Topic 6: Sequence Alignment II

```
ATALVALNIN  ALVALLN


ATALVALNIN
A-–LVALL-N


ATALVAL-NIN
--ALVALLN--
```

Sequence Alignment

The sequence alignment problem:

- we have a pair of strings (sequences)
- we want to find the best way to "match” the two

Examples:

- `leaf` and `please`
- `ATCCA` and `TCGA`
- `MPPDNE` and `PADQE`

Sequence Alignment

Dot plots

	l	e	a	f
p				
l				
e				
a				
s				
e				

	T	C	G	A
A				
T				
C				
C				
A				

Sequence Dot Plots

	l	e	a	f
p				
l	*			
e		*		
a			*	
s				
e		*		

	T	C	G	A
A				*
T	*			
C		*		
C		*		
A				*

Sequence Dot Plots + windowing

	l	e	a	f
p				
l	*			
e		*		
a			*	
s				
e		*		

	T	C	G	A
A				*
T	*			
C		*		
C		*		
A				*

Sequence Alignment

The sequence alignment problem:

- you have 2 strings (sequences)
- find the ”best match” between the two

Ex: GCCAT and GAAT

```
GCCAT
+ooo
GAAT
```

A **+** means a match
A **o** means a mismatch

This alignment is probably not the best.

Sequence Alignment

Ex: GCCAT and GAAT

```
GCCAT
+ooo
GAAT
```

A **+** means a match
A **o** means a mismatch

```
GCCAT
 oo++
 GAAT
```

This alignment is better.

Can we do even better
if we allow gaps?

Sequence Alignment with gaps

```
GCCAT
+-o++
G_AAT
```

A **+** means a match
A **o** means a mismatch
A **–** is a gap _

```
GCCAT     GCCAT     GCCA_T
+o-++     +oo-+     +--+-+
GA_AT     GAA_T     G__AAT
```

What we want is the best - an ”optimal” - alignment

Sequence Alignment Algorithm

```
GCCAT
  oo++
  GAAT
```

We want the ***optimal alignment*** of two strings.

Can we go through all the possible alignments and pick the best one?

If both sequences are N bases long, and gaps are not allowed, the best techniques take $c\text{N}^2$ time steps (c is a constant)

Optimal Alignment w/o gaps

Optimal alignment: If both sequences are N bases long, and gaps are not allowed, the best techniques take cN^2 time steps.

For large values of N, the ”*c*” is not important and we say that the ungapped alignment has a time complexity of N^2.

Optimal Alignment with gaps

Optimal alignment: For sequences that are N bases long when gaps **are** allowed, the best techniques have a ”time complexity” of $\mathbf{N^{4N}}$.

As N becomes large, the time needed to find the best alignment becomes very large very quickly!

Finding the optimal sequence alignment problem is a hard problem to solve – so hard that we abandon our quest for the best and settle for something that is ”good enough”.

"Acceptable" Alignment with gaps

The optimal sequence alignment problem is hard: a ”brute-force” search will take too long as sequence lengths increase.

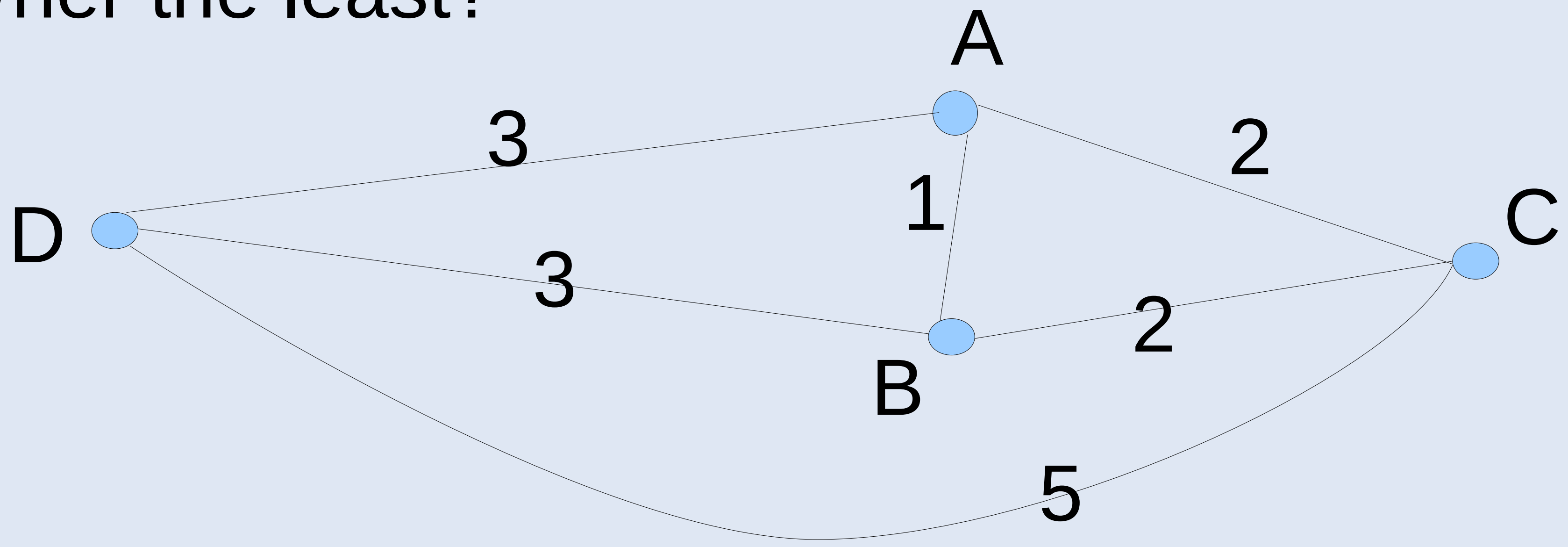
Instead, we use methods that can give us reasonable – good enough - alignments but not necessarily the ”optimal” one.

There are many such ”heuristics” in computer science.

Traveling Salesman

Another problem that is hard to solve is the Traveling Salesman problem:

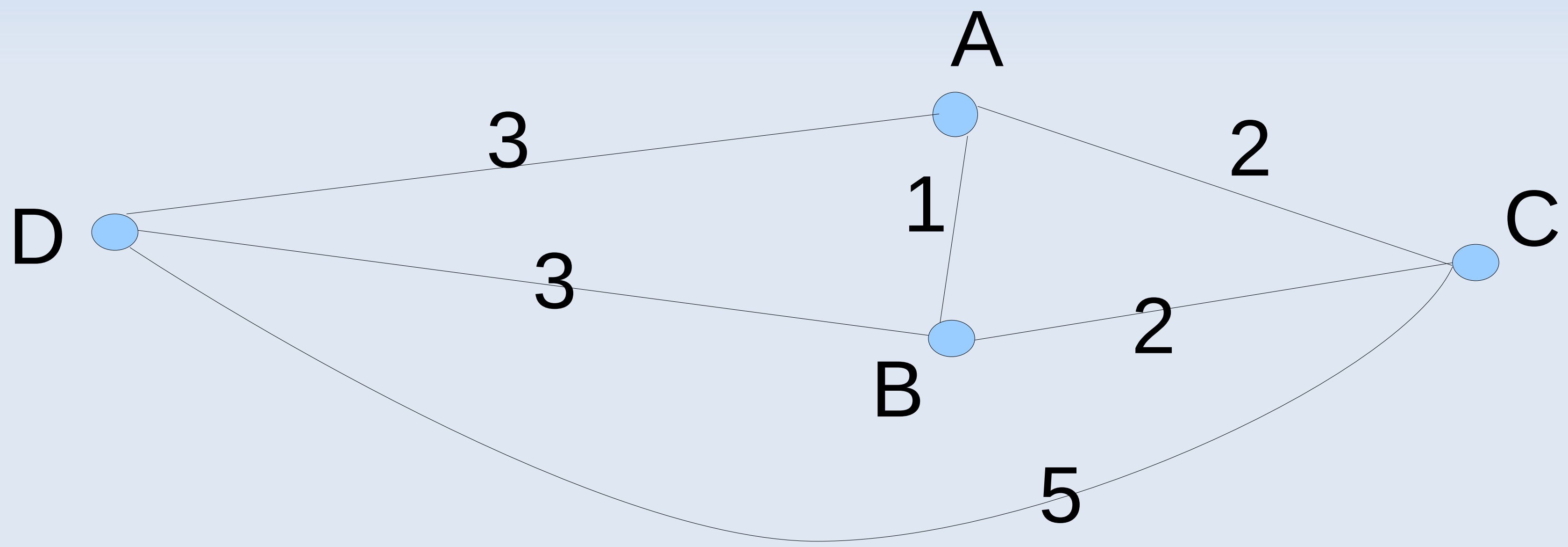
A salesperson wants to visit a number of cities, A-D, returning to the starting city having been to each of the other cities only once; what route will cost him/her the least?



"Cost" can be time, money, etc.

Traveling Salesman

Try a heuristic: start at a city, find the nearest city, and repeatedly visit the closest next city that we have not yet seen, ending back in the starting city



