Sequence analysis and Genomics

October 12th – November 23rd
2 PM – 5 PM

Prof. Peter Stadler
Dr. Katja Nowick

Katja: group leader TFome and Transcriptome Evolution
Bioinformatics group
Paul-Flechsig-Institute for Brain Research
www. nowicklab.info
nowick@bioinf.uni-leipzig.de

Sequence analysis and genomics

1. Alignments

Dr. Katja Nowick

Group leader TFome and Transcriptome Evolution
Bioinformatics group
Paul-Flechsig-Institute for Brain Research
www. nowicklab.info
nowick@bioinf.uni-leipzig.de

What is an alignment?

- Arranging sequences in a way to identify regions of similarity
- DNA, RNA, protein sequences
- Gaps are inserted, so that identical characters are in the same column
- The more common letters in the sequences the higher their similarity
- The number of non-matches is the *edit sequence*
- Goal: find the longest common subsequence or minimize number of necessary edits
- In example: 4 matches, edit distance = 3 (delete J, T→R, add N)

```
JAEHTE
AEHREN
```

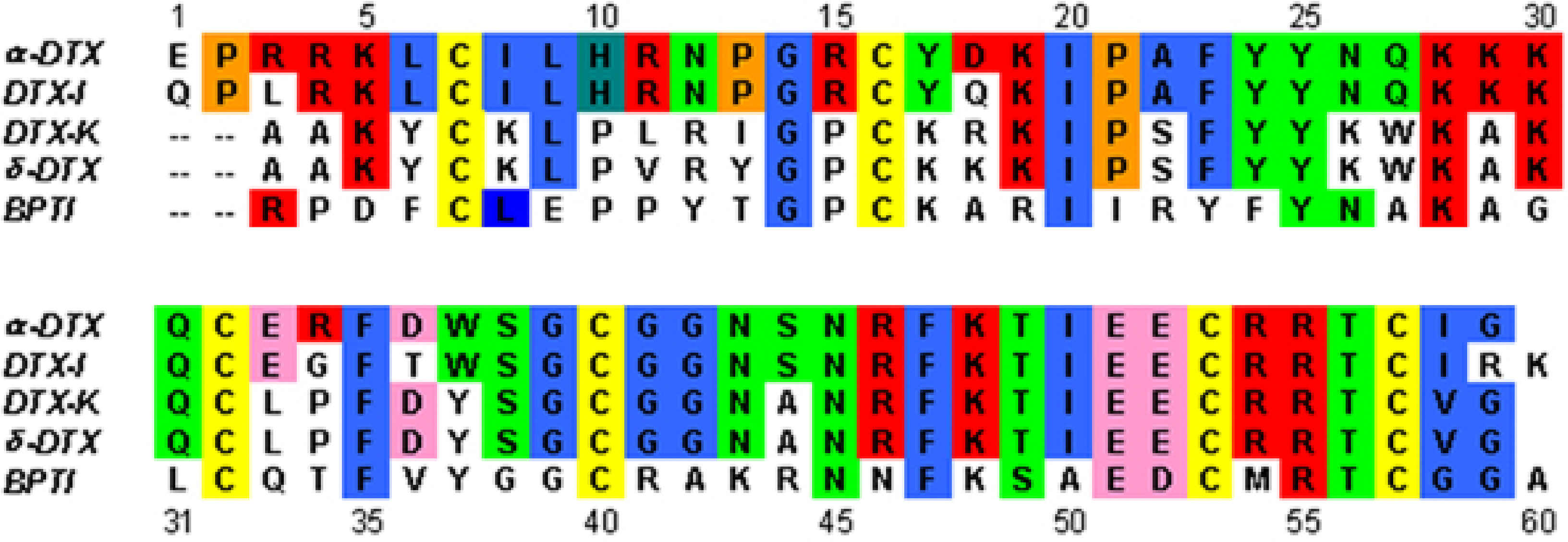
```
JAEHTE-
 |||  |
-AEHREN
```

Why making alignments?

- To see if two sequence are similar

  → might have the same ancestor (evolution)

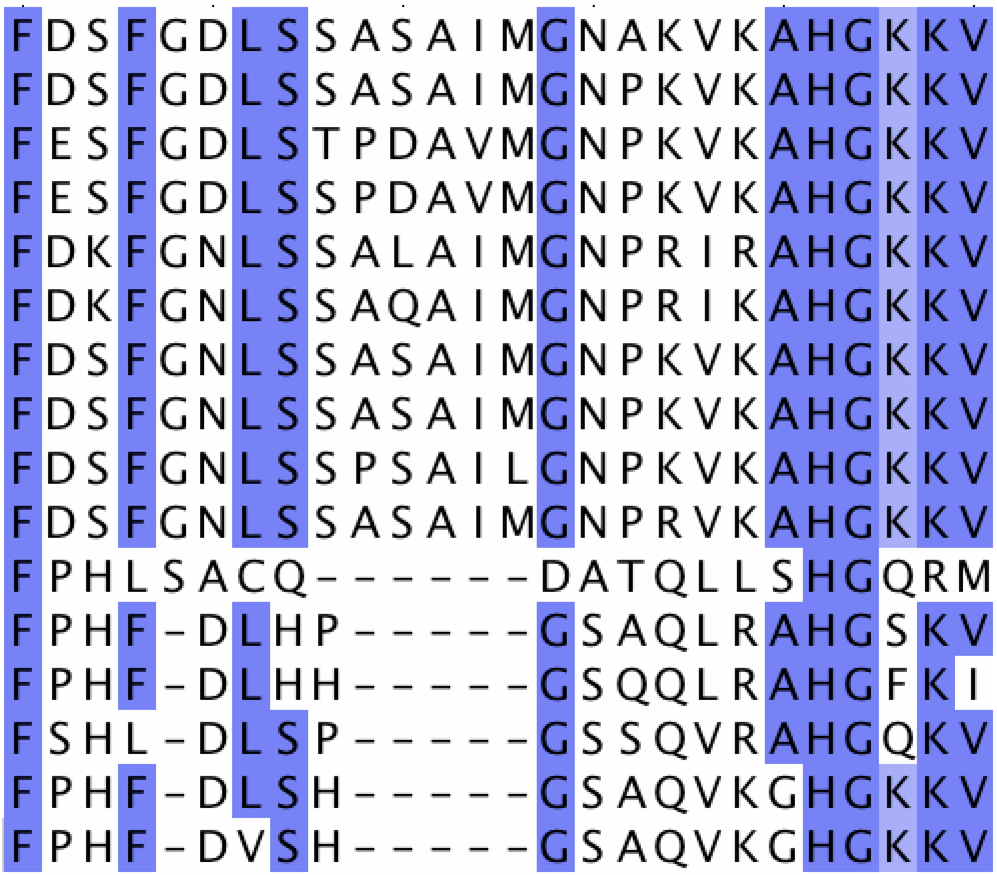
  → might have a similar structure or functional



