionary events did really occur in each of these 3 cases?



Scoring gaps

- An optimal alignment of two biological sequences is intended to reflect the likelihood of mutational events.
- Since a gap of more than 1 space can be created by a single mutational event, the alignment model should reflect the true distribution of indels in gaps, not merely the number of indels in an alignment

Scoring gaps

- Constant gap weights
 - Give score -1 for each gap independently of its length

- Affine gap weights
 - Give score $\rho + \mu M$ for a gap of length M
 - ρ is comparatively large (for example, -1)
 - μ is comparatively small (for example -0.01)

In this way we count each gap as a single mutational event, but we take into account that longer gaps are less likely to occur than the shorter gaps

The recurrence relation for affine gap weights

$$\begin{array}{l} \downarrow \\ \text{COST}(i,j)=\max \end{array} \left\{ \begin{array}{l} \downarrow \\ \text{COST}(i-1,j) - 0.01 \\ \\ \text{COST}(i-1,j) - 1 - 0.01 \end{array} \right.$$

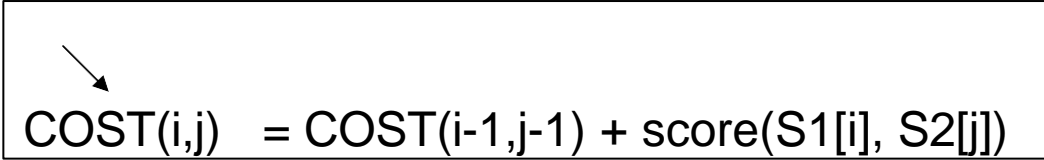
$$\begin{array}{l} \rightarrow \\ \text{COST}(i,j)=\max \end{array} \left\{ \begin{array}{l} \rightarrow \\ \text{COST}(i,j-1) - 0.01 \\ \\ \text{COST}(i,j-1) - 1 - 0.01 \end{array} \right.$$

When we compute the cost of moving from the top, we distinguish 2 cases:

1. if the top character was already a part of a gap, we just penalize for the extension of the gap.
2. Otherwise, we penalize for the opening of a new gap of length 1

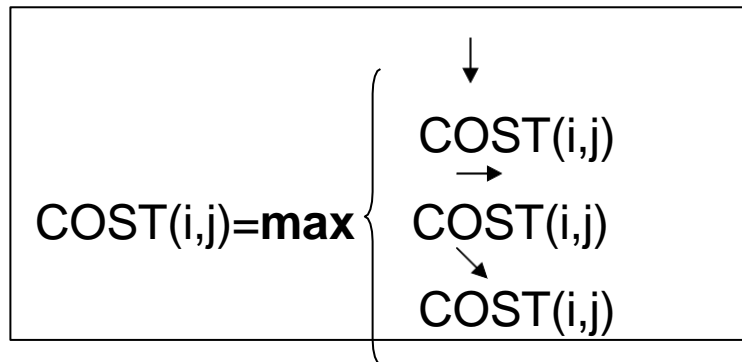
The same when computing the cost of moving from the left to the current cell