This heuristic is called a ”Greedy” algorithm

Greedy Traveling Salesman

Start with at A, find the nearest city → B; cost = 1

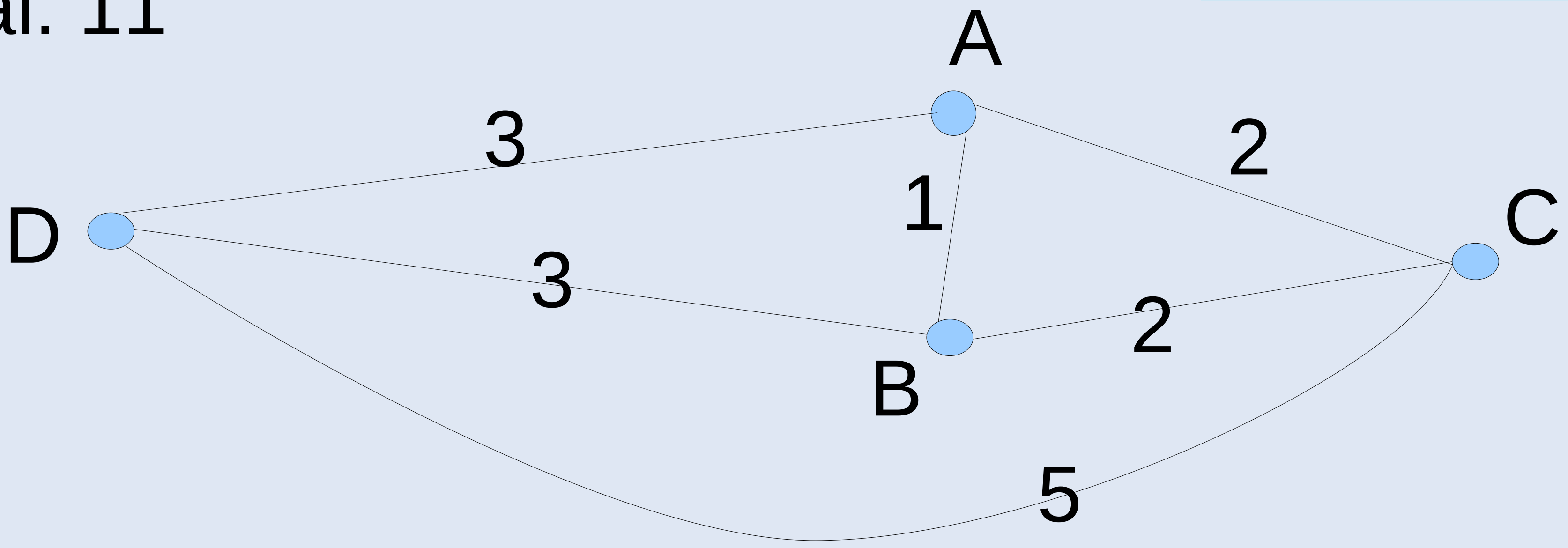
closest next city → C; cost = 2

closest next one → D; cost = 5

And back to A; cost = 3

”Greedy” method:
Find the nearest neighb

Total: 11



Optimal Solution Traveling Salesman

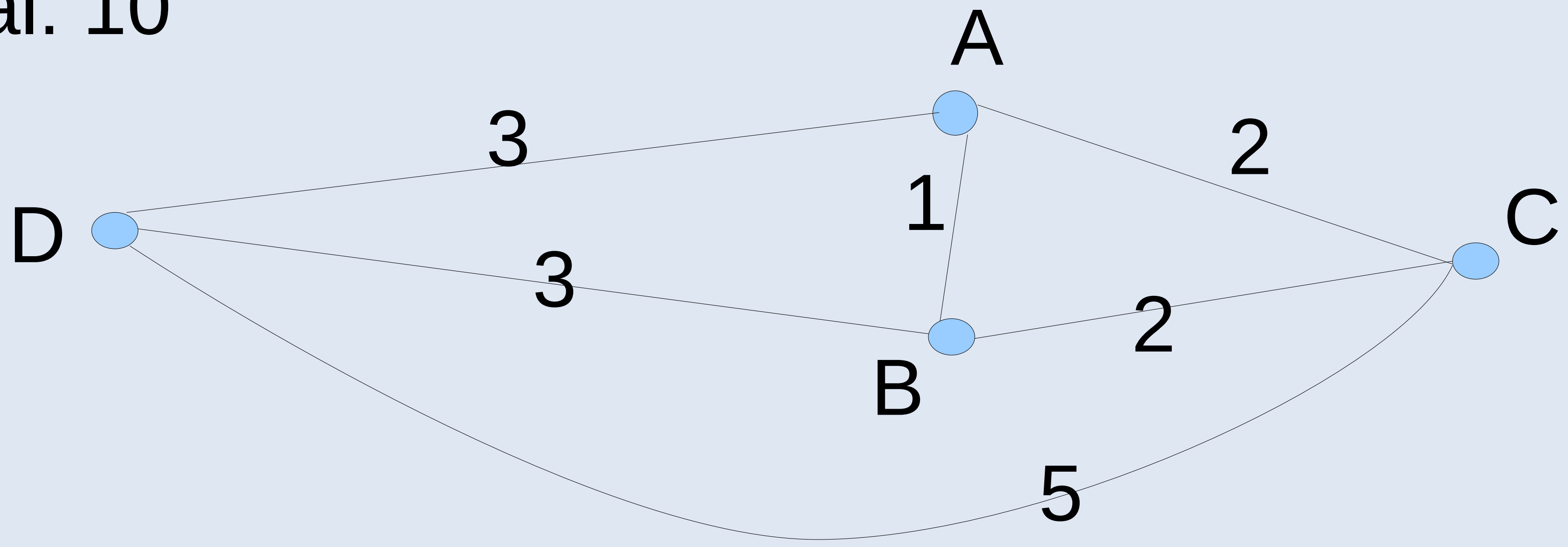
Start from A, go to D; cost = 3

next city → B; cost = 3

next one → C; cost = 2

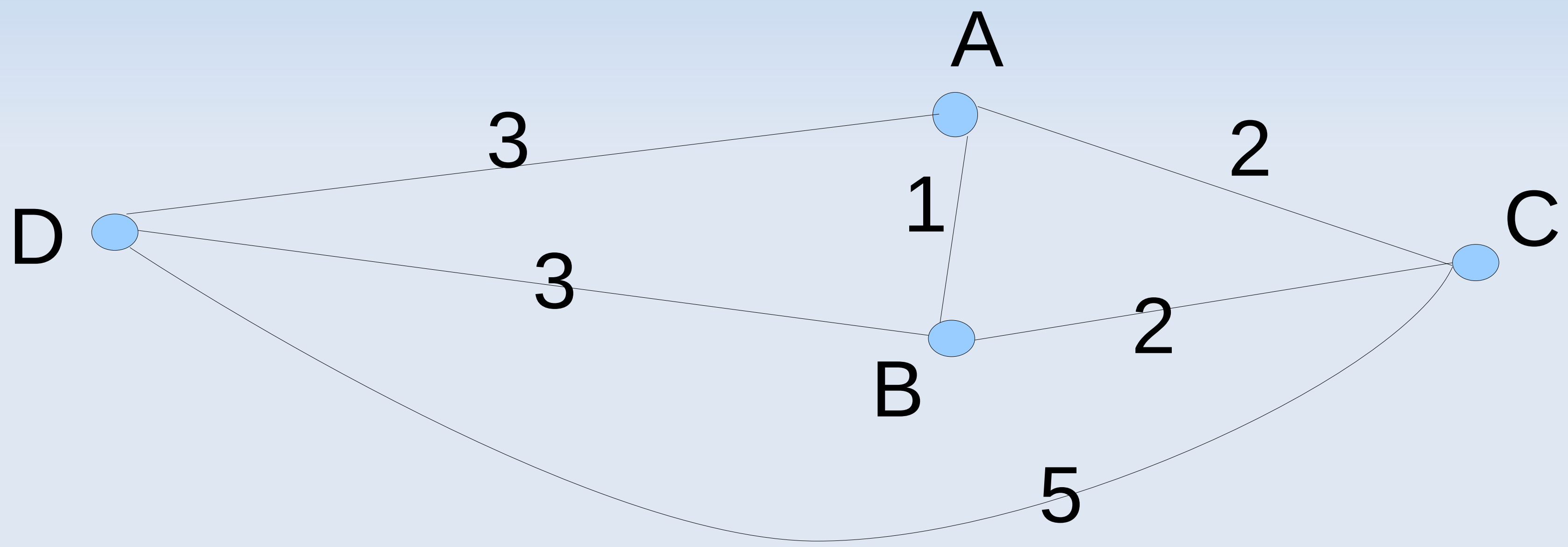
And back to A; cost = 2

Total: 10



What makes the Traveling Salesman Problem difficult

Knowing the optimal solution for N-1 cities (ABC), does not help with the optimal solution for N cities.



→ The greedy solution may not be the optimal one

"Acceptable" Alignment with gaps

We want a reasonably good alignment – may not be the optimal one

Dynamic Programming Strategy: Try to find an optimal solution by by finding optimal solutions to smaller subproblems.

- i.e. break a big problem into a bunch of smaller problems and ”remember” the best previous solutions.

If a problem satisfies some properties, DP or *greedy* can find the optimal solution.

Dynamic Programming (DP) Dot Plot Approach

We will use a matrix like the one we used for dot plots:

TGA

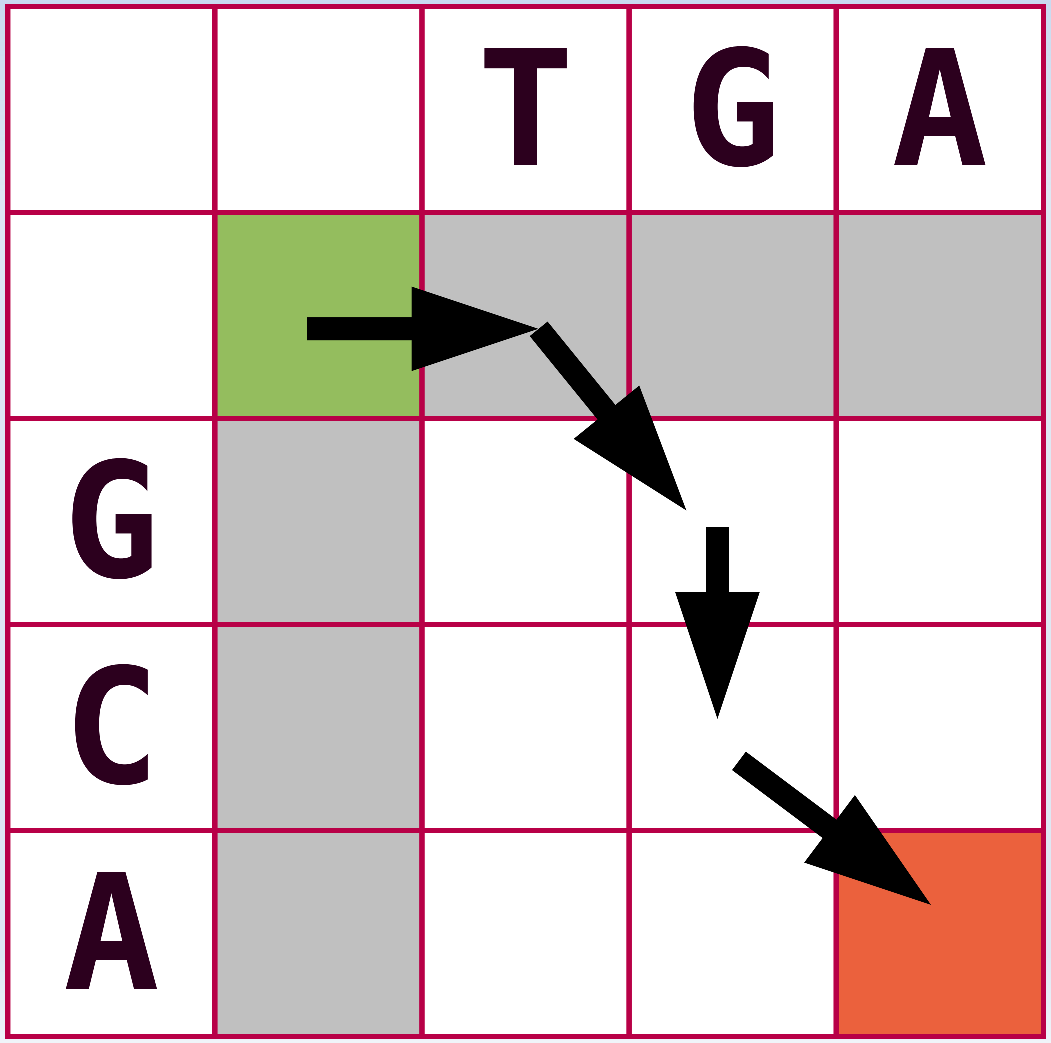
GCA

		T	G	A
	🟩	▨	▨	▨
G	▨			
C	▨			
A	▨			🟥

DP Path

We will end up with a **path** from the upper-left corner to the lower-right corner. This path represents an alignment:

```
TG_A
-+-+
_GCA
```



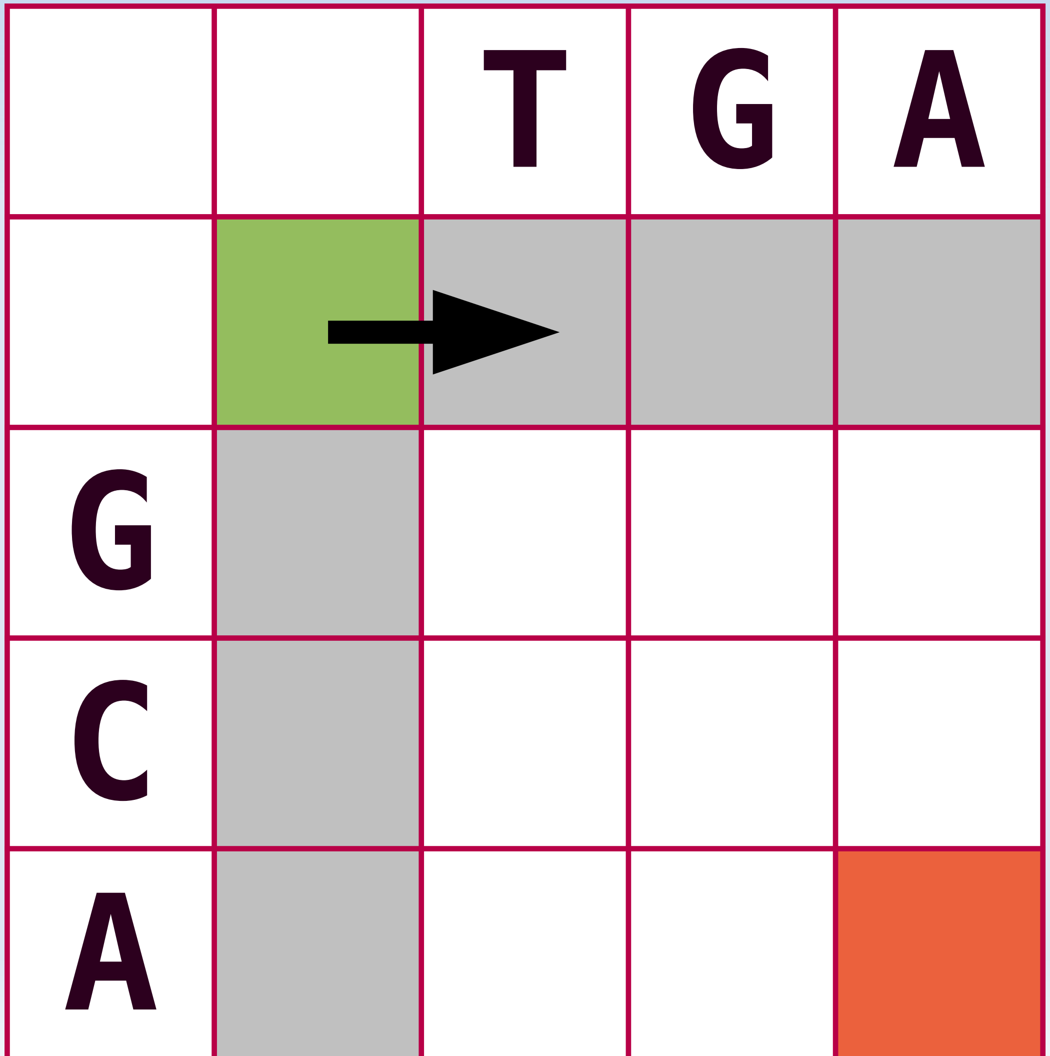
DP Path

Path through the matrix: a horizontal step along a row is a gap (`-`) in the seq on the leftmost column

T

-

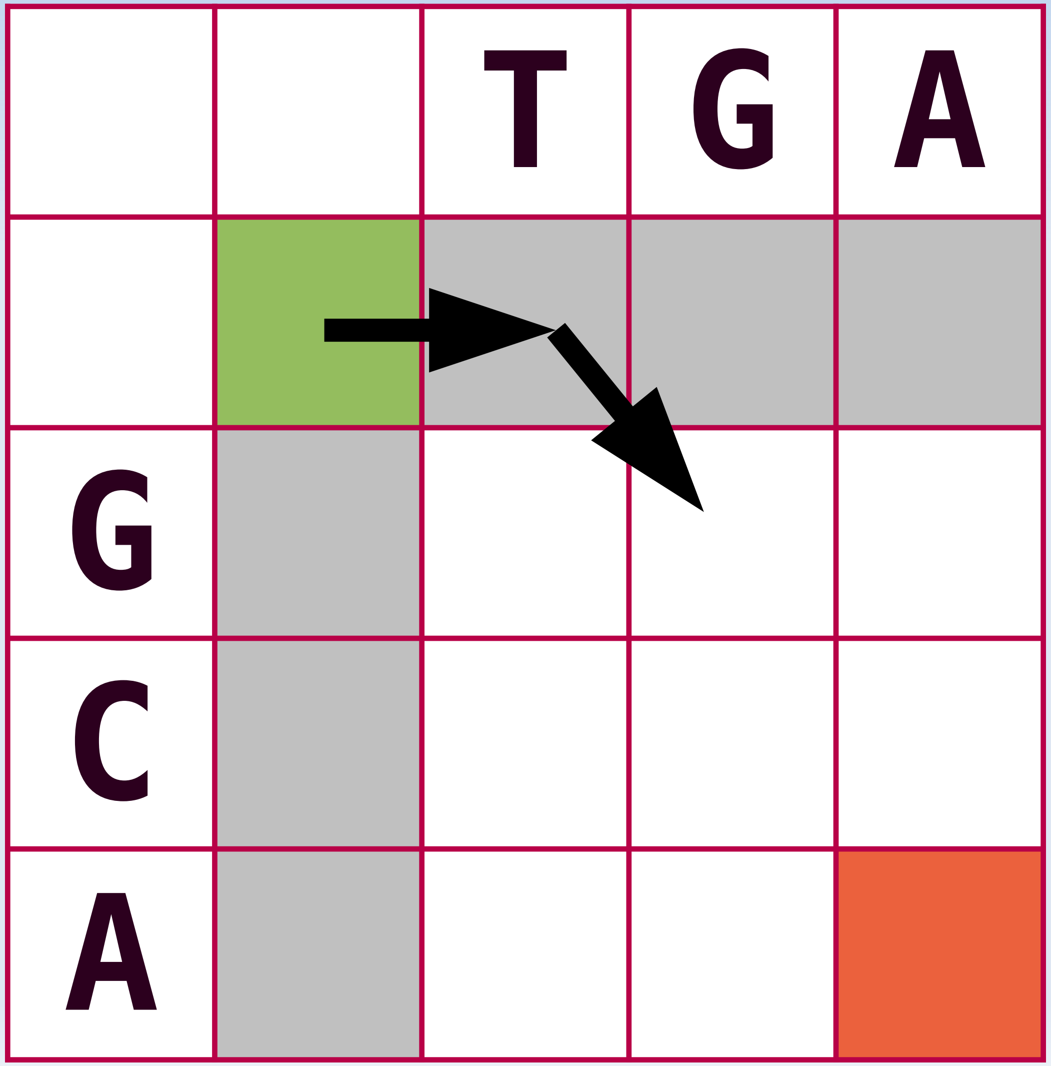
—



DP Path

Path through the matrix: a diagonal step is an alignment (+) of one base in each sequence:

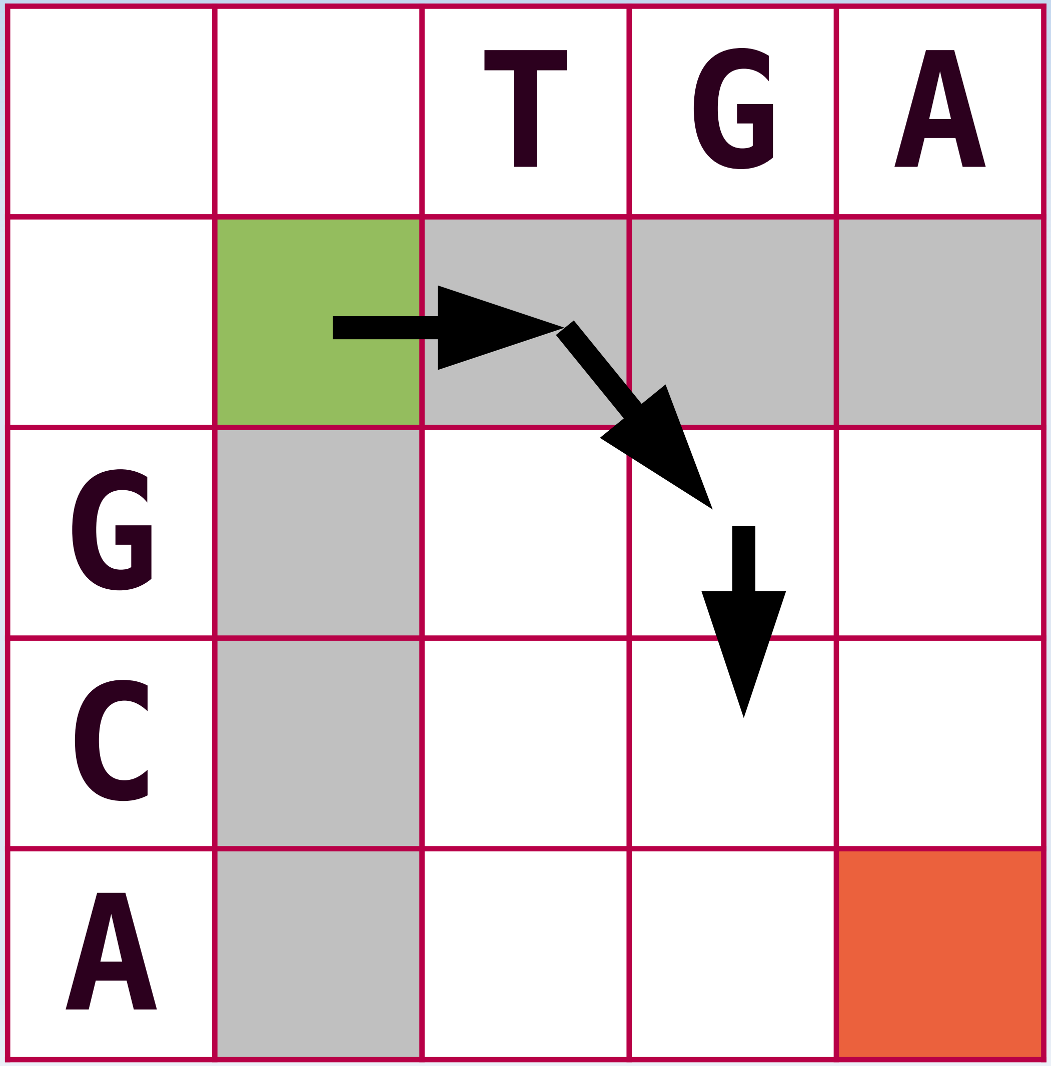
```
TG
-+
_G
```



DP Path

Path through the matrix: a vertical step along a column is a gap (-) in the seq in the top row

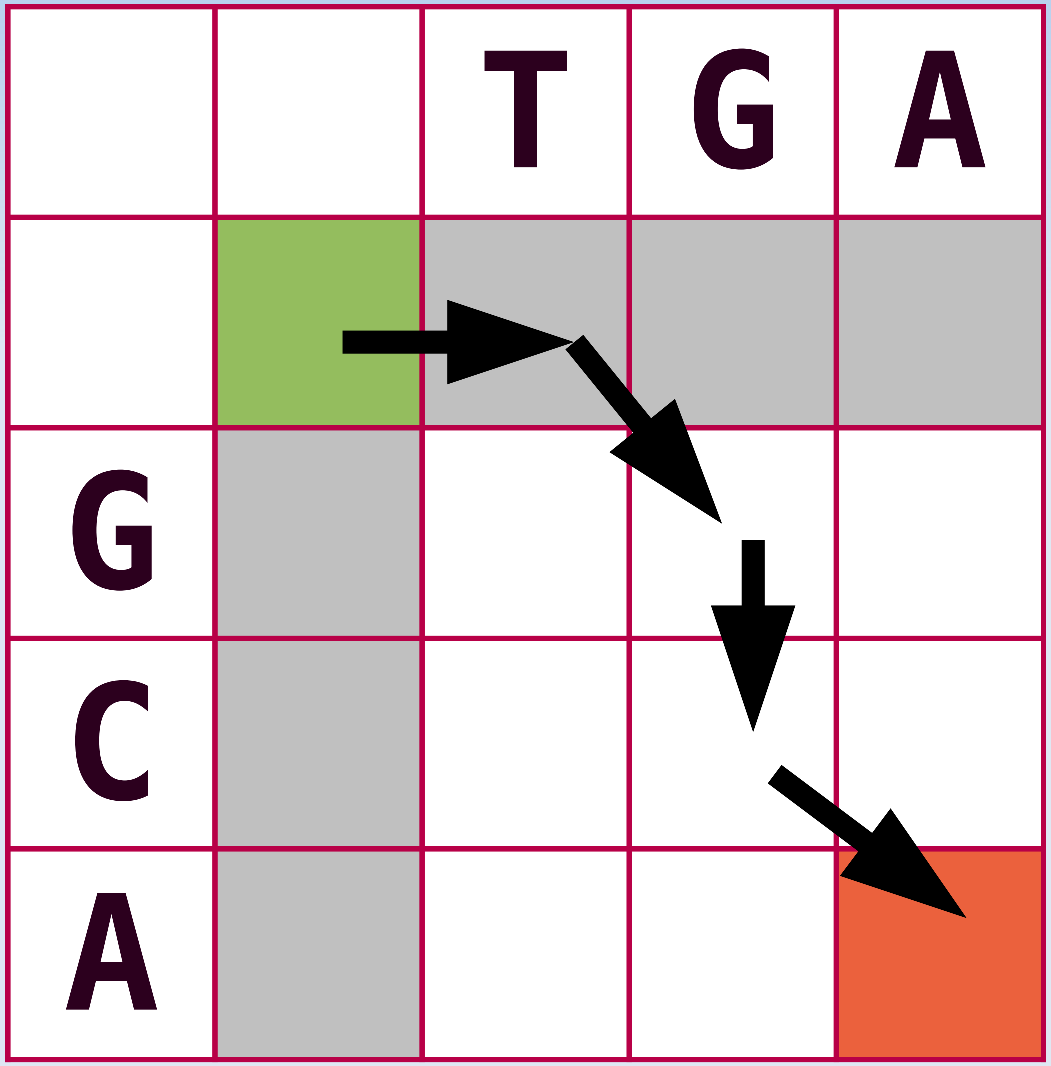
```
TG_
-+-
_GC
```



DP Path

The **path** through the matrix represents an alignment

```
TG_A
-+-+
_GCA
```



At each point in the path, we have to choose the best of 3 choices based on the path so far and the nucleotides in the current row and column.

Sequence Alignment Problem

Suppose we have the sequences

GCCAT

GAAT

We want to find a ”good” alignment of these two sequences using a Dynamic Programming algorithm

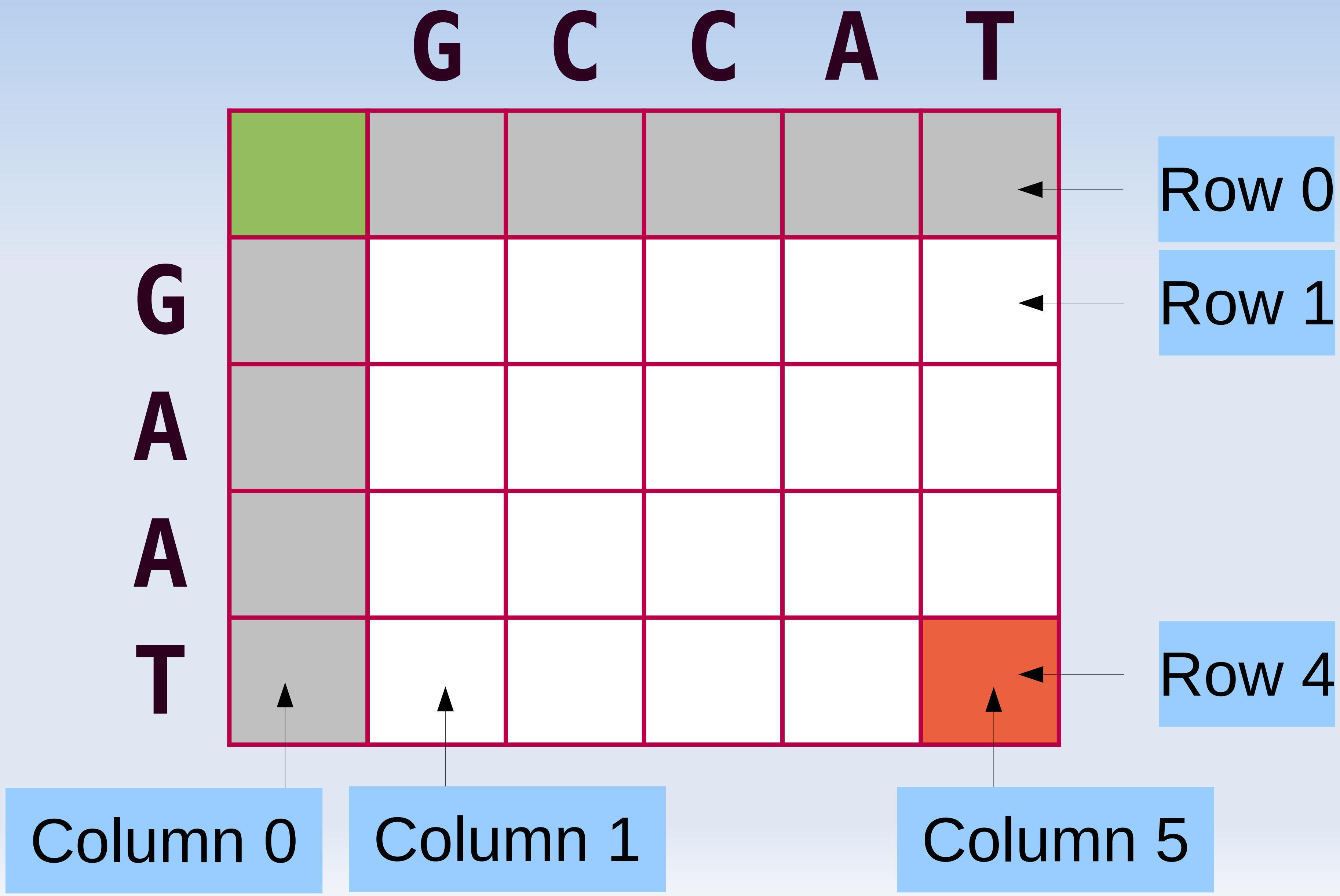
DP Path algorithm Step 1

Set up the matrix

		G	C	C	A	T
	🟩	▨	▨	▨	▨	▨
G	▨					
A	▨					
A	▨					
T	▨					🟧

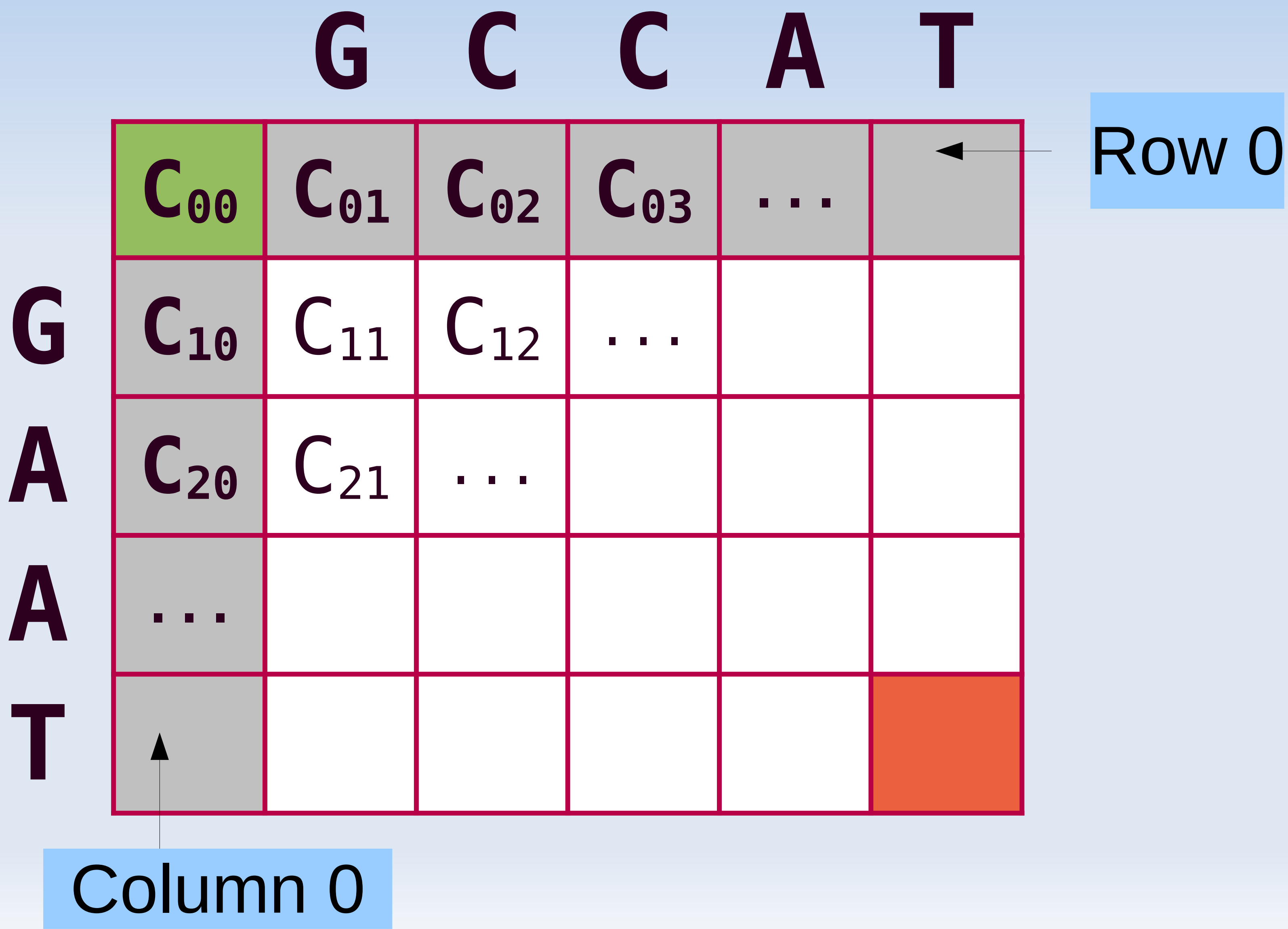
DP Path algorithm Step 1

Set up the matrix



DP Path algorithm Step 1

Set up the matrix


G C C A T
G A A T
C_{00} C_{01} C_{02} C_{03} ...
C_{10} C_{11} C_{12} ...
C_{20} C_{21} ...
...
Row 0
Column 0