	1				5					10					15					20					25					30
α-DTX	E	P	R	R	K	L	C	I	L	H	R	N	P	G	R	C	Y	D	K	I	P	A	F	Y	Y	N	Q	K	K	K
DTX-I	Q	P	L	R	K	L	C	I	L	H	R	N	P	G	R	C	Y	Q	K	I	P	A	F	Y	Y	N	Q	K	K	K
DTX-K	--	--	A	A	K	Y	C	K	L	P	L	R	I	G	P	C	K	R	K	I	P	S	F	Y	Y	K	W	K	A	K
δ-DTX	--	--	A	A	K	Y	C	K	L	P	V	R	Y	G	P	C	K	K	K	I	P	S	F	Y	Y	K	W	K	A	K
BPTI	--	--	R	P	D	F	C	L	E	P	P	Y	T	G	P	C	K	A	R	I	I	R	Y	F	Y	N	A	K	A	G

α-DTX	Q	C	E	R	F	D	W	S	G	C	G	G	N	S	N	R	F	K	T	I	E	E	C	R	R	T	C	I	G	
DTX-I	Q	C	E	G	F	T	W	S	G	C	G	G	N	S	N	R	F	K	T	I	E	E	C	R	R	T	C	I	R	K
DTX-K	Q	C	L	P	F	D	Y	S	G	C	G	G	N	A	N	R	F	K	T	I	E	E	C	R	R	T	C	V	G	
δ-DTX	Q	C	L	P	F	D	Y	S	G	C	G	G	N	A	N	R	F	K	T	I	E	E	C	R	R	T	C	V	G	
BPTI	L	C	Q	T	F	V	Y	G	G	C	R	A	K	R	N	N	F	K	S	A	E	D	C	M	R	T	C	G	G	A
	31				35					40					45					50					55					60

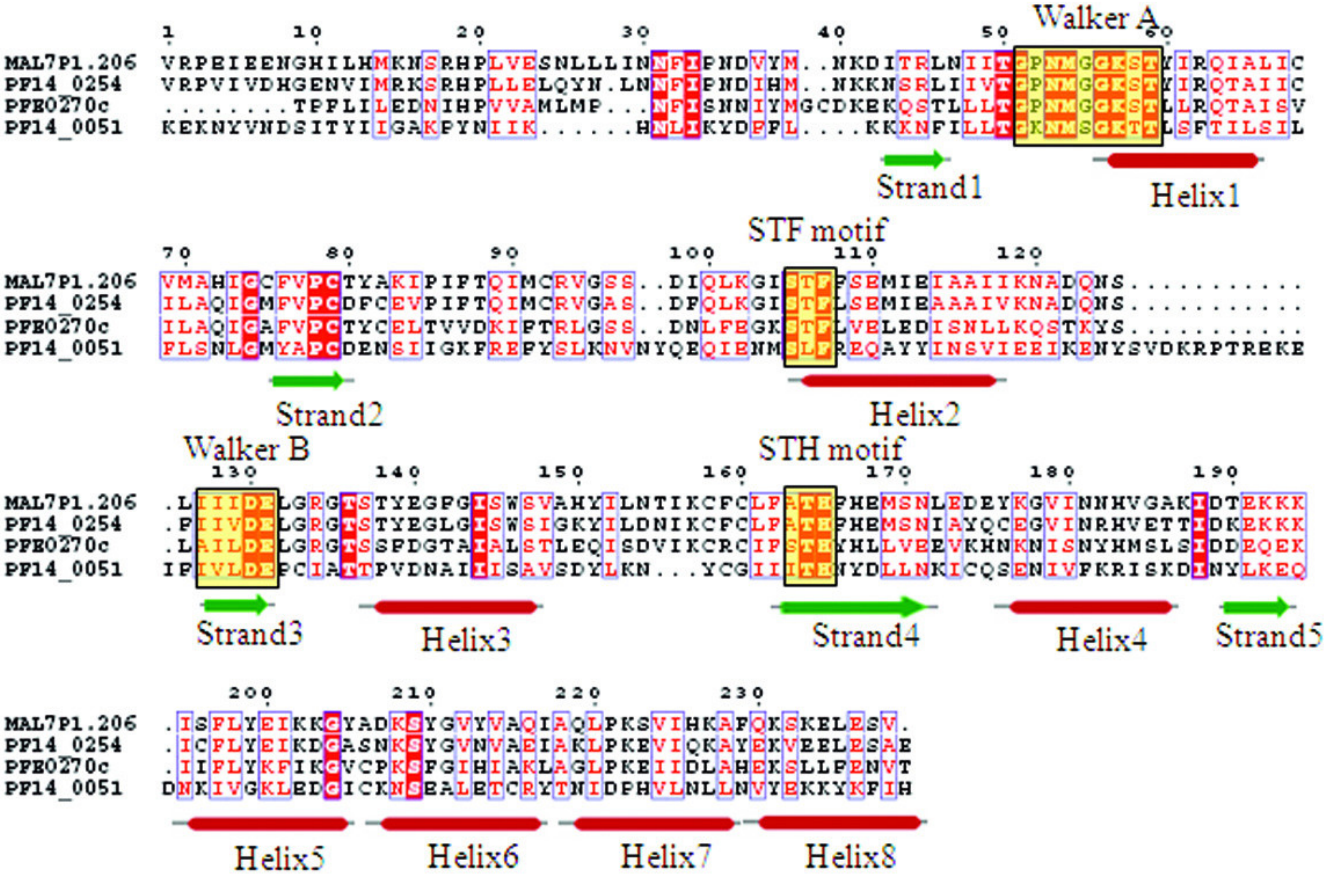
Reconstructing evolutionary history



```
FDSFGDLSSASAIMGNAKVKAHGKKV
FDSFGDLSSASAIMGNPKVKAHGKKV
FESFGDLSTPDAVMGNPKVKAHGKKV
FESFGDLSSPDAVMGNPKVKAHGKKV
FDKFGNLSSALAIMGNPRIRAHGKKV
FDKFGNLSSAQAIMGNPRIKAHGKKV
FDSFGNLSSASAIMGNPKVKAHGKKV
FDSFGNLSSASAIMGNPKVKAHGKKV
FDSFGNLSSPSAILGNPKVKAHGKKV
FDSFGNLSSASAIMGNPRVKAHGKKV
FPHLSACQ------DATQLLSHGQRM
FPHF-DLHP-----GSAQLRAHGSKV
FPHF-DLHH-----GSQQLRAHGFKI
FSHL-DLSP-----GSSQVRAHGQKV
FPHF-DLSH-----GSAQVKGHGKKV
FPHF-DVSH-----GSAQVKGHGKKV
```