The recurrence relation for affine gap weights


$$\text{COST}(i,j) = \text{COST}(i-1,j-1) + \text{score}(S1[i], S2[j])$$

When computing the cost of moving from a diagonal cell,

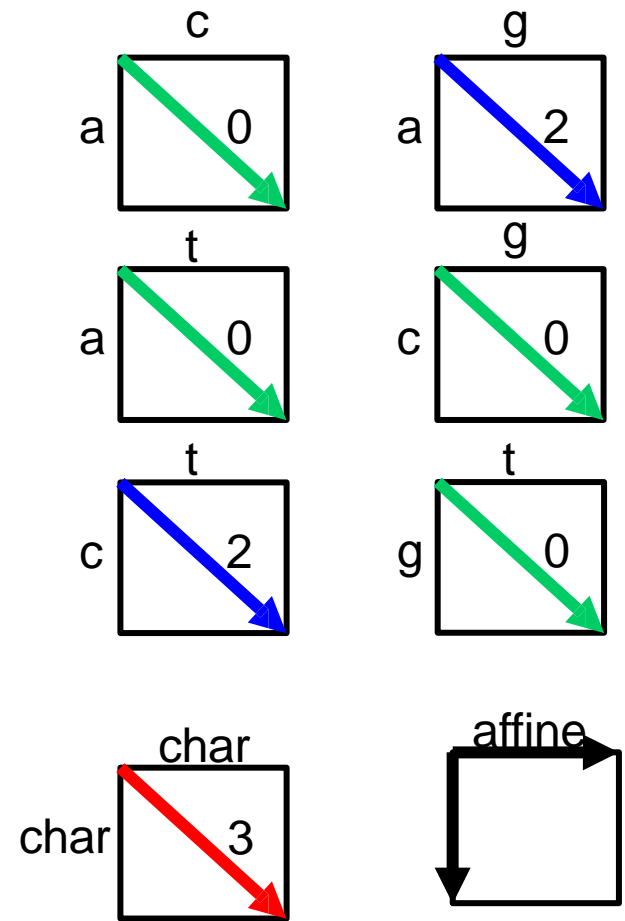
we account only for a score of aligning characters at current positions $S1[i]$ and $S2[j]$, as we did before



Then we take the max of these 3 values

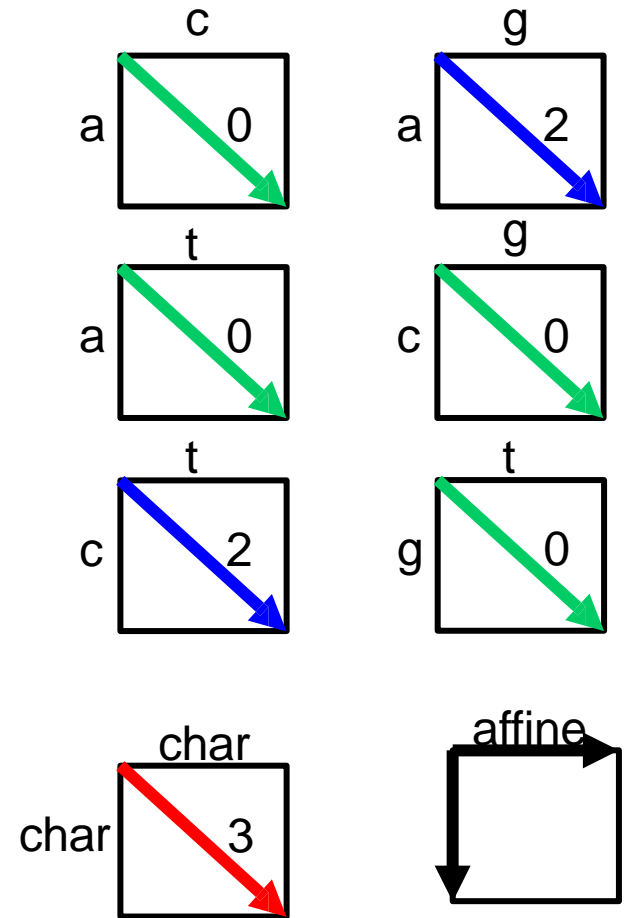
Optimal alignment with affine gap weights and the DNA scoring matrix

	S2	t	g	c	a	t	a
S1	0						
a							
t							
c							
t							
g							
a							
t							