DP Path algorithm Step 2

Set row 0 (C_{0j}) and column 0 (C_{i0}) elements to 0

		G	C	C	A	T
	0	0	0	0	0	0
G	0	C_{11}	C_{12}	...		
A	0	C_{21}	...			
A	0					
T	0					🟧

Row 0

Column 0

DP Path algorithm Step 3

Set row 1 (C_{1j}) elements from left to right

		G	C	C	A	T
	0	0	0	0	0	0
G	0	C_{11}	C_{12}	. . .		
A	0					
A	0					
T	0					🟥

$\leftarrow$ Row 1

DP Path algorithm Fill rule

Fill the matrix based on the cells to the left, above and diagonally up and left

		G	C	C	A	T
	0	0	0	0	0	0
G	0					
A	0		$C_{i-1,j-1}$	$C_{i-1,j}$		
A	0		$C_{i,j-1}$	C_{ij}		
T	0					🟥

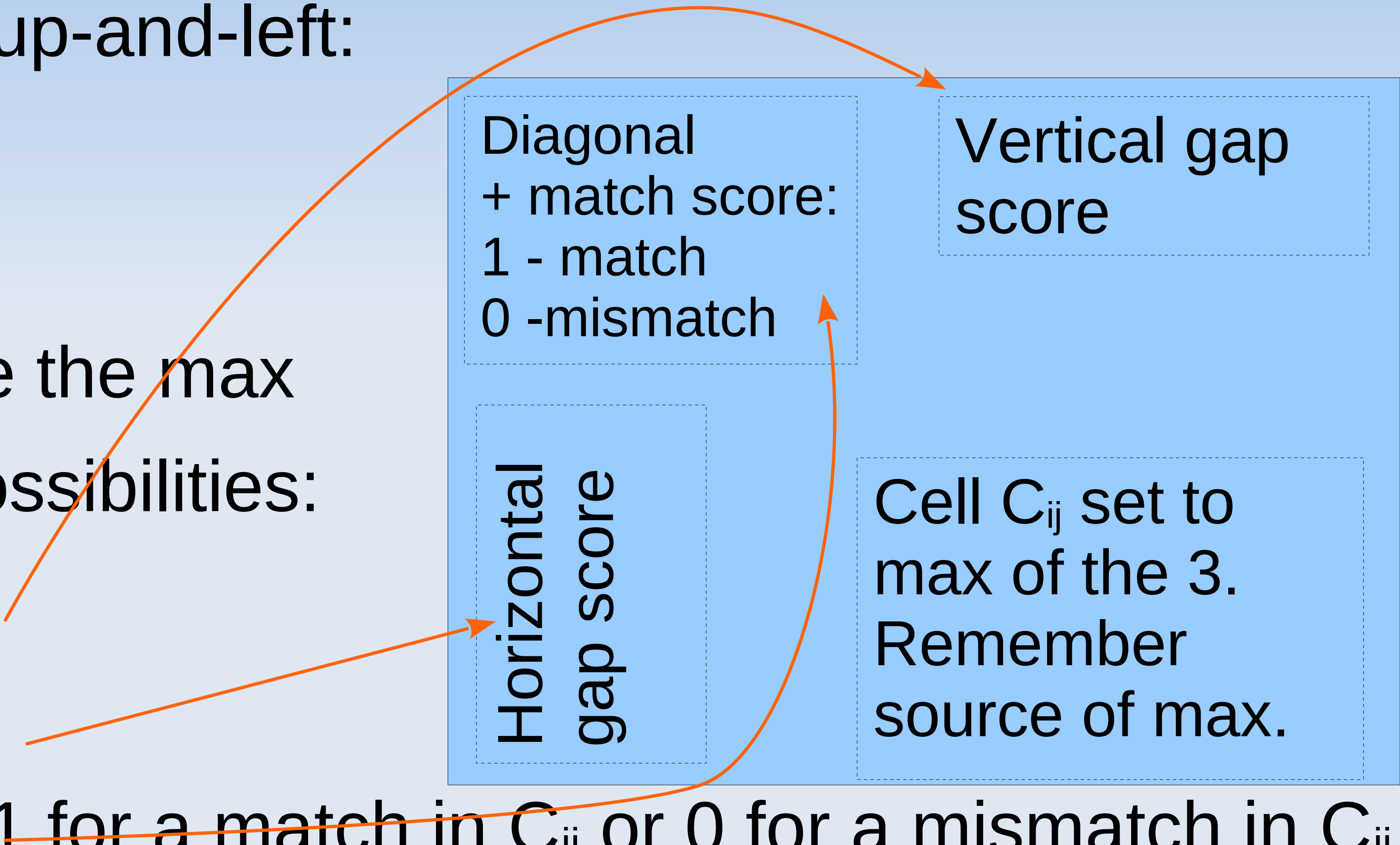
DP Path algorithm Fill rule

Fill the matrix based on the cells to the left, above and diagonally up-and-left:

C_{ij} set to be the max

of three possibilities:

1. $C_{(i-1)j}$ - 1
2. $C_{i(j-1)}$ - 1
3. $C_{(i-1)(j-1)}$+ 1 for a match in C_{ij} or 0 for a mismatch in C_{ij}



DP Path algorithm Fill rule

Example:

		A
	4	5
T	3	4 4 2 ?

		T
	4	5
T	3	5 4 2 ?

Gap scores

▪ Horizontal	3 – 1	3 – 1
▪ Vertical	5 – 1	5 – 1
Diagonal	4 + 0 (mismatch)	4 + 1

DP Path algorithm Step 3

Set row 1 (C_{1j}) elements from left to right:

to the max of C_{0j}- 1, C_{1j-1}- 1, or C_{0j-1}+ (1 if C_{1j} = match)

For C_{11}, the

3 values are

C_{01}- 1 = -1

C_{10}- 1 = -1

C_{00}+ 1 = 1

		G	C	C	A	T
	0	0	0	0	0	0
G	0	C_{11}	C_{12}	...		
A	0					
A	0					
T	0					🟧

DP Path algorithm Step 3

For C_{12}, what are the 3 values?

C_{02}- 1 = ?

C_{11}- 1 = ?

C_{01}+ ? = ?

Max = ?

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	C_{12}	...		
A	0					
A	0					
T	0					🟥

DP Path algorithm Step 3

Finish the row from left to right:

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0					
A	0					
T	0					🟥

DP Path algorithm Step 4

Similar step for row 2 from left to right:

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0					
T	0					🟧

DP Path algorithm Step 5

Row 3 from left to right:

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0					🟥

DP Path algorithm Step 6

Row 4 from left to right:

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

Done with the ”Fill” phase

DP Path algorithm Traceback phase

Work from lower-right corner back to upper-left

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

Choose the best cell to the left, up, and diagonally; in this case all are 1 → try all 3!

DP Path algorithm Traceback phase

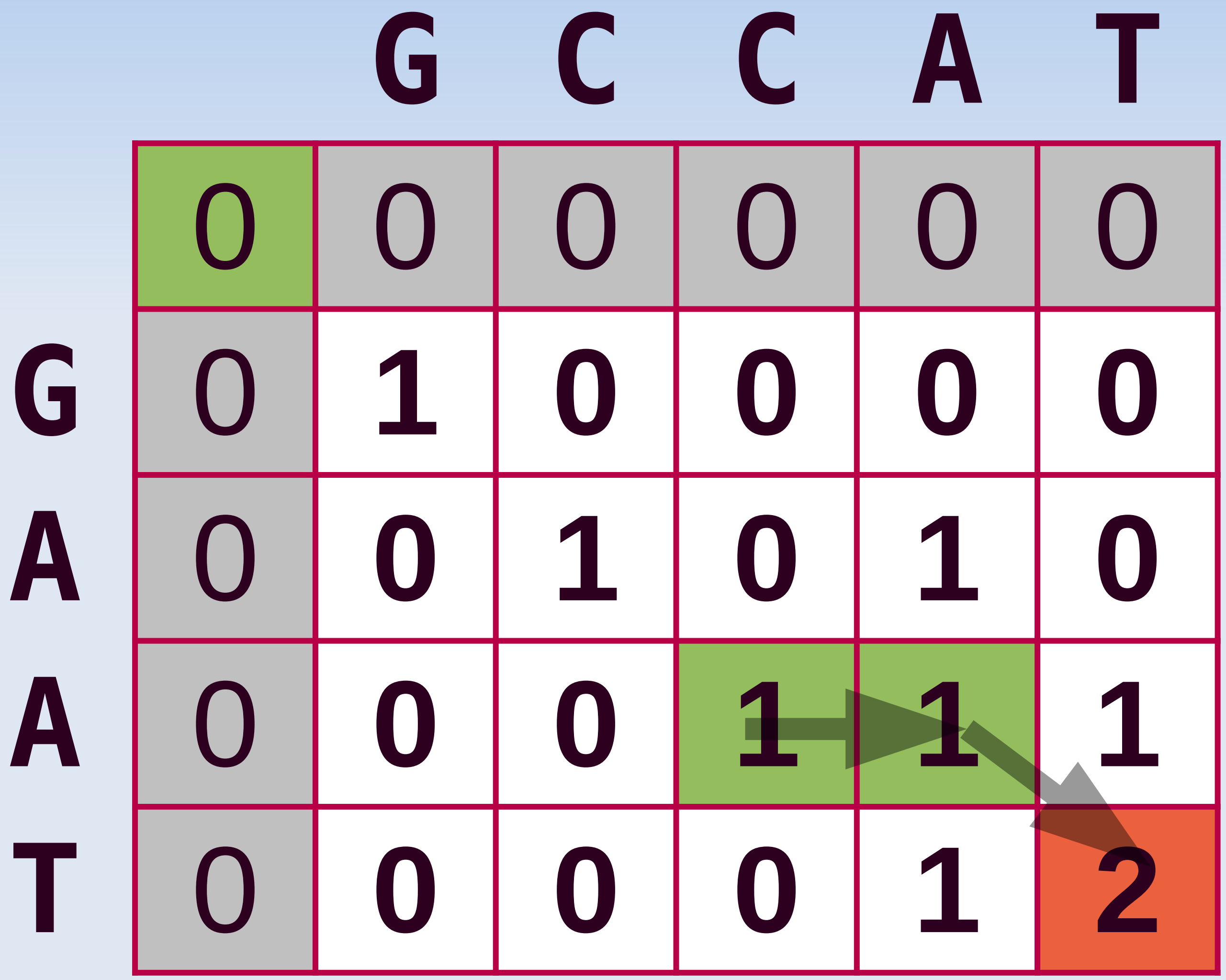
Work from lower-right corner back to upper-left