If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as indels (that is, insertion or deletion mutations) introduced in one or both lineages since the time they diverged from one another

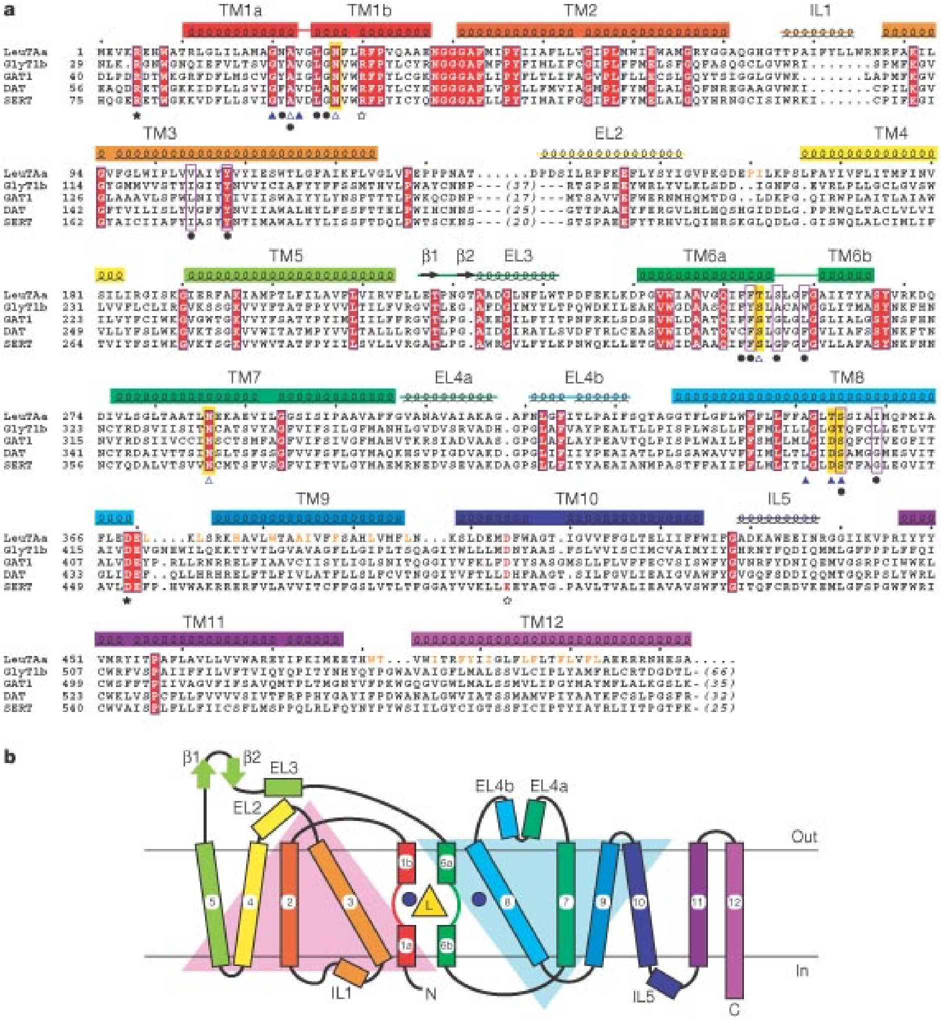
Identification of similar structures



The degree of similarity between amino acids at a particular position in the sequence can be interpreted as a rough measure of how conserved a particular region or sequence motif is

The absence of substitutions, or the presence of only very conservative substitutions suggest that this region has structural or functional importance

Similar sequences can indicate conserved domains



A simple example

```
ATCTCGAG
ATCTCCGAG
AGCTCGAG
ATCCGAG
```

```
ATCT-CGAG
ATCTCCGAG
AGCT-CGAG
ATC--CGAG
```

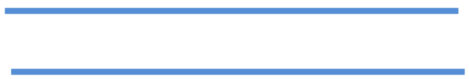
Editing: Introducing of substitutions, insertion, deletions

