CMSC423: Bioinformatic Algorithms, Databases and Tools
Lecture 9

inexact alignment

dynamic programming, gapped alignment

Recap

Global alignment recap

Score[i,j] is the maximum of:

1. Score[i-1, j-1] + Value[S1[i-1],S2[j-1]] (S1[i-1], S2[j-1] aligned)
2. Score[i – 1, j] + Value[S1[i], -] (S1[i] aligned to gap)
3. Score[i, j – 1] + Value[-, S2[j]] (S2[j] aligned to gap)

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

```
AGCGTAG
GTCAGAC
```

Value(A,A) = 10
Value(A,G) = -5
Value(A,-) = -2

Global alignment recap

Score[i,j] is the maximum of:

1. Score[i-1, j-1] + Value[S1[i-1],S2[j-1]] (S1[i-1], S2[j-1] aligned)
2. Score[i – 1, j] + Value[S1[i], -] (S1[i] aligned to gap)
3. Score[i, j – 1] + Value[-, S2[j]] (S2[j] aligned to gap)

	-	A	G	C	G	T	A	G
-	0	-4	-8	-12	-16	-20	-24	-28
G	-4	-5	6	2	-2	-6	-10	-14
T	-8	-9	2	1	-3	8	4	0
C	-12	-13	-2	12	8	4	3	-1
A	-16	-2	-6	8	7	3	14	10
G	-20	-6	8	4	18	14	10	24
A	-24	-10	4	3	14	13	24	20
C	-28	-14	0	14	10	9	20	19

```
AG-C-GTAG
-GTCAG-AC
```

Value(A,A) = 10
Value(A,G) = -5
Value(A,-) = -4

Local alignment recap

Score[i,j] is the maximum of:

0. 0
1. Score[i-1, j-1] + Value[S1[i-1],S2[j-1]] (S1[i-1], S2[j-1] aligned)
2. Score[i – 1, j] + Value[S1[i], -] (S1[i] aligned to gap)
3. Score[i, j – 1] + Value[-, S2[j]] (S2[j] aligned to gap)

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

AGCGTAG
GTCAGAC

Value(A,A) = 10
Value(A,G) = -5
Value(A,-) = -2

Alignment scores

Where do the alignment scores come from?

- PAM matrices
  - PAM1 – based on frequency of mutations between closely related proteins (within 1 "evolutionary step")
  - PAM 2 - ... within 2 evolutionary steps
  - ... PAM 250 – commonly used
- BLOSUM matrices
  - Frequency of mutations between proteins that are x% similar
  - BLOSUM100 – based on proteins that are exactly the same (e.g. score(A,A) is defined but not score(A,G))
  - BLOSUM62 – commonly used
- gap scores usually determined empirically

BLOSUM62

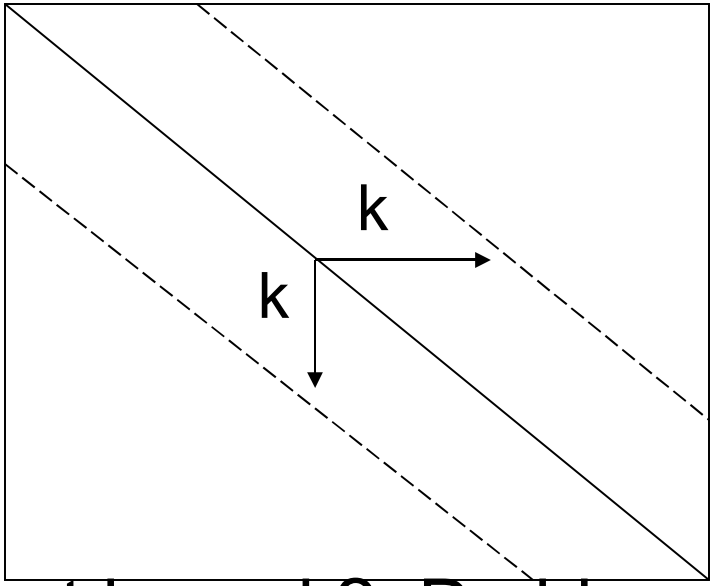
Table 2 - The log odds matrix for BLOSUM 62

	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

Heuristics

Heuristics

- What if limit the # of differences allowed? E.g. we expect the sequences to be very similar.
- Compute 'banded' alignment – stay within # of differences (k) from the diagonal.
- Optimal alignment cannot stray too far from diagonal