		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

Turns out the diagonal one is best

DP Path algorithm Traceback phase

Work from lower-right corner back to upper-left



		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

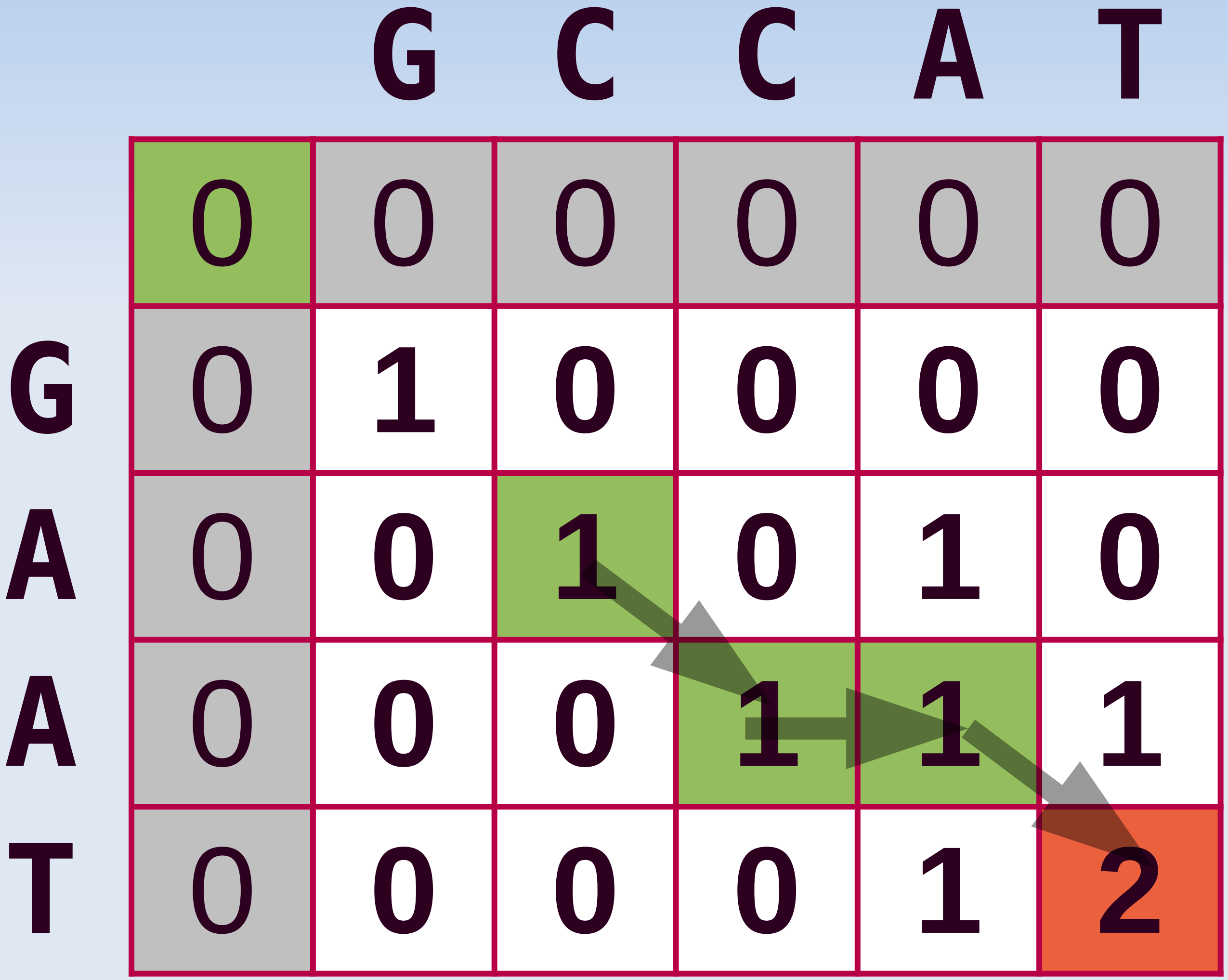
Two possibilities

turns out the

left one is best

DP Path algorithm Traceback phase

Work from lower-right corner back to upper-left



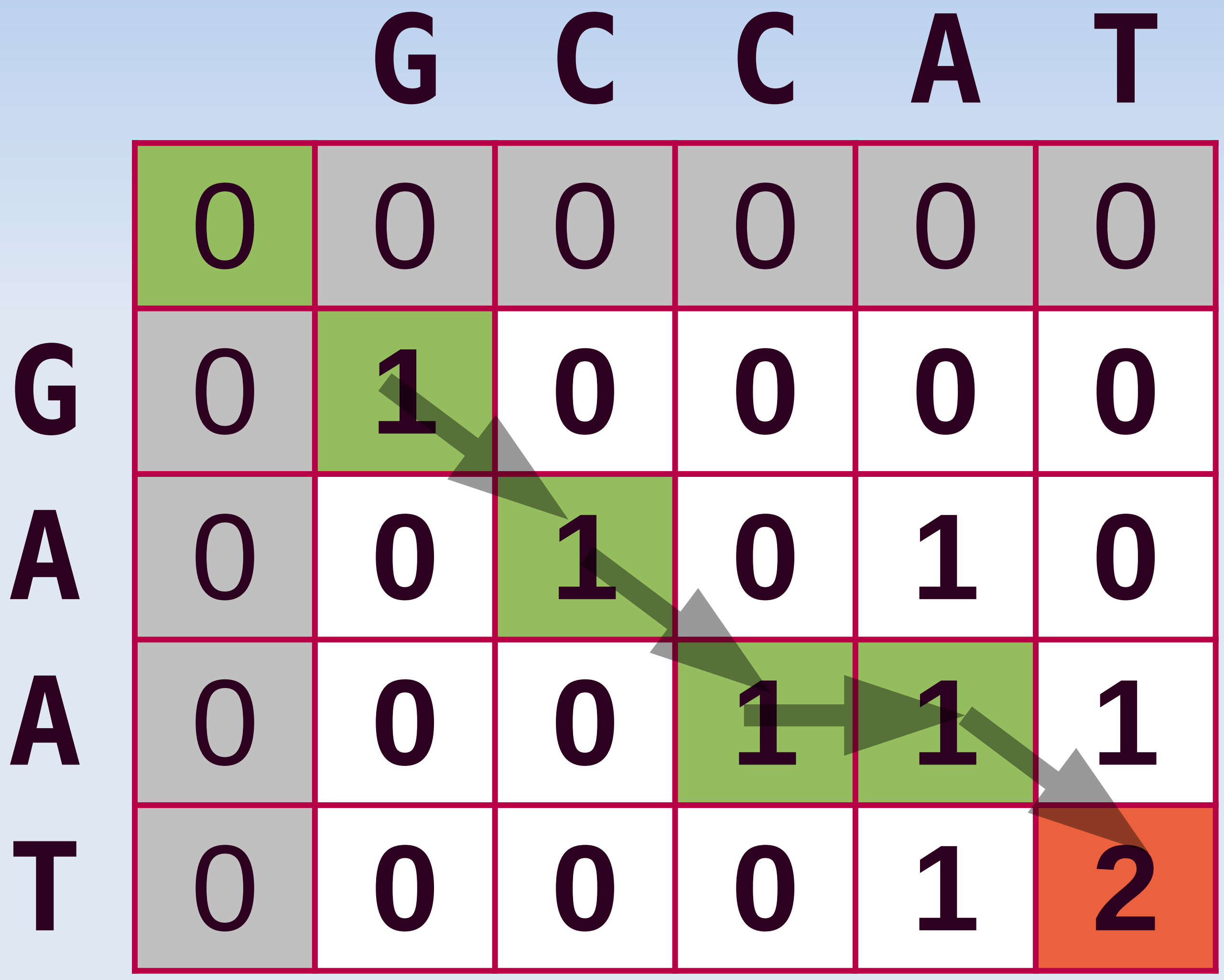
		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

One best choice

 - diagonal

DP Path algorithm Traceback phase

Work from lower-right corner back to upper-left



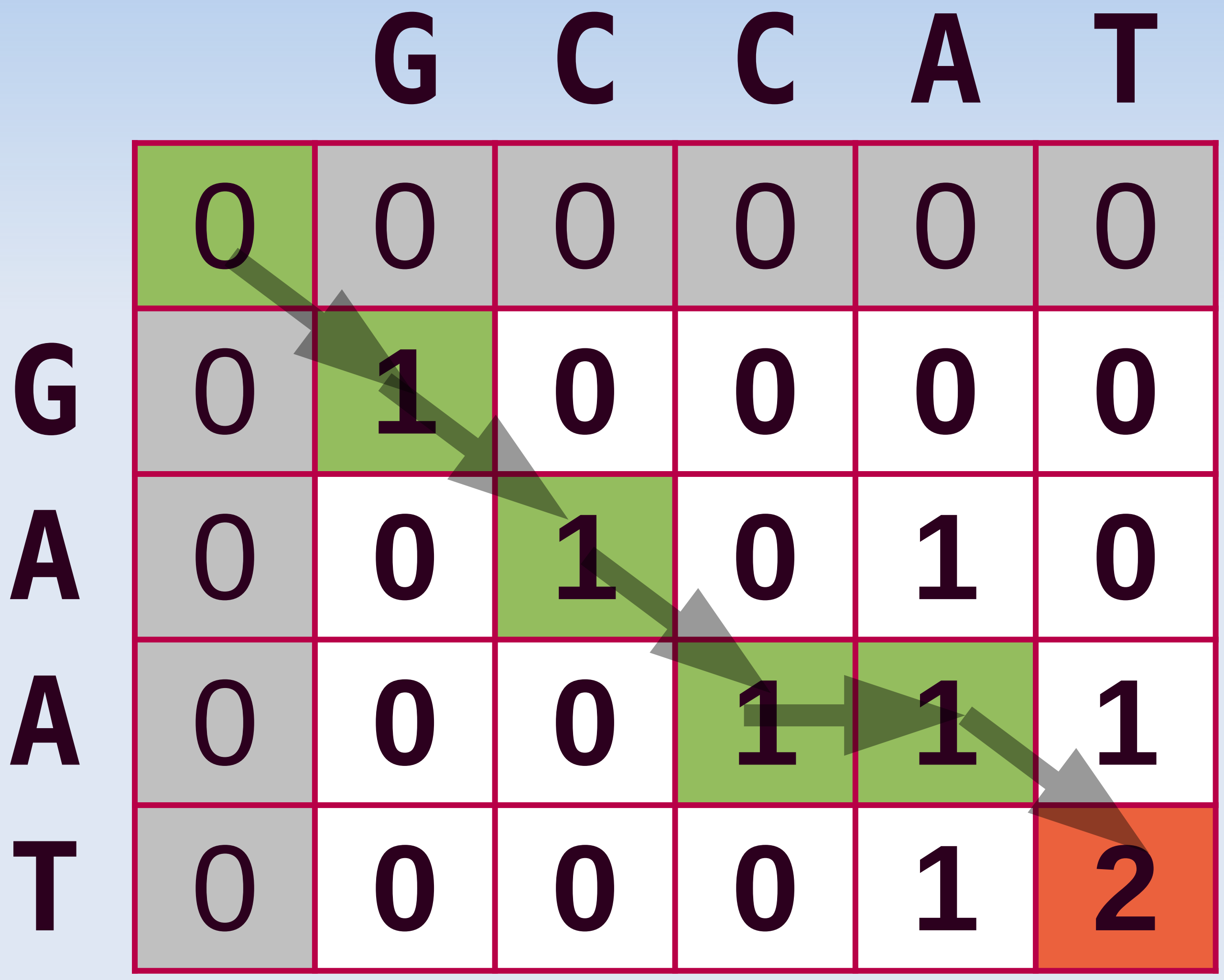
		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

One best choice

- diagonal

DP Path algorithm Traceback phase

Work from lower-right corner back to upper-left



		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

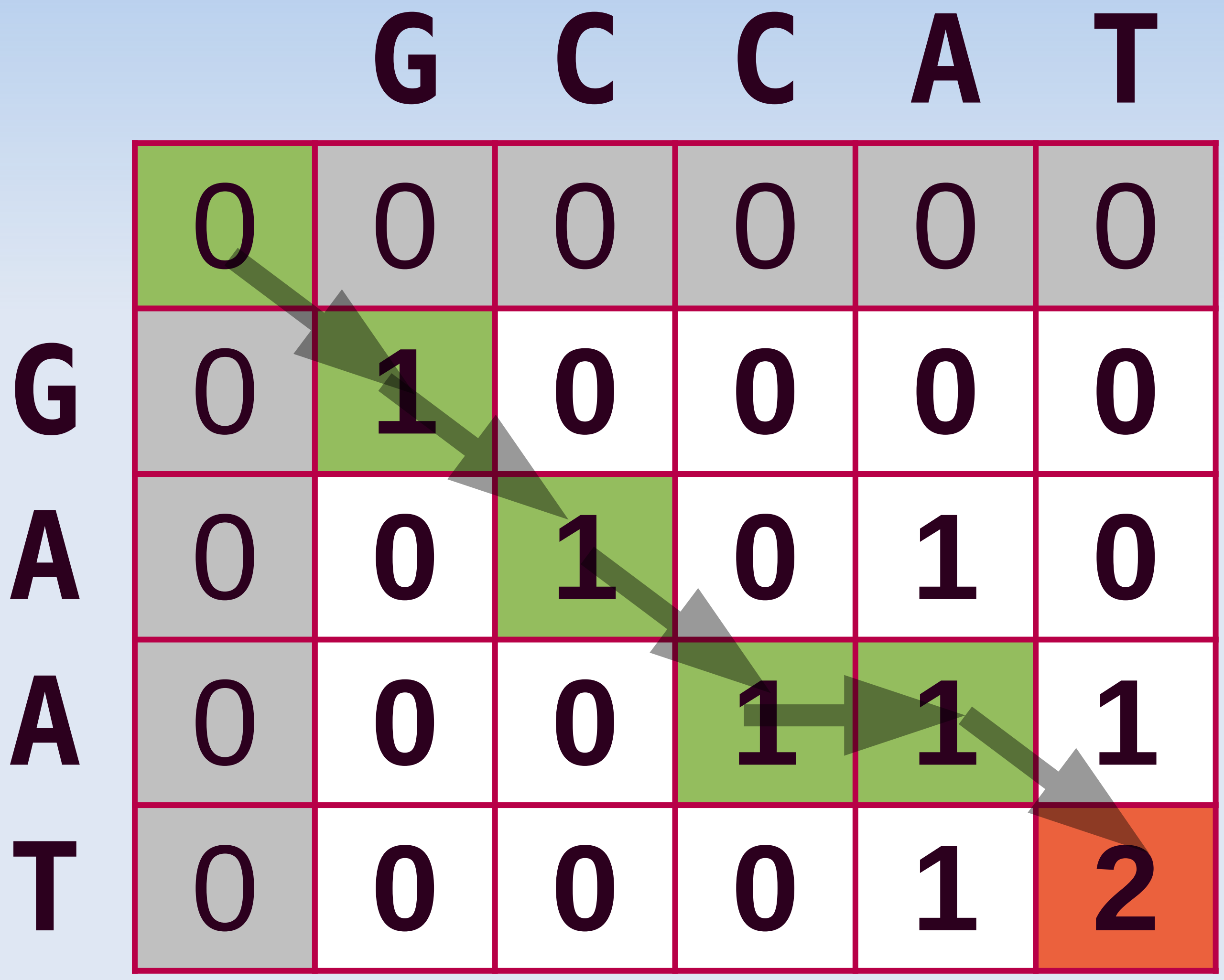
Last step is also a diagonal

DP Path algorithm - Align

Resulting path gives us this alignment:

```
GCCAT
+oo-+
GAA_T
```

Probably not optimal: one gap, two mismatches



		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

DP Path alternative Step 2

Set row 0 (C_{0j}) and column 0 (C_{i0}) elements to penalize origination gaps

		G	C	C	A	T
	0	-1	-2	-3	-4	-5
G	-1	C_{11}	C_{12}	∵		
A	-2	C_{21}	∵			
A	-3					

Sequence Alignment Problem

The dynamic programming approach attempts to maximize the score of an alignment.

An alignment score depends on the scoring matrix and gap penalty – these can be combined into one if the gap penalty is a constant:

	A	T	C	G	–
A	1	0	0	0	-1
T	0	1	0	0	-1
C	0	0	1	0	-1
G	0	0	0	1	-1
–	-1	-1	-1	-1	0

We can also use an ”affine gap penalty”: one penalty for starting a gap and an additional penalty for continuing it.

Sequence Alignment - Alternative

We could also phrase the sequence alignment problem as the **minimum distance** between two sequences – the Edit Distance approach.

Not considered here but very interesting approach.

Sequence Alignment – Big Picture

We are trying to align sequences to see how closely related they may be: ”sequence similarity”.

Percent identity is a good measure of how close 2 sequences are. We may also want to measure the differences.

Similar sequences are usually related due to a common ancestor.