Global vs. Local Alignments

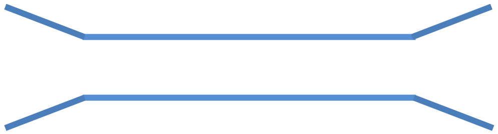
- treat the two sequences as potentially equivalent
- attempt to align every residue in every sequence
- most useful when the sequences in the query set are similar and of roughly equal size
- Can still have gaps
- e.g. for homologous genes

- Sequences may or not be related
- Attempts to find similar parts (substring) in a sequence
- more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context.
- e.g. looking for conserved domains or finding a sequence in the genome

```
FTFTALILLAVAV
F--TAL-LLA-AV
```



```
FTFTALILL-AVAV
--FTAL-LLAAV--
```



With sufficiently similar sequences, there is no difference between local and global alignments

Semiglobal (“glocal”)

- Hybrid method
- Two sequences of different lengths
- attempts to find the best possible alignment that includes the start and end of one or the other sequence (Is the shorter sequence contained within the longer one?)
- Useful when the downstream part of one sequence overlaps with the upstream part of the other sequence. In this case, neither global nor local alignment is entirely appropriate: a global alignment would attempt to force the alignment to extend beyond the region of overlap, while a local alignment might not fully cover the region of overlap

```
ATATCGACGTC
     GACGTCATTTG
```

Pairwise alignment methods

- To align two sequences

3 main methods:

1. Dot matrix
2. Dynamic programming
3. Word methods

1. Dot matrix

The two sequences are written along the top row and leftmost column of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match

	C	T	G	C	A	T	C
C	■						
T		■					
C							
C				■			
A					■		
G							
C							■

1. Dot matrix

The two sequences are written along the top row and leftmost column of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match

	C	T	G	C	A	T	C
C	■			■			■
T		■				■	
C	■			■			■
C	■			■			■
A					■		
G			■				
C	■			■			■

1. Dot matrix

If aligning a sequence to itself, you get the diagonal
and you can see repetitive regions in your sequence (regions with self-similarity): lines parallel to the diagonal

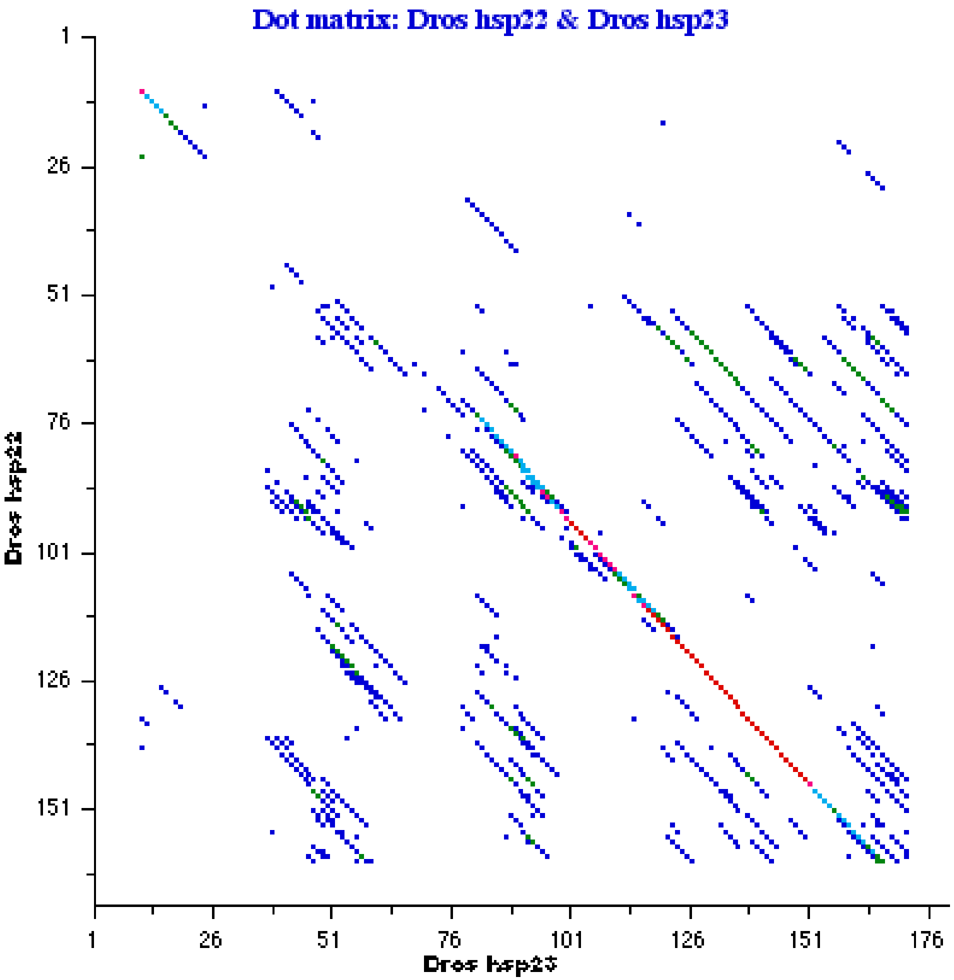
e.g. CTAG

	A	C	T	A	G	C	G	C	T	A	G	T	C	T	A
A	■			■						■					■
C		■				■		■					■		
T			■						■					■	
A	■			■						■					■
G					■		■				■				
C		■				■		■					■		
G					■		■				■				
C		■				■		■					■		
T			■						■			■		■	
A	■			■						■					■
G					■		■				■				
T			■						■			■		■	
C		■				■		■					■		
T			■						■			■		■	
A	■			■						■					■