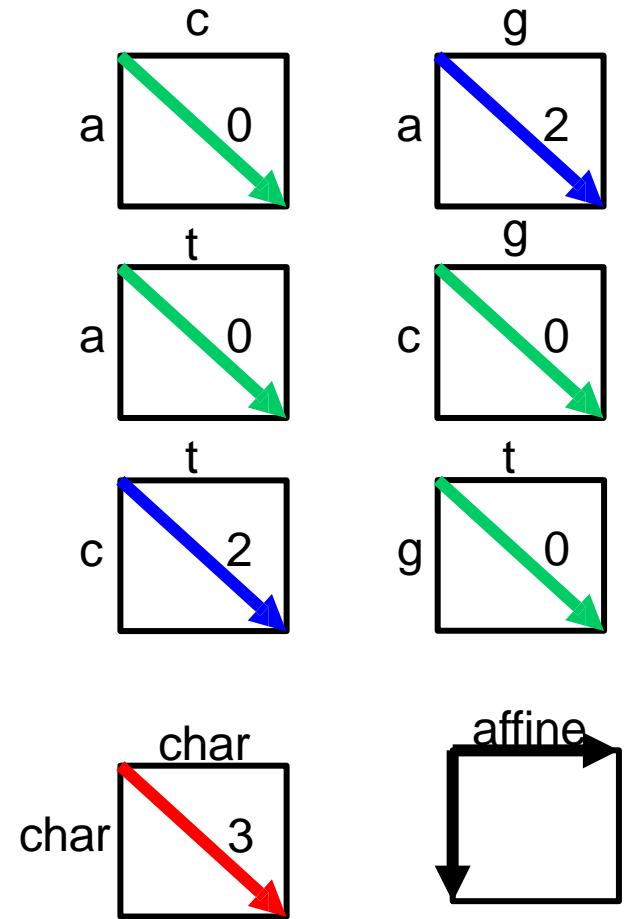
Optimal alignment with affine gap weights and the DNA scoring matrix. Base condition

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓						
t	-1.02 ↓						
c	-1.03 ↓						
t	-1.04 ↓						
g	-1.05 ↓						
a	-1.06 ↓						
t	-1.07 ↓						



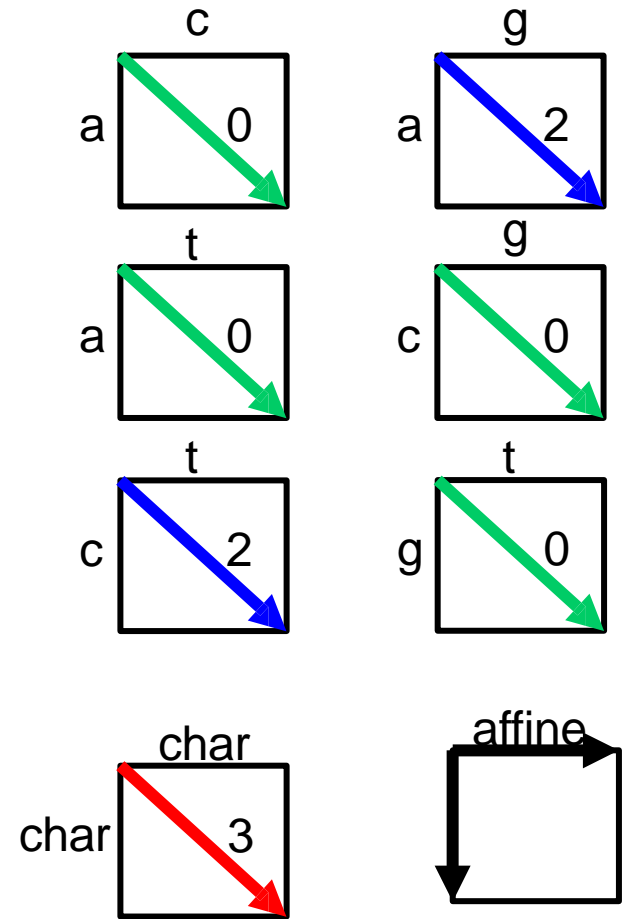
Optimal alignment with affine gap weights and the DNA scoring matrix. Row 1

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓						
c	-1.03 ↓						
t	-1.04 ↓						
g	-1.05 ↓						
a	-1.06 ↓						
t	-1.07 ↓						