The main forces that changes sequences are:

mutations, genetic drift, and natural selection

Homologous Sequences

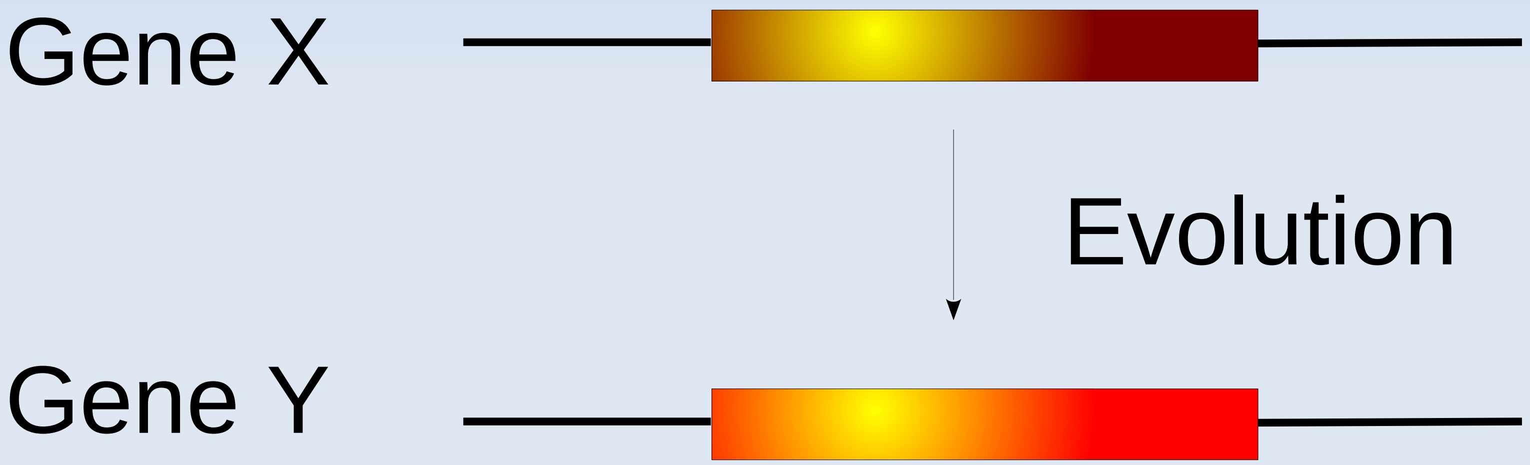
Sequences with significant similarity are said to be homologous

Homologs – two or more sequences possibly from different species and related by descent from a single common ancestral sequence.

Homologs can be orthologous or paralogous.

Orthologs

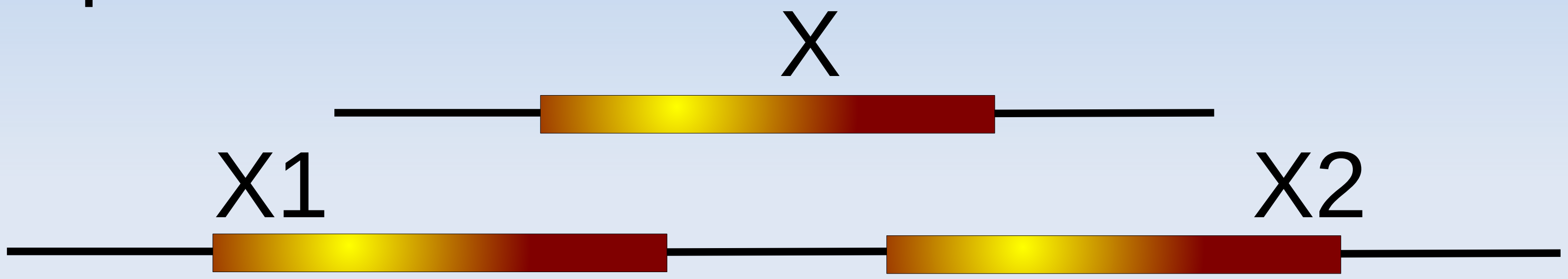
Homologous genes with the same function in two different species, evolved from a common ancestral gene by speciation



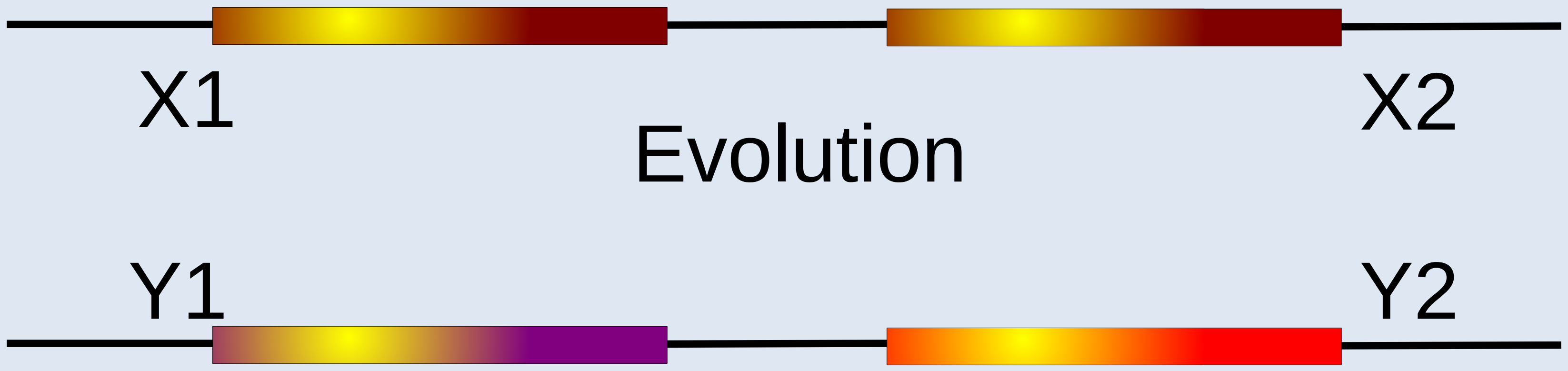
Likely to have similar protein sequence → similar structure → similar biological function

Paralogs

Homologous genes with similar function in two species, evolved from a common ancestral gene by gene duplication



followed by changes due to evolution:



E.g.: Apple MYB genes MYB10 and MYB110a appear to be paralogs that control fruit color. The Apple plant likely had a whole genome duplication event ~65 mya (Velasco 2010)

Mutations

Mutations in genomic DNA sequences occur when proteins responsible for copying DNA make mistakes. The three main kinds of mutations are:

Substitutions – one base is replaced by another

Insertions – a base is inserted into a sequence

Deletion – a base is deleted from a sequence

Substitution Mutations

Substitutions can occur in 3 ways. Eg:

GCCAT	GCCAT	GCCAT
GCGAT	GCAAT	GCTAT

Adenine and guanine are purines

Cytosine and thymine are pyramidines

A **transition** substitution is a replacement of a purine by a purine or a pyramidine by a pyramidine

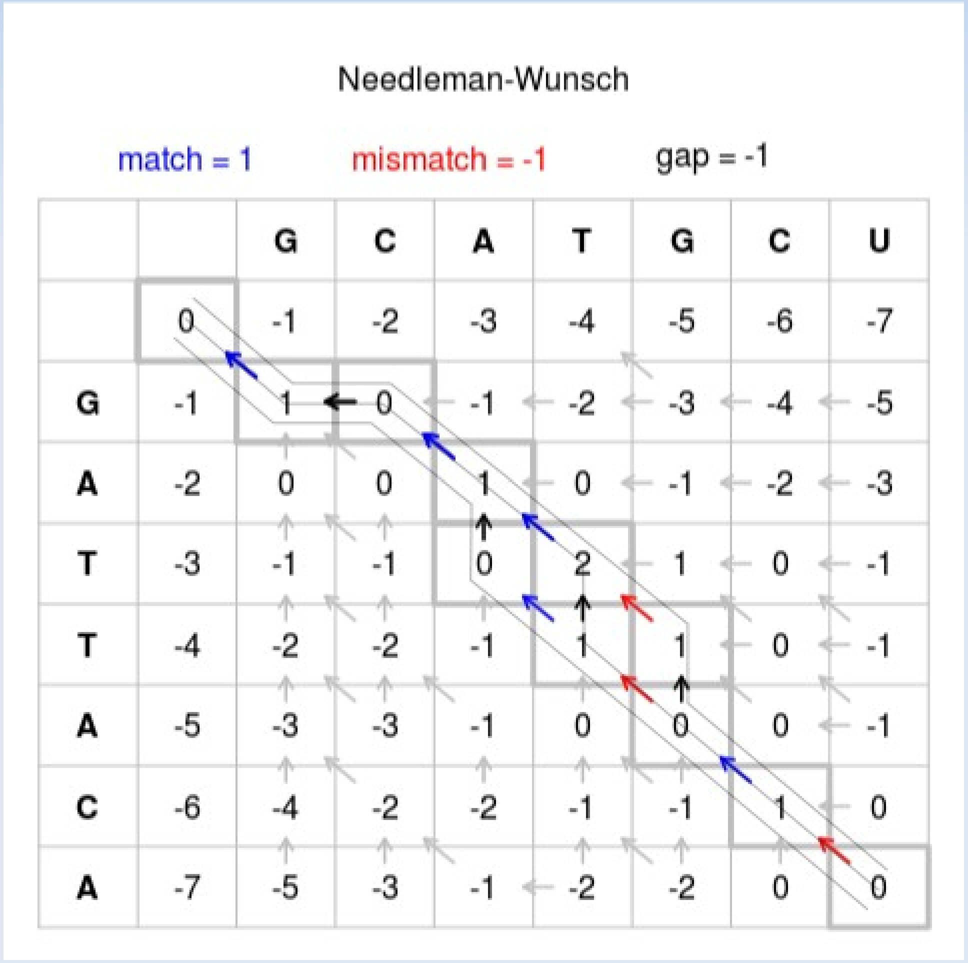
A **transversion** is a replacement of a purine by a pyramidine or a pyramidine by a purine.

Kinds of Alignments

Global alignment – Needleman-Wunsch algorithm

Scoring matrix:

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1



Needleman-Wunsch

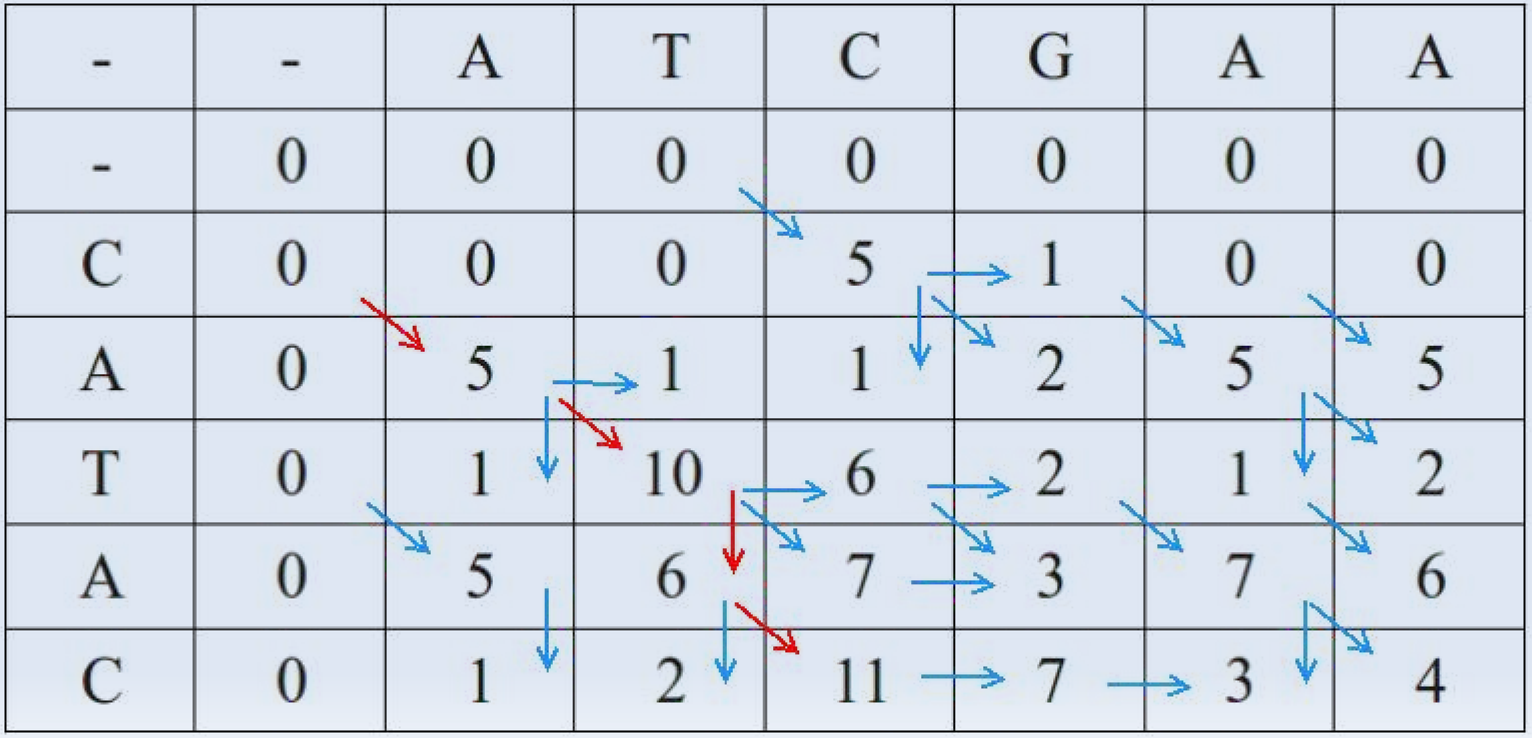
match = 1 mismatch = -1 gap = -1

		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

Kinds of Alignments

Local alignment – **Smith-Waterman** algorithm, a variation of Needleman-Wunsch:

- Replace negative scores with zero
- Starts the traceback at the highest scoring cell and ends at a 0 → highest scoring local alignment



-	-	A	T	C	G	A	A
-	0	0	0	0	0	0	0
C	0	0	0	5	1	0	0
A	0	5	1	1	2	5	5
T	0	1	10	6	2	1	2
A	0	5	6	7	3	7	6
C	0	1	2	11	7	3	4

Protein Alignments

Protein alignments are the same except that the scoring matrix is a 20 by 20 matrix and may also have weights based on physicochemical and biological properties of amino acids:

- Cys/Pro are important for structure and function
- Trp has a large side chain
- Lys/Arg are positively charged

Which residues can substitute for another without affecting protein function:

- Ile/Val are small and hydrophobic
- Ser/Thr are polar

BLAST

- A BLAST alignment is like the Smith-Waterman (SW) alignment
- SW is ”too slow” → BLAST uses an algorithm that is 50X faster
- BLAST alignments are close to SW but may not be as good as SW