1. Dot matrix

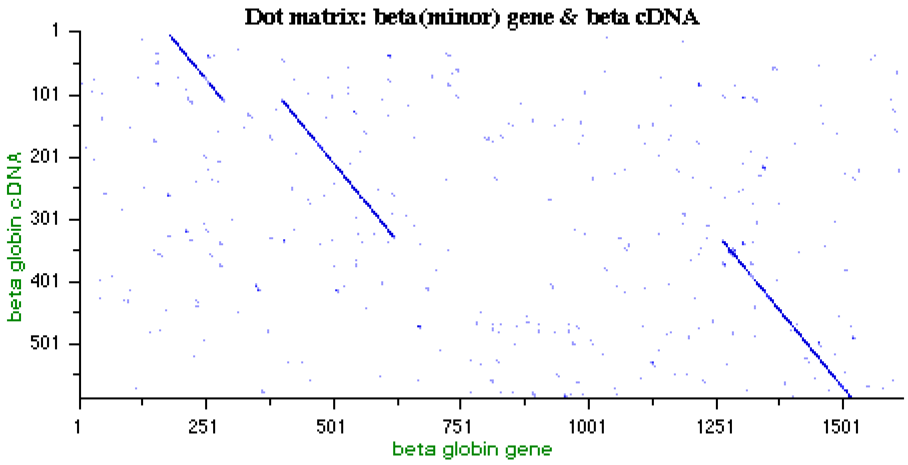
If aligning two similar sequence with repetitive regions in your sequence (regions with self-similarity): lines off the diagonal

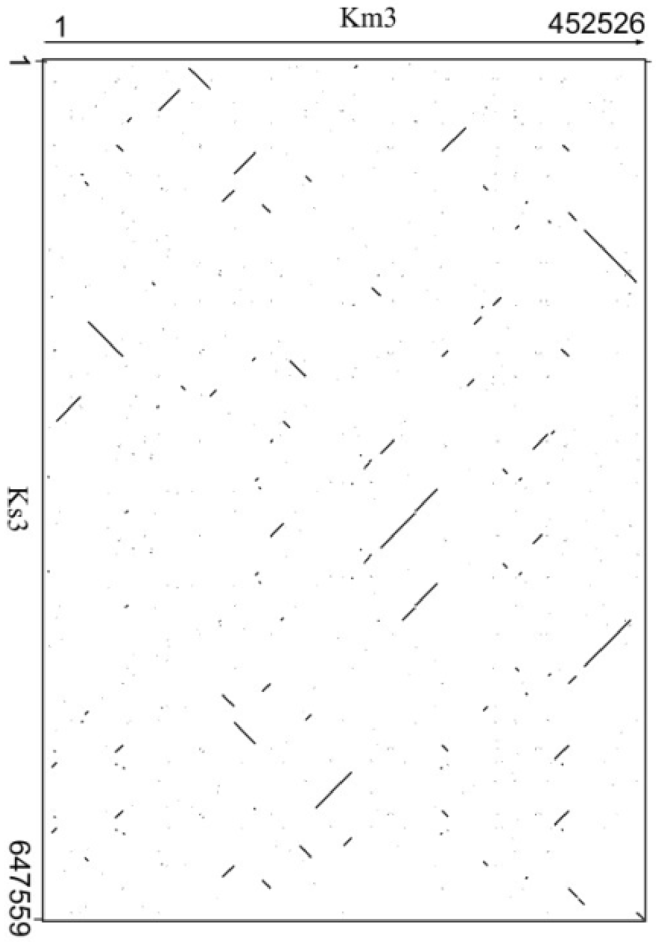
e.g. Hsp 22 vs. Hsp 23



1. Dot matrix

Other features to visually identify with dot matrices:
e.g. insertions/deletions, inverted repeats





1. Dot matrix

Problems with dot matrices:

- Noise
- Lack of clarity
- Non-intuitiveness
- Difficult to extract match summary statistics and matched positions in the two sequences

  Wasted space: most of the actual area of the plot is taken up by either empty space or noise where the match data is inherently duplicated across the diagonal
- Limited to two sequences

2. Dynamic Programming (DP)

- A method to solve a complex problem by splitting it into simpler sub-problems (recursion):

- Solve the smallest sub-problem first
- Then go back to solve increasingly bigger sub-problems until the problem is solved
- Result of the bigger sub-problems depend on the previous results of the smaller sub-problems

- Only works for problems that can be split into smaller sub-problems and when sub-problems can be nested recursively into larger problems
- Only works if there is a relation between the result of the larger and the smaller problems

- Each sub-problem is only solved once and then the solution is stored (memorized)
- Next time the same sub-problem occurs, the solution is looked up instead of computed again
- This makes the computation faster

- e.g. Alignments: align increasingly larger sub-sequences until the whole sequences are aligned

2. Dynamic Programming for alignments

- Calculate the whole alignment from alignments of increasing substrings

- Need a scoring function to evaluate the quality of the alignment:
- Idea: perfect matches get a high score
  similar letters get a moderate score
  gaps get a low score

- Make all possible alignments and pick the one with the highest score

- The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function
- However, identifying a good scoring function is often an empirical rather than a theoretical matter

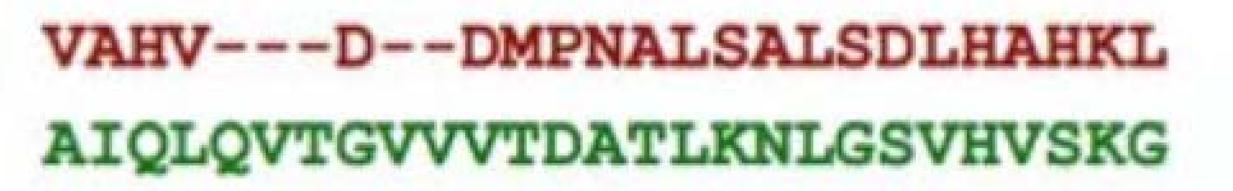
- e.g. Needleman-Wunsch algorithm for global alignments
  Smith-Waterman algorithm for local alignments