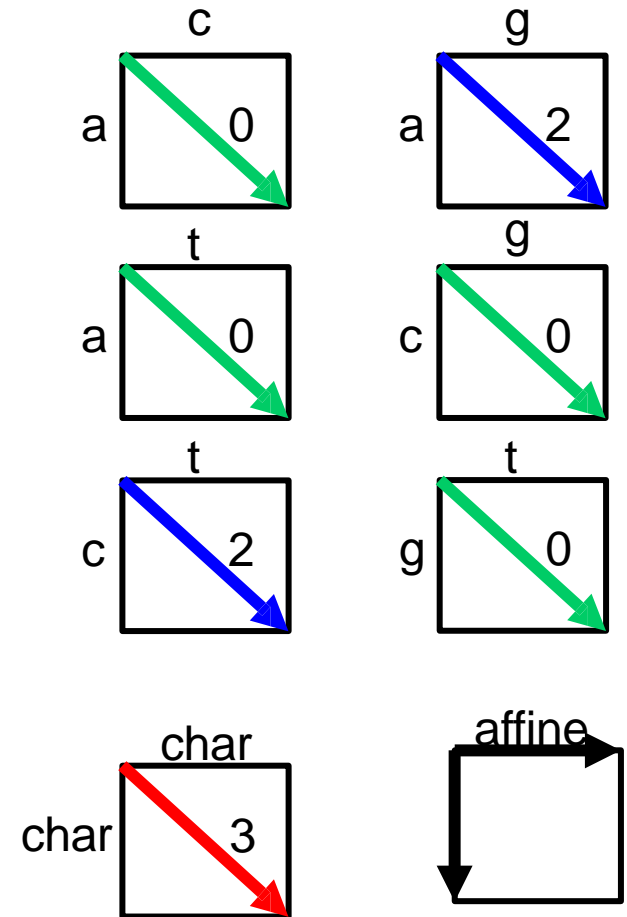
Optimal alignment with affine gap weights and the DNA scoring matrix. Row 2

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓	1.99 ↘	0.98 →	2.99 ↘	1.98 →	4.97 ↘	3.96 →
c	-1.03 ↓						
t	-1.04 ↓						
g	-1.05 ↓						
a	-1.06 ↓						
t	-1.07 ↓						



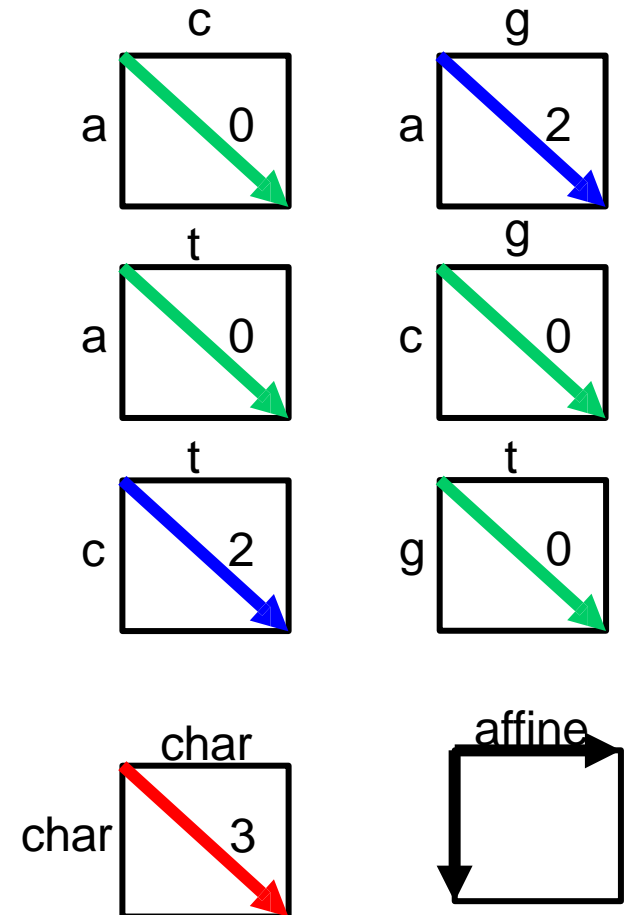
Optimal alignment with affine gap weights and the DNA scoring matrix. Row 3

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓	1.99 ↘	0.98 →	2.99 ↘	1.98 →	4.97 ↘	3.96 →
c	-1.03 ↓	0.88 ↘	1.99 ↘	3.98 ↘	2.97 →	3.98 ↘	4.97 →
t	-1.04 ↓						
g	-1.05 ↓						
a	-1.06 ↓						
t	-1.07 ↓						