Outline of BLAST

1. Remove low-complexity areas of query
2. Divide query up into words and for each word
    1. find good matches in database sequences
    2. create a search tree from high-scoring words
    3. extend matches to high-scoring segment pairs
3. List the best HSPs

   Compute the E-values of HSPs
4. Combine HSPs into a longer alignment
5. Show Smith-Waterman local alignments of query with database sequences

BLAST step 1

Remove low-complexity areas of query:

Replace repetitive sequences like ”ATATATA” with things like ”NNNNNNN” or ”XXXXXXX”

Code		Complement
A	Adenine	T
C	Cytosine	G
G	Guanine	C
T	Thymine	A
Y	Pyrimidine (C,T)	R
R	Purine (A,G)	Y
W	Weak (A,T)	W
S	Strong (G,C)	S
K	Keto (T,G)	M
M	Amino (A,C)	K
D	A, G, T	H
V	A, C, G	B
H	A, C, T	D
B	C, G, T	V
X/N	any	X/N
-	Gap	-

BLAST step 2

Divide query up into words

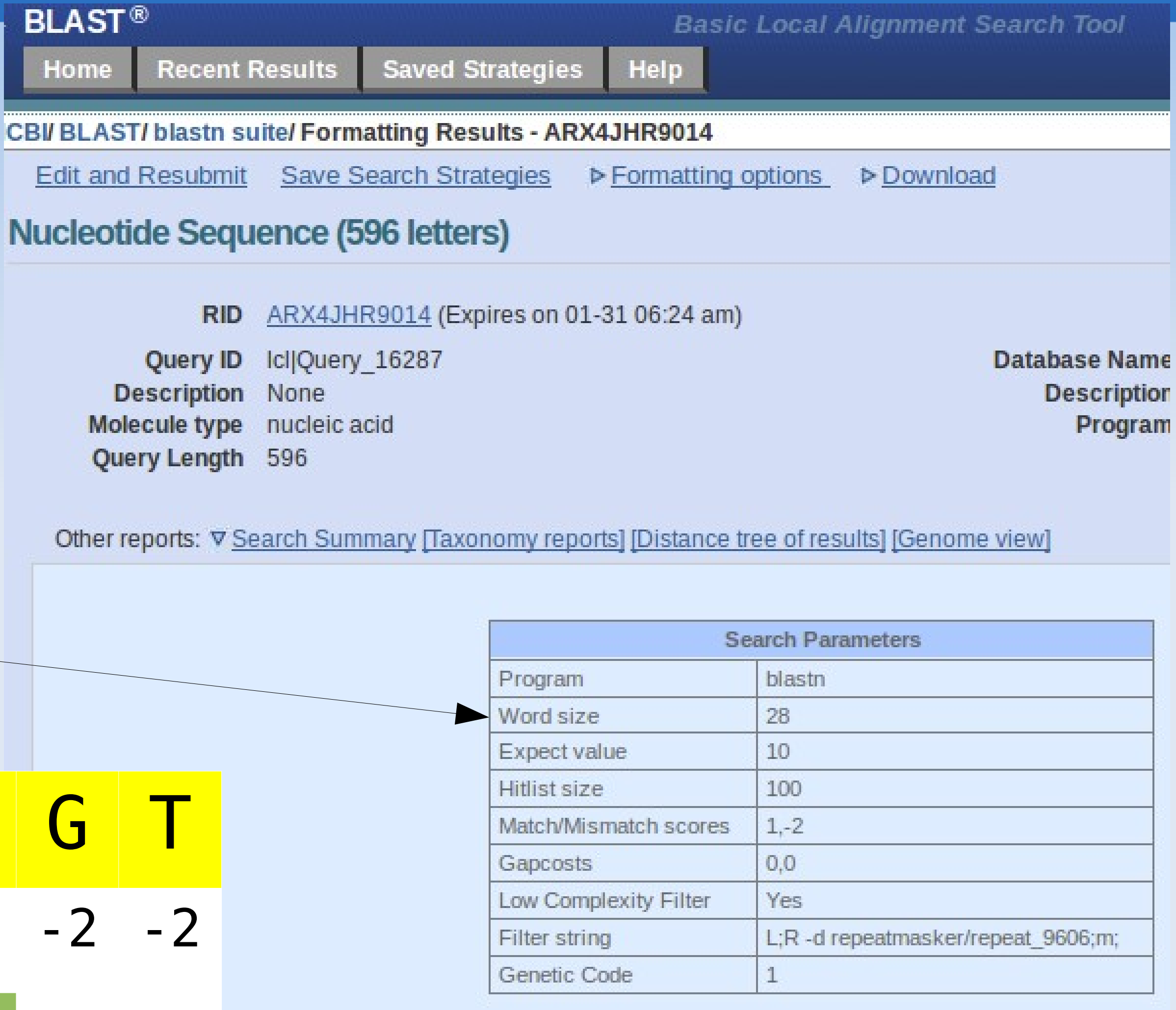
A ”word” is a number of consecutive letters

Default nucleotide word size is 28 for megablast

Note:

scoring matrix is

	A	C	G	T
A	1	-2	-2	-2
C	-2	1	-2	-2
G	-2	-2	1	-2
T	-2	-2	-2	1



CBI/ BLAST/ blastn suite/ Formatting Results - ARX4JHR9014

Edit and Resubmit Save Search Strategies ▷ Formatting options ▷ Download

Nucleotide Sequence (596 letters)

RID	ARX4JHR9014 (Expires on 01-31 06:24 am)
Query ID	lcl|Query_16287
Description	None
Molecule type	nucleic acid
Query Length	596

Database Name
Description
Program

Other reports: ∇ Search Summary [Taxonomy reports] [Distance tree of results] [Genome view]

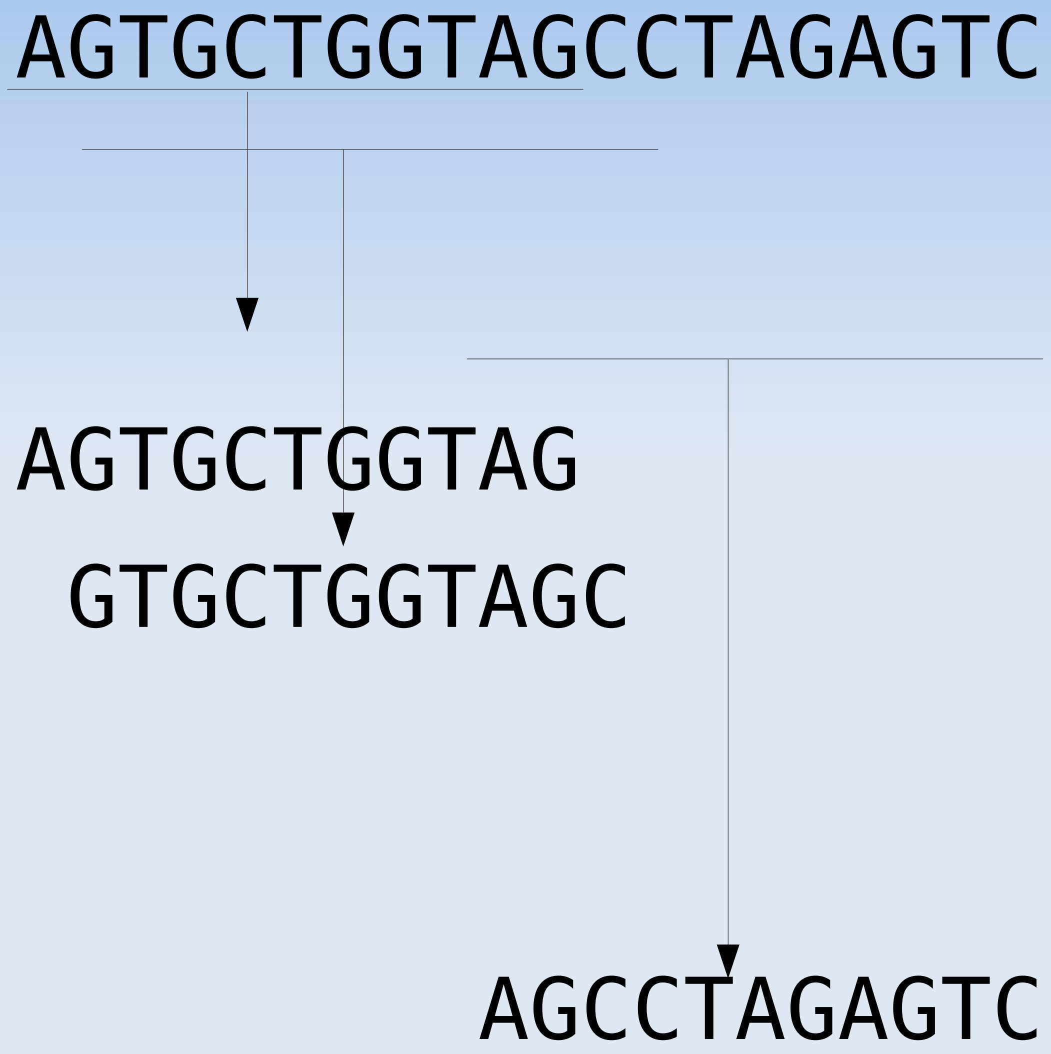
Search Parameters	
Program	blastn
Word size	28
Expect value	10
Hitlist size	100
Match/Mismatch scores	1,-2
Gapcosts	0,0
Low Complexity Filter	Yes
Filter string	L;R -d repeatmasker/repeat_9606;m;
Genetic Code	1

BLAST step 2

Divide query up into words

Default nucleotide word size can be 11 (blastn) to 28 (megablast)

Protein words are smaller

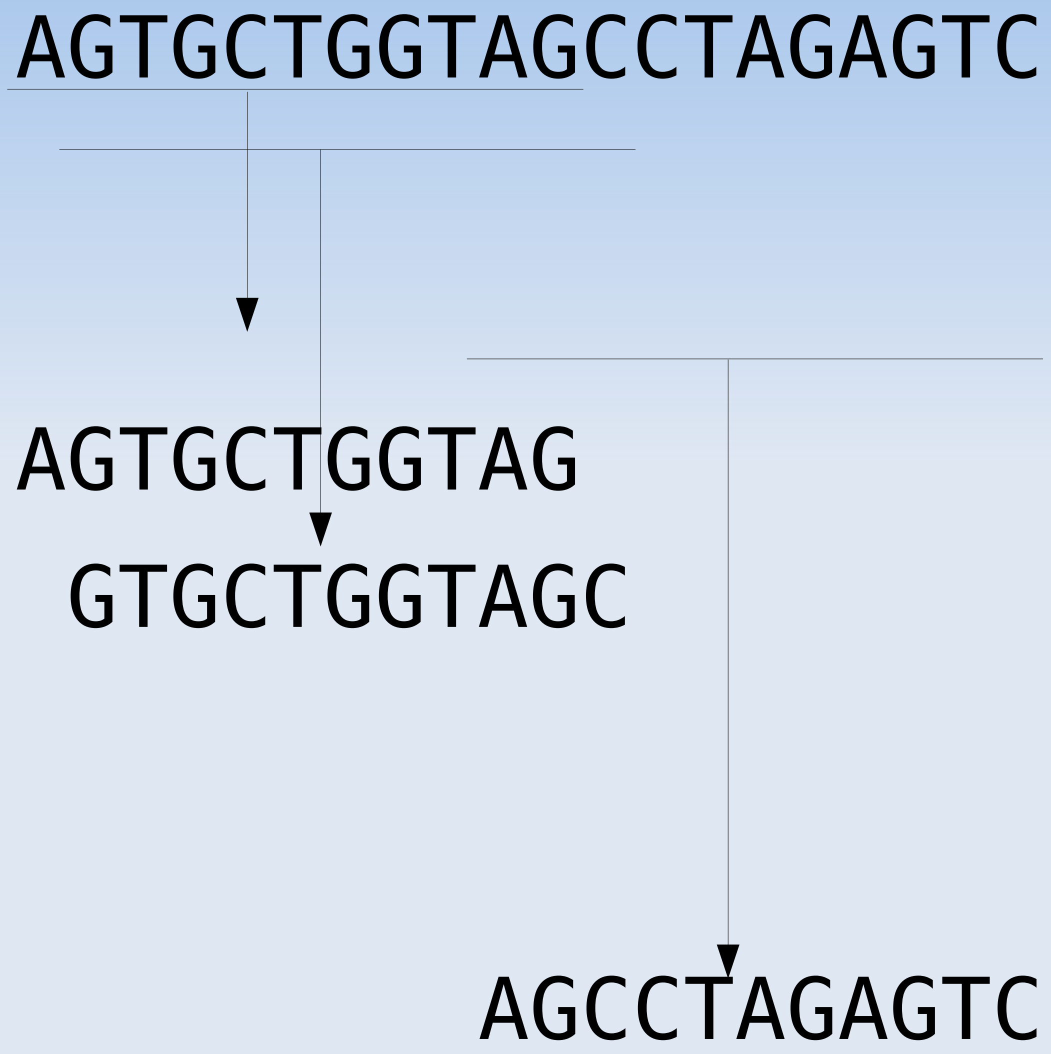


BLAST step 2

There are 4^{11} (> 4 million) possibile 11-nucleotide words

How many for a 28 nucleotide word?

Which is likely to have more exact matches in sequences in a database? A 11nt word or a 28nt word?