Scoring alignments

- Substitution matrix to assign scores to matches or mismatches:
  for amino acids, different substitutions are usually rated differently (see next slide)
  for DNA and RNA, in practice often simply assign a positive match and a negative mismatch score

- Gap penalty for matching an amino acid/nucleotide in one sequence to a gap in the other

- The score of an alignment is the sum of the scores for each position plus gap costs



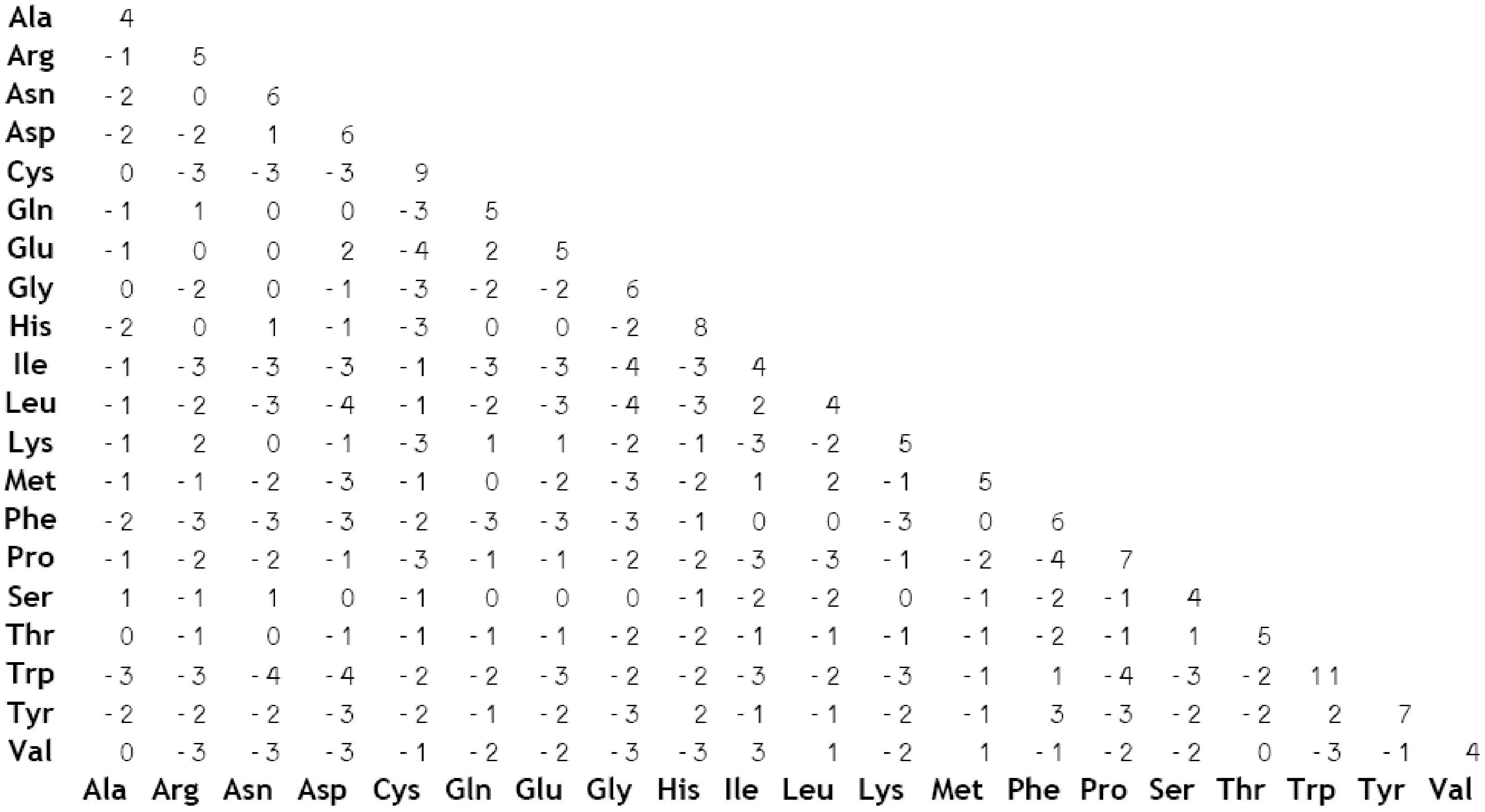
```
VAHV---D--DMPNALSALSDLHAHKL
AIQLQVTGVVVTDATLKNLGSVHVSKG
```

S(V,A) + s(A,I) + s(H,Q) + s (V,L) + 3 gaps +....

Scoring protein alignments

Are all amino acid changes equally likely?

Scoring matrices for protein alignments: PAM (Point accepted mutations)
BLOSUM (Block substitution matrix)



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

Scoring matrix

Problem components:

- An alphabet

e.g., *{ A, C, G, T }* (DNA

e.g., *{ M, A, L, ..., }* (protein)

- Two input strings

- Scoring matrix (costs for editing):
  insert value for match, mismatch, or gap
  e.g. match = 1
  mismatch = -1
  gap = -2

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

Scoring matrix

Problem components:

- An alphabet

e.g., *{ A, C, G, T }* (DNA
e.g., *{ M, A, L, ..., }* (protein)

- Two input strings

- Scoring matrix (costs for editing):

  insert value for match, mismatch, or gap
  e.g. match = 1
  mismatch = -1
  gap = -2

	A	**C**	**T**	**G**	**gap**
A	1	-1	-1	-1	-2
C	-1	1	-1	-1	-2
T	-1	-1	1	-1	-2
G	-1	-1	-1	1	-2
gap	-2	-2	-2	-2	1