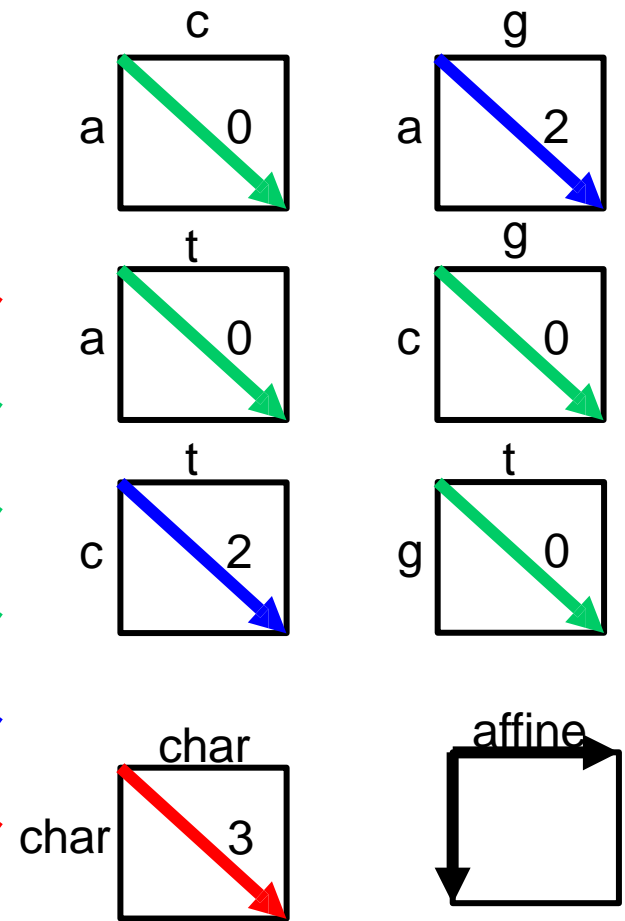
Optimal alignment with affine gap weights and the DNA scoring matrix. Rows 4,5

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓	1.99 ↘	0.98 →	2.99 ↘	1.98 →	4.97 ↘	3.96 →
c	-1.03 ↓	0.88 ↘	1.99 ↘	3.98 ↘	2.97 →	3.98 ↘	4.97 →
t	-1.04 ↓	1.97 ↘	0.98 →	3.99 ↘	3.98 →	5.97 ↘	4.96 →
g	-1.05 ↓	0.96 ↘	4.97 ↘	3.96 →	5.99 ↘	4.98 →	7.97 ↘
a	-1.06 ↓						
t	-1.07 ↓						



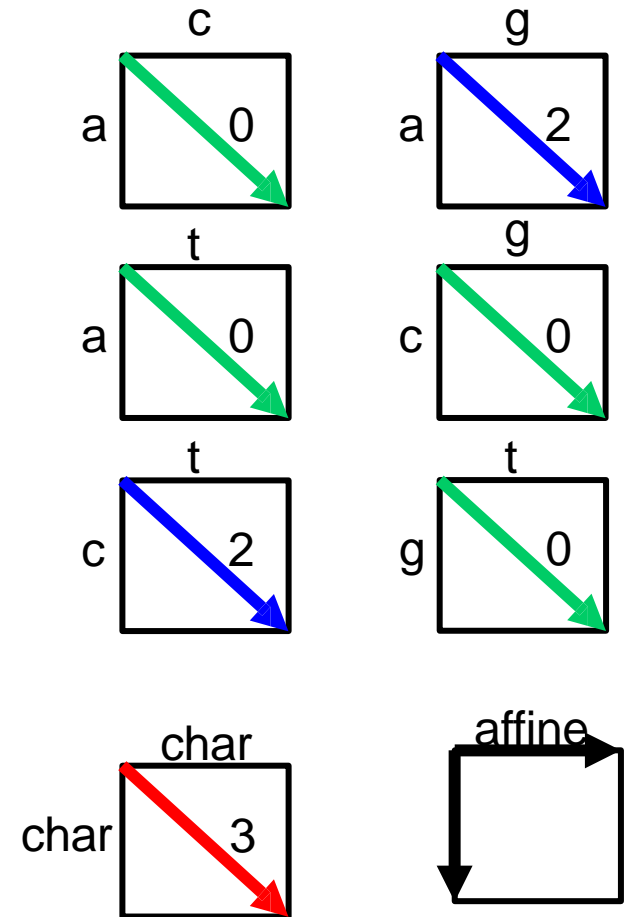
Optimal alignment with affine gap weights and the DNA scoring matrix. Rows 6,7