BLAST step 2

Find all common words between the query and each database sequence

Evaluate (use scoring matrix) word matches and keep those that exceed a threshold

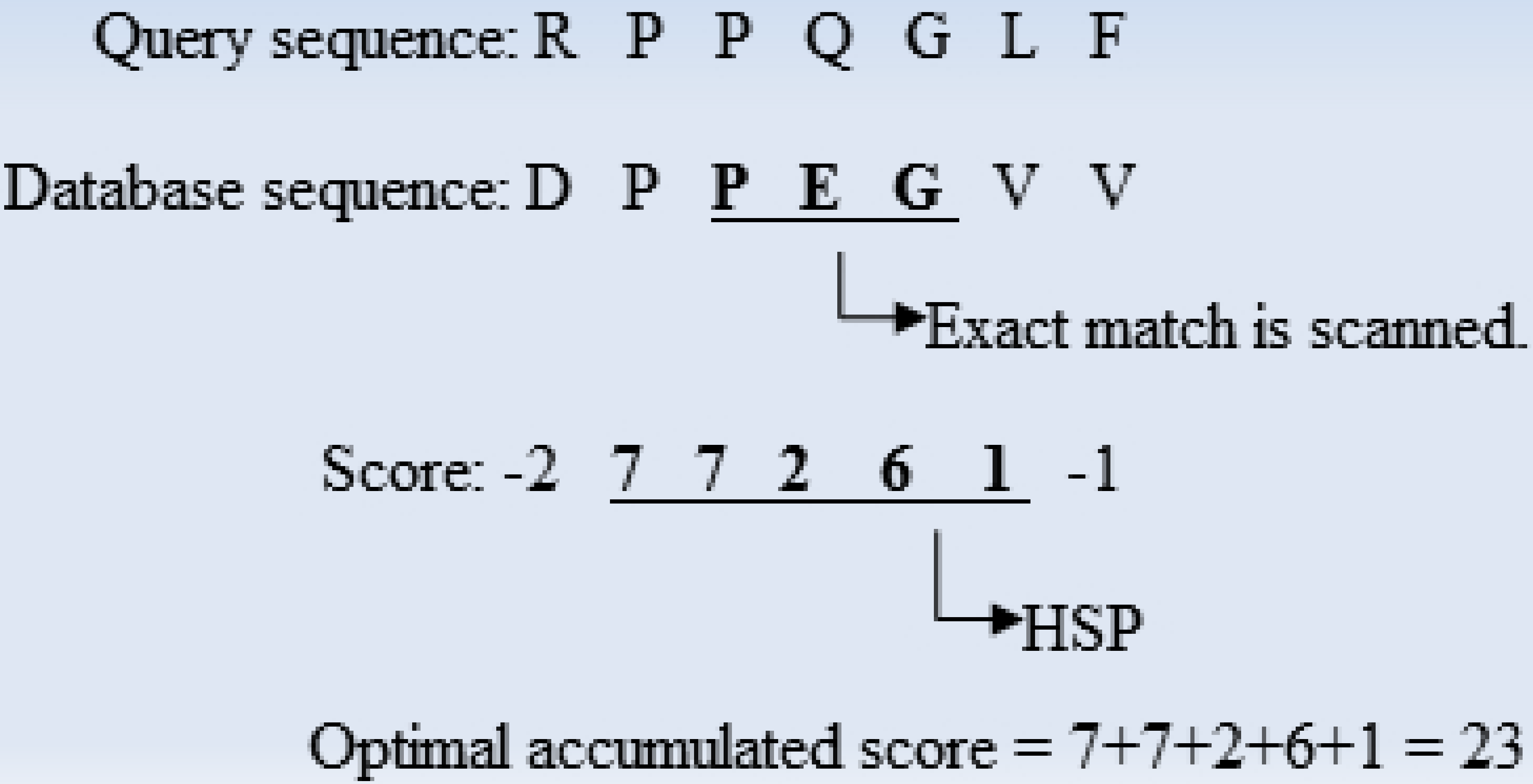
```
Query:     ACGAGATCAGGCACAGGA
Database:  ACTAGATCAGTCACAGCA
```

For each good word match, the alignment is extended until the score drops (below 20 for nucleotides).

Remind you of somehing? Dot-plots with window and stringency?

BLAST step 2

Protein example of extending matches to High-scoring Segment Pairs



```
Query sequence:    R  P  P  Q  G  L  F
Database sequence: D  P  P  E  G  V  V
                         └─►Exact match is scanned.
Score:            -2  7  7  2  6  1  -1
                            └─►HSP
```

Optimal accumulated score = 7+7+2+6+1 = 23

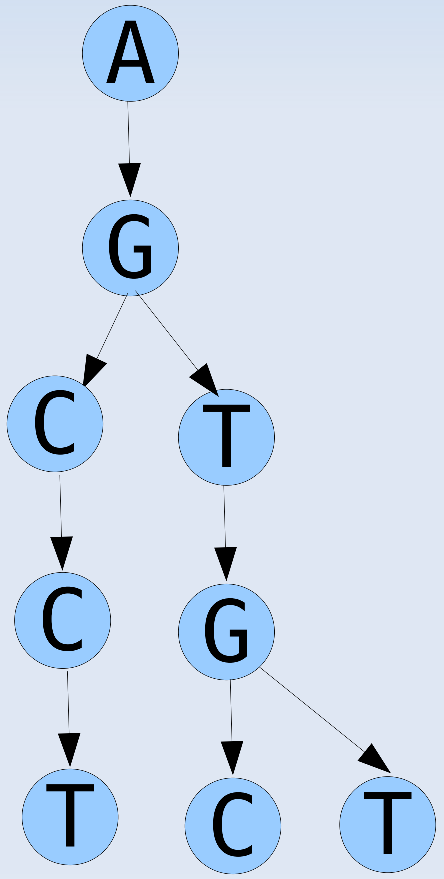
BLAST step 2

Use an efficient search tree to list the high-scoring words that appear in a potential database sequence:

AGCCT

AGTGC

AGTGT



BLAST step 3: Expect (E) values

E value meaning: measures randomness of relationship between sequences.

e.g. E-value of 0.01 for an alignment means there is 1% chance that the two sequences are unrelated.

< 1e-100 → identical sequences

1e-50 to 1e-100 → almost identical sequences

1e-10 to 1e-50 → closely related

1 to 1e-10 → possible but unlikely to be related

10 to 1 → unlikely to be related

Default threshold for E value is 10 because >10 means alignments are no good

Karlin-Altschul eqn

Expectation value $E = kmNe^{-\lambda S}$

measures randomness of relationship between 2 sequences; depends on sequence composition, length, scoring matrices (coming up)

E ~ number of HSPs purely by chance

- k is a constant
- m is the number of letters in query
- N is the total number of letters in target database
- %lambda is a normalizing constant
- S is the score of the high-scoring segment pair

Scoring Matrices

$$\begin{array}{c|cccc|} & 1 & 2 & \cdots & n \\ \hline 1 & a_{11} & a_{12} & \cdots & a_{1n} \\ 2 & a_{21} & a_{22} & \cdots & a_{2n} \\ 3 & a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m & a_{m1} & a_{m2} & \cdots & a_{mn} \\ \hline \end{array}$$

An m × n matrix: the m rows are horizontal,
 the n columns are vertical.
Each element of a matrix is denoted by a variable with two subscripts.
e.g. $a_{2,1}$ is the element in the second row
 and first column.

Nucleotide Scoring Matrices

- A way to rank ”similarity” or ”homology”

  e.g. nucleotide match/no match, w/o penalty

	A	T	C	G
A	1	0	0	0
T	0	1	0	0
C	0	0	1	0
G	0	0	0	1

Scoring Matrices

- Example: match/no match, different penalties for purines, pyramidines transitions/transversions
- Transition = purine (A↔G) pyramidine (C↔T)

	A	T	C	G
A	5	-4	-4	-2
T	-4	5	-2	-4
C	-4	-2	5	-4
G	-2	-4	-4	5

Proteins: PAM Matrices

"Point Accepted Mutation" or PAM scoring matrix.

- Margaret Dayhoff's group in the 1970s looked at mutations seen in proteins by natural selection:
  - 71 groups of related proteins
  - each group had 85% sequence identity
  - 1572 changes

Wanted to explain small changes in sequences

- PAM-1 ~ 1% divergence ~ 1 amino acid change per 100 residues = a PAM unit of time
- PAM-K predicts changes after K PAM time units

PAM Matrices

Matrix values, $M_{i,j}$ represents the probability of a J→I substitution.

$$M_{ij} = \frac{m_j F_{ij}}{\sum_i F_{ij}} \qquad M_{G,A} = \frac{2.8 \times 3}{4}$$

The entries of the scoring matrix are the $M_{i,j}$ values divided by the frequency of occurrence - f_i - of residue *i*.

- e.g. f_G = 10 GLY / 63 residues = 0.1587
- $R_{G,A} = \log(2.1/0.1587) = \log(12.760) = 1.106$
- *Log-odds* matrix
- Diagonal entries are $M_{jj} = 1 - m_j$

Log odds matrix values:
>0 if substitution is frequently seen
<0 if infrequent

Compute PAM-K Matrices

Assume a Markov chain process:

changes at time T+1 are independent of the changes at time T

$$P(A \rightarrow B) = \sum_X P(A \rightarrow X)\ P(X \rightarrow B)$$

PAM-K = [PAM-K-1] [PAM-1]   Matrix multiplication

Small K for closely related sequences

Large K for more distant sequences but Matrix multiplication magnifies errors

PAM-250 is common

Proteins: BLOSUM Matrices

- Used a set of related protein sequences to obtain “blocks”.

  ~2000 blocks from 500 families of related proteins → *more data than PAM*

- A *block* is the ungapped alignment of a highly conserved region of a family of proteins ~ functional protein “motif”.

- BLOck SUBstitutions seen in blocks → BLOSUM matrices

Computing BLOSUM-K

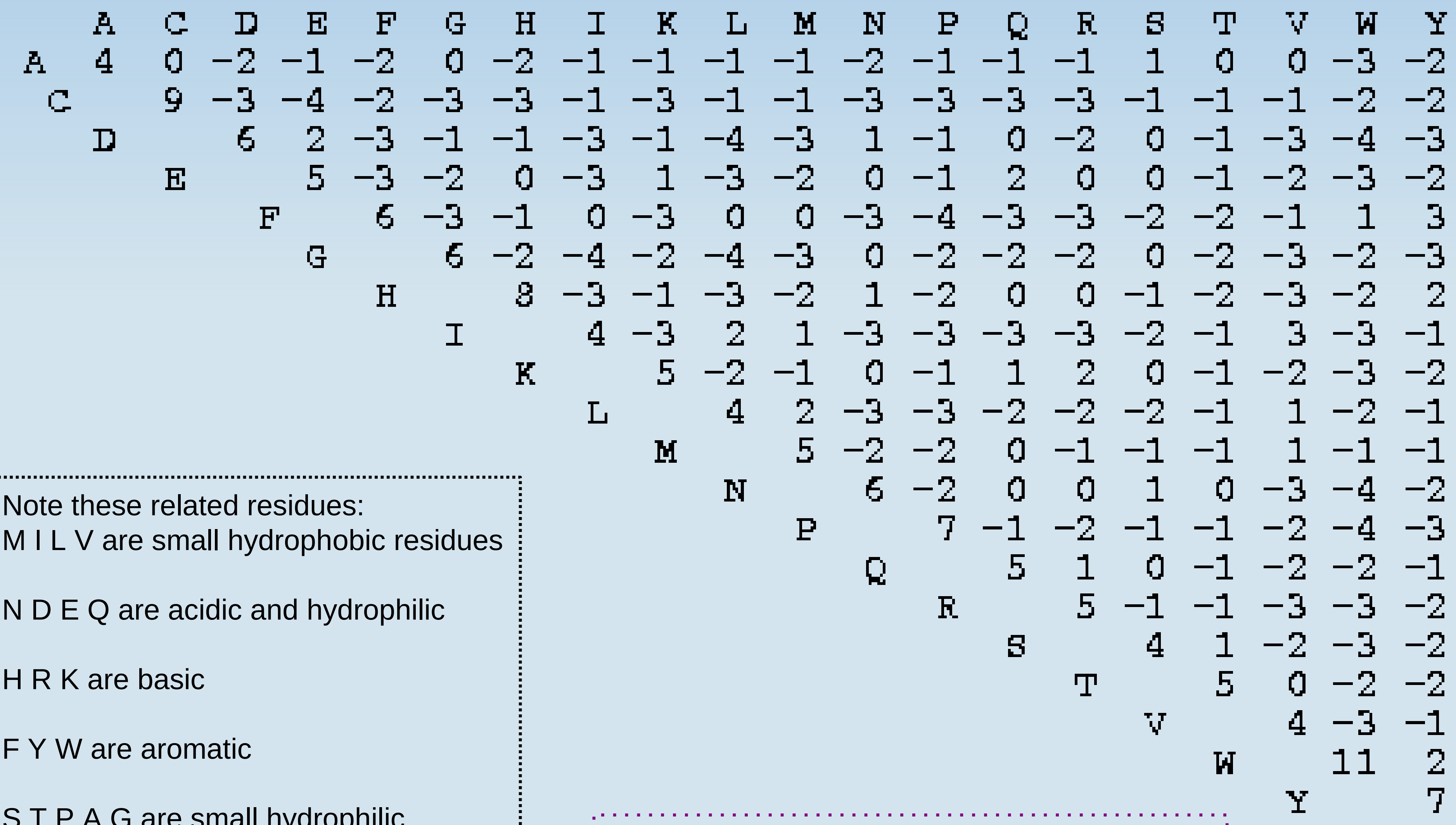
- BLOSUM-K matrix created by weighting the degree of similarity between sequences.

  e.g. BLOSUM-62 is calculated from protein blocks: if two sequences are more than **62% identical**, contribution of these sequences is weighted to sum to one.

- Contributions of multiple entries of closely related sequences is reduced.

- Larger numbers used to measure more recent divergence, default is BLOSUM-62

BLOSUM-62



	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-2	-1	-1	-1	1	0	0	-3	-2
C		9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D			6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	-1	-3	-4	-3
E				5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	-1	-2	-3	-2
F					6	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G						6	-2	-4	-2	-4	-3	0	-2	-2	-2	0	-2	-3	-2	-3
H							8	-3	-1	-3	-2	1	-2	0	0	-1	-2	-3	-2	2
I								4	-3	2	1	-3	-3	-3	-3	-2	-1	3	-3	-1
K									5	-2	-1	0	-1	1	2	0	-1	-2	-3	-2
L										4	2	-3	-3	-2	-2	-2	-1	1	-2	-1
M											5	-2	-2	0	-1	-1	-1	1	-1	-1
N												6	-2	0	0	1	0	-3	-4	-2
P													7	-1	-2	-1	-1	-2	-4	-3
Q														5	1	0	-1	-2	-2	-1
R															5	-1	-1	-3	-3	-2
S																4	1	-2	-3	-2
T																	5	0	-2	-2
V																		4	-3	-1
W																			11	2
Y																				7

Note these related residues:
M I L V are small hydrophobic residues

N D E Q are acidic and hydrophilic

H R K are basic

F Y W are aromatic

S T P A G are small hydrophilic

C is a sulphydryl
Lecture 11 has info on amino acids

Matrix values:
>0 if substitution is frequently seen
<0 if infrequent

PAM *vs.* BLOSUM

- Approximate equivalences:
  Note: lower BLOSUM means more divergence

PAM-100	~	BLOSUM-90
PAM-120	~	BLOSUM-80
PAM-160	~	BLOSUM-60
PAM-200	~	BLOSUM-52
PAM-250	~	BLOSUM-45

- BLOSUM-62 is a good default to use
  Both are available in BLAST

Local *vs.* Global Alignments

Local	Global
Look for matching local regions	Align entire sequences
Good for divergent sequences with some local similarities	Good for sequences related by homology
Sequence lengths can be very different	Sequence lengths are similar
Example: Smith-Waterman	Example: Needleman-Wunsch