String 1: A C A
String 2: A C G

Costs: 1 1 -1 => alignment score = 1

Gap penalties

- Contribute to the overall score of alignments
- The size of the gap penalty affects the alignment that is finally selected
- Selecting a higher gap penalty will cause less favorable characters to be aligned, to avoid creating as many gaps

Match: 1	ACTGA--	ACTGA ✓
Mismatch: -1	AC--ACT	ACACT
Gap: -2	$11_{-2-2}1_{-2-2}=-5$	$11_{-1-1-1}=-1$
Match: 1	ACTGA– ✓	ACTGA
Mismatch: -2	AC--ACT	ACACT
Gap: -1	$11_{-1-1}1_{-1-1}=-1$	$11_{-2-2-2}=-4$

Needleman–Wunsch algorithm

- global alignment on two sequences
- Saul B. Needleman and Christian D. Wunsch 1970
- the first application of dynamic programming to biological sequence comparison

Needleman–Wunsch algorithm

Example:

String 1: TG
String 2: ATT

Scoring matrix:
match = 1
mismatch = -1
gap = -2

String 2

String 1

	i =	0	1	2	3
		$	**A**	**T**	**T**
0	**$**	0			
1	**T**				
j = 2	**G**				

Rules for filling in the matrix: $a_{0,0} = 0$

Needleman–Wunsch algorithm

Example:

String 1: TG
String 2: ATT

Scoring matrix:
match = 1
mismatch = -1
gap = -2

String 2

String 1

		i = 0	1	2	3
		$	**A**	**T**	**T**
0	**$**	0	-2	-4	-6
1	**T**	-2			
j = 2	**G**	-4			

Rules for filling in the matrix:

$$a_{0,\,0} = 0$$
$$a_{0,\,j} = a_{0,\,j\text{-}1} + \textit{gappenalty}$$
$$a_{i,\,0} = a_{i\text{-}1,\,0} + \textit{gappenalty}$$

Needleman–Wunsch algorithm

Example:

String 1: TG
String 2: ATT

Scoring matrix:
match = 1
mismatch = -1
gap = -2

String 2

String 1

i =		0	1	2	3
		$	**A**	**T**	**T**
0	**$**	0	-2	-4	-6
1	**T**	-2	-1	-1	-3
j = 2	**G**	-4	-3	-2	-2

Rules for filling in the matrix:

$$a_{0,0} = 0$$

$$a_{0,j} = a_{0,j\text{-}1} + \mathit{gappenalty}$$

$$a_{i,0} = a_{i\text{-}1,0} + \mathit{gappenalty}$$

$$a_{i,j} = \max\left(a_{i\text{-}1,j} + \mathit{gappenalty},\ a_{i,j\text{-}1} + \mathit{gappenalty},\ a_{i\text{-}1,j\text{-}1} + \mathit{equal}(i,j)\right)$$

Needleman–Wunsch algorithm

Example:

String 1: TG
String 2: ATT

Scoring matrix:
match = 1
mismatch = -1
gap = -2

String 2

String 1

	i =	0	1	2	3
		$	**A**	**T**	**T**
0	**$**	**0**	-2	-4	-6
1	**T**	-2	**-1**	**-1**	-3
j = 2	**G**	-4	-3	-2	**-2**

The values in each cell represent the similarity of the alignment of the subsequences
The value in the bottom right corner represents the similarity of the two complete sequences: **-2**

Reconstructing the alignment = Backtracking: Where did I come from?

Go backwards always to the field with the highest value
Moving diagonally means match/mismatch
Moving left means insertion in string 1
Moving up means insertion in string 2

String 1: - TG
String 2: ATT
Score: -2 1 -1 = -2

Needleman–Wunsch algorithm

Example:

String 1: JAEHTE
String 2: AEHREN

Scoring matrix:
match = 3
mismatch = 0
gap = -1

	$	A	E	H	R	E	N
$	**0**	-1	-2	-3	-4	-5	-6
J	-1	**0**	-1	-2	-3	-4	-5
A	-2	**2**	1	0	-1	-2	-3
E	-3	1	**5**	4	3	2	1
H	-4	0	4	**8**	7	6	5
T	-5	-1	3	7	**8**	7	6
E	-6	-2	2	6	7	**11**	**10**

```
JAEHTE-
 ||| |
-AEHREN
```

Needleman–Wunsch algorithm

“Costs” for matches, mismatches, and gaps can be different (asymmetric)
e.g. when certain substitutions occur more or less frequent
(see e.g. PAM or BLOSSUM matrix)

	A	**C**	**T**	**G**	**gap**
A	1	-2	-1	-1	-2
C	-2	2	-1	-1	-2
T	-1	-1	3	-4	-6
G	-1	-1	-4	1	-2
gap	-2	-2	-6	-2	

String 1: A T T
String 2: A T G T

Costs: 1 3 -4 -6 => alignment score = 1

Smith-Waterman algorithm

- Local alignment of two sequences
- Temple F. Smith and Michael S. Waterman in 1981
- An example of dynamic programming

Smith-Waterman algorithm

Example:

String 1: AGCACACA
String 2: ACACACTA

Scoring matrix:
match = 2
mismatch = -1
gap = -1

String 2

String 1

	i =	0	1	2	3	4	5	6	7	8
		-	A	C	A	C	A	C	T	A
0	-									
1	A									
2	G									
3	C									
4	A									
5	C									
6	A									
7	C									
j = 8	A									

Rules for filling in the matrix:

$H_{i,0} = 0$
$H_{0,j} = 0$

Smith-Waterman algorithm

Example:

String 1: AGCACACA
String 2: ACACACTA

Scoring matrix:
match = 2
mismatch = -1
gap = -1

String 2

String 1

i =		0	1	2	3	4	5	6	7	8
		-	**A**	**C**	**A**	**C**	**A**	**C**	**T**	**A**
0	**-**	0	0	0	0	0	0	0	0	0
1	**A**	0								
2	**G**	0								
3	**C**	0								
4	**A**	0								
5	**C**	0								
6	**A**	0								
7	**C**	0								
j = 8	**A**	0								