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓	1.99 ↘	0.98 →	2.99 ↘	1.98 →	4.97 ↘	3.96 →
c	-1.03 ↓	0.88 ↘	1.99 ↘	3.98 ↘	2.97 →	3.98 ↘	4.97 →
t	-1.04 ↓	1.97 ↘	0.98 →	3.99 ↘	3.98 ↘	5.97 ↘	4.96 →
g	-1.05 ↓	0.96 ↘	4.97 ↘	3.96 →	5.99 ↘	4.98 →	7.97 ↘
a	-1.06 ↓	-0.14 ↘	3.96 ↘	4.97 ↘	6.96 ↘	5.99 ↘	7.98 ↘
t	-1.07 ↓	1.94 ↘	2.95 ↘	5.96 ↘	5.95 ↘	9.96 ↘	8.95 →



Optimal alignment with affine gap weights and the DNA scoring matrix. Row 7

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01 ↓	0 ↘	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02 ↓	1.99 ↘	0.98 →	0.99 ↘	0.96 ↓	4.97 ↘	3.96 →
c	-1.03 ↓	0.98 ↓	1.99 ↘	4.98 ↘	3.97 →	3.96 ↓	4.97 ↘
t	-1.04 ↓	1.97 ↘	0.98 →	3.99 ↘	4.98 ↘	6.97 ↘	5.96 →
g	-1.05 ↓	0.96 ↘	4.97 ↘	3.96 →	5.99 ↘	5.96 ↓	8.97 ↘
a	-1.06 ↓	0.95 ↓	3.96 ↘	4.97 ↘	6.96 ↘	5.99 ↘	8.96 ↘
t	-1.07 ↓	1.94 ↘	3.95 ↘	5.96 ↘	5.95 ↓	9.96 ↘	8.95 →



Optimal alignment with affine gap weights and the DNA scoring matrix. Traceback

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01	0 ↙	0.99 ↘	-0.02 →	1.97 ↘	0.96 →	1.95 ↘
t	-1.02	1.99 ↙	0.98 →	0.99 ↘	0.96 ↘	4.97 ↘	3.96 →
c	-1.03	0.98 ↘	1.99 ↘	4.98 ↘	3.97 →	3.96 ↘	4.97 ↘
t	-1.04	1.97 ↙	0.98 →	3.99 ↘	4.98 ↘	6.97 ↘	5.96 →
g	-1.05	0.96 ↘	4.97 ↘	3.96 ↘	5.99 ↘	5.96 ↘	8.97 ↘
a	-1.06	0.95 ↘	3.96 ↘	4.97 ↘	6.96 ↘	5.99 ↘	8.96 ↘
t	-1.07	1.94 ↙	3.95 ↘	5.96 ↘	5.95 ↘	9.96 ↘	8.95 →

The global alignment

	S2	t	g	c	a	t	a
S1	0	-1.01 →	-1.02 →	-1.03 →	-1.04 →	-1.05 →	-1.06 →
a	-1.01	0	0.99	-0.02	1.97	0.96	1.95
t	-1.02	1.99	0.98	0.99	0.96	4.97	3.96
c	-1.03	0.98	1.99	4.98	3.97	3.96	4.97
t	-1.04	1.97	0.98	3.99	4.98	6.97	5.96
g	-1.05	0.96	4.97	3.96	5.99	5.96	8.97
a	-1.06	0.95	3.96	4.97	6.96	5.99	8.96
t	-1.07	1.94	3.95	5.96	5.95	9.96	8.95

This alignment is called **global** since it represents an alignment with the best overall cost for the entire strings S1 and S2

S1	a ↓	t ↓	c ↓	t ↘	g ↘	- →	a ↘	t ↘	- →
S2	-	-	-	t	g	c	a	t	a

The local alignment

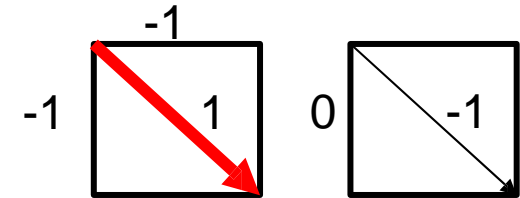
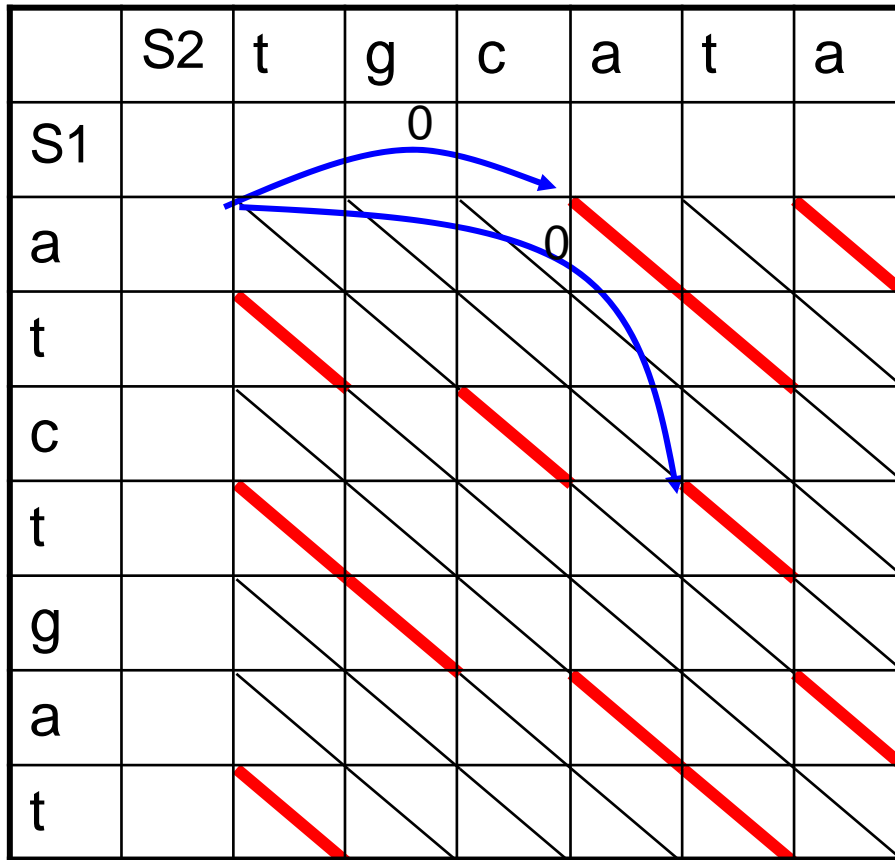
- The similarity of biological strings rarely extends through the entire length of these strings
- Example: homeodomain of the homeobox genes is a very conserved substring in overall very different sequences
- How to detect the regions of **local** similarity?

The local alignment problem

- Find a pair $(S1[i1\dots i2], S2[j1\dots j2])$ of substrings of $S1$ and $S2$ such that the global alignment score between these substrings is maximal among all possible pairs of substrings of $S1$ and $S2$
- In terms of paths in edit graph: find the path with the best cost **between any pair of vertices**

The solution to the local alignment problem.

Simple scoring example



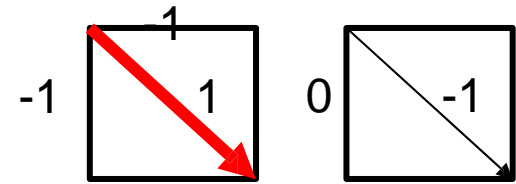
When choosing the best move through the next cell, take into account an additional possibility to start from vertex (0,0) with an overall 0-cost

This means that if the cost of some path drops below 0, we abandon this path and restart the cost to find a better local path starting from the current position.

The local alignment.

Base condition

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0						
t	0						
c	0						
t	0						
g	0						
a	0						
t	0						



The local alignment. Recurrence relation

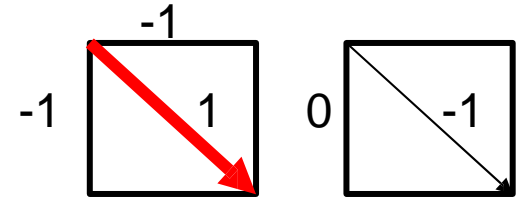
$$\text{COST}(i,j)=\max \begin{cases} 0 \\ \text{COST}(i-1,j) + \text{gapCost} \\ \text{COST}(i,j-1) + \text{gapCost} \\ \text{COST}(i-1,j-1)+\text{score}(S1[i], S2[j]) \end{cases}$$

The cost never drops below 0.

If it is negative, we start a new path from the same point with a cost 0

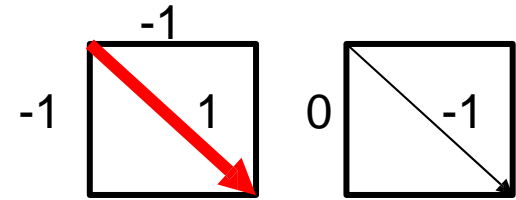
The local alignment. Row 1

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0						
c	0						
t	0						
g	0						
a	0						
t	0						



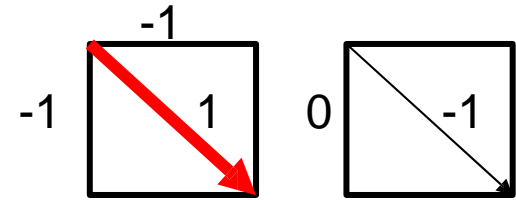
The local alignment. Row 2

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0						
t	0						
g	0						
a	0						
t	0						



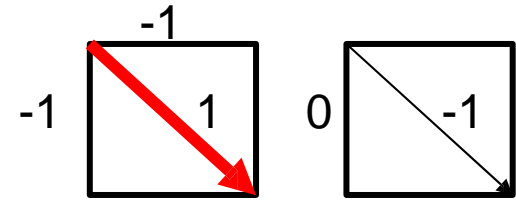
The local alignment. Row 3

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0						
g	0						
a	0						
t	0						



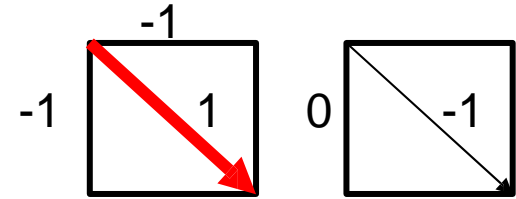
The local alignment. Row 4

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0	1	0	0	0	1	0
g	0						
a	0						
t	0						



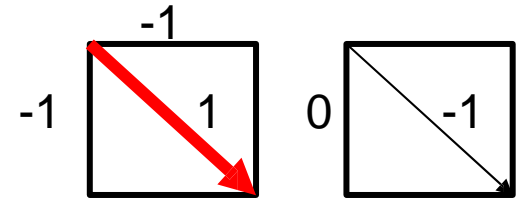
The local alignment. Row 5

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0	1	0	0	0	1	0
g	0	0	2	1	0	0	0
a	0						
t	0						



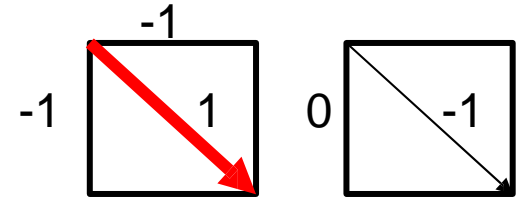
The local alignment. Row 6

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0	1	0	0	0	1	0
g	0	0	2	1	0	0	0
a	0	0	1	1	2	1	1
t	0						



The local alignment. Row 7

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0	1	0	0	0	1	0
g	0	0	2	1	0	0	0
a	0	0	1	1	2	1	1
t	0	1	0	0	1	3	2



Most similar local pattern

	S2	t	g	c	a	t	a
S1	0	0	0	0	0	0	0
a	0	0	0	0	1	0	1
t	0	1	0	0	0	2	0
c	0	0	0	1	0	1	1
t	0	1	0	0	0	1	0
g	0	0	2	1	0	0	0
a	0	0	1	1	2	1	1
t	0	1	0	0	1	3	2

S1	a	t	c	t	g	-	a	t	-
S2	-	-	-	t	g	c	a	t	a

Alignments: Running time

- $O(NM)$
- If we want to find the regions of high similarity between a new sequence of size M and all G genes of size N each in the database, we need to perform $O(MNG)$ operations