Rules for filling in the matrix:

$H_{i,0} = 0$
$H_{0,j} = 0$

Smith-Waterman algorithm

<u>Example:</u>

String 1: AGCACACA
String 2: ACACACTA

Scoring matrix:
match = 2
mismatch = -1
gap = -1

String 2

String 1

	i =	0	1	2	3	4	5	6	7	8
		-	**A**	**C**	**A**	**C**	**A**	**C**	**T**	**A**
0	**-**	0	0	0	0	0	0	0	0	0
1	**A**	0	2	1	2					
2	**G**	0	1	1	1					
3	**C**	0	0	3	2					
4	**A**									
5	**C**									
6	**A**									
7	**C**									
j = 8	**A**									

Rules for filling in the matrix:

$H_{i,0} = 0$

$H_{0,j} = 0$

$H_{i,j} = \max(H_{i-1,j} + \textit{gappenalty},\ H_{i,j-1} + \textit{gappenalty},\ H_{i-1,j-1} + \textit{equal}(i, j))$

Smith-Waterman algorithm

Example:

String 1: AGCACACA
String 2: ACACACTA

Scoring matrix:
match = 2
mismatch = -1
gap = -1

String 2

String 1

	i =	0	1	2	3	4	5	6	7	8
		-	**A**	**C**	**A**	**C**	**A**	**C**	**T**	**A**
0	**-**	0	0	0	0	0	0	0	0	0
1	**A**	0	2	1	2	1	2	1	0	2
2	**G**	0	1	1	1	1	1	1	0	1
3	**C**	0	0	3	2	3	2	3	2	1
4	**A**	0	2	2	5	4	5	4	3	4
5	**C**	0	1	4	4	7	6	7	6	5
6	**A**	0	2	3	6	6	9	8	7	8
7	**C**	0	1	4	5	8	8	11	10	9
j = 8	**A**	0	2	3	6	7	10	10	10	12

Rules for filling in the matrix:

$H_{i,0} = 0$

$H_{0,j} = 0$

$H_{i,j} = \max(H_{i-1,j} + \mathit{gappenalty},\ H_{i,j-1} + \mathit{gappenalty},\ H_{i-1,j-1} + \mathit{equal}(i, j))$

Smith-Waterman algorithm

Example:

String 1: AGCACACA
String 2: ACACACTA

Scoring matrix:
match = 2
mismatch = -1
gap = -1

String 2

String 1

	i =	0	1	2	3	4	5	6	7	8
		-	A	C	A	C	A	C	T	A
0	-	**0**	0	0	0	0	0	0	0	0
1	A	0	**2**	1	2	1	2	1	0	2
2	G	0	**1**	1	1	1	1	1	0	1
3	C	0	0	**3**	2	3	2	3	2	1
4	A	0	2	2	**5**	4	5	4	3	4
5	C	0	1	4	4	**7**	6	7	6	5
6	A	0	2	3	6	6	**9**	8	7	8
7	C	0	1	4	5	8	8	**11**	10	9
j = 8	A	0	2	3	6	7	10	10	**10**	**12**

Reconstructing the alignment:

Start at the position with the highest value: here 12
Go backwards always to the field with the highest value
Moving diagonally means match/mismatch
Moving left means insertion in string 1
Moving up means insertion in string 2

String 1: AGCACAC-A
String 2: A-CACACTA
Score: $2_{-1}22222_{-1}2$ = **12**

Smith-Waterman algorithm

The main differences to the Needleman–Wunsch algorithm:

Backtracking does not need to start at the bottom right corner
Instead starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered → yielding the highest scoring local alignment

If alignments get too expensive, just start a new alignment:
If the value of a cell was negative, set it to zero
Therefore change the recurrence to:
$H_{i,j}$ = max (0, $\mathrm{H}_{i\text{-}1,\, j}$ + *gappenalty*, $\mathrm{H}_{i,\, j\text{-}1}$ + *gappenalty*, $\mathrm{H}_{i\text{-}1,\, j\text{-}1}$ + *equal*(*i*, *j*))

	-	**N**	**N**	**N**	**N**	**N**	**N**	**N**	**N**
-	0	0	0	0	0	0	0	0	0
N	0	0	🟥						
N	0			🟥					
N	0								
N	0			0	0				
N	0			0	🟥				
N	0					🟥	🟥		
N	0							🟥	
N	0								

Gap penalties

Contribute to the overall score of alignments
The size of the gap penalty affects the alignment that is finally selected
Selecting a higher gap penalty will cause less favorable characters to be aligned, to avoid creating as many gaps

- **Constant gap penalty**

The simplest type of gap penalty
The only parameter, d, is added to the alignment score when the gap is first opened. This means that any gap receives the same penalty, regardless of its size

- **Linear gap penalty**

The parameter d is a penalty per unit length of gap
This is almost always negative, so that the alignment with fewer gaps is favored over the alignment with more gaps
The overall penalty for one large gap is the same as for many small gaps

- **Affine gap penalty**

Use a gap opening penalty, o, and a gap extension penalty, e
A gap of length l is then given a penalty o + (l-1)e
o is almost always negative to discourage gaps
E is negative but almost always less negative than o to encourage gap extension, rather than gap introduction.
Makes sense, because a biological sequence is much more likely to have one big gap of length 10, due to a single insertion or deletion event, than it is to have 10 small gaps of length 1.

```
AAAA    AAAA
A-A-    A--A
         more likely
```

3. Word based methods

- Heuristic method
- Not guaranteed to find an optimal alignment solution
- But significantly more efficient than dynamic programming

- Useful in large-scale database searches where it is understood that a large proportion of the candidate sequences will have essentially no significant match with the query sequence

- Implementation in the database search tools like FASTA and BLAST

  … to be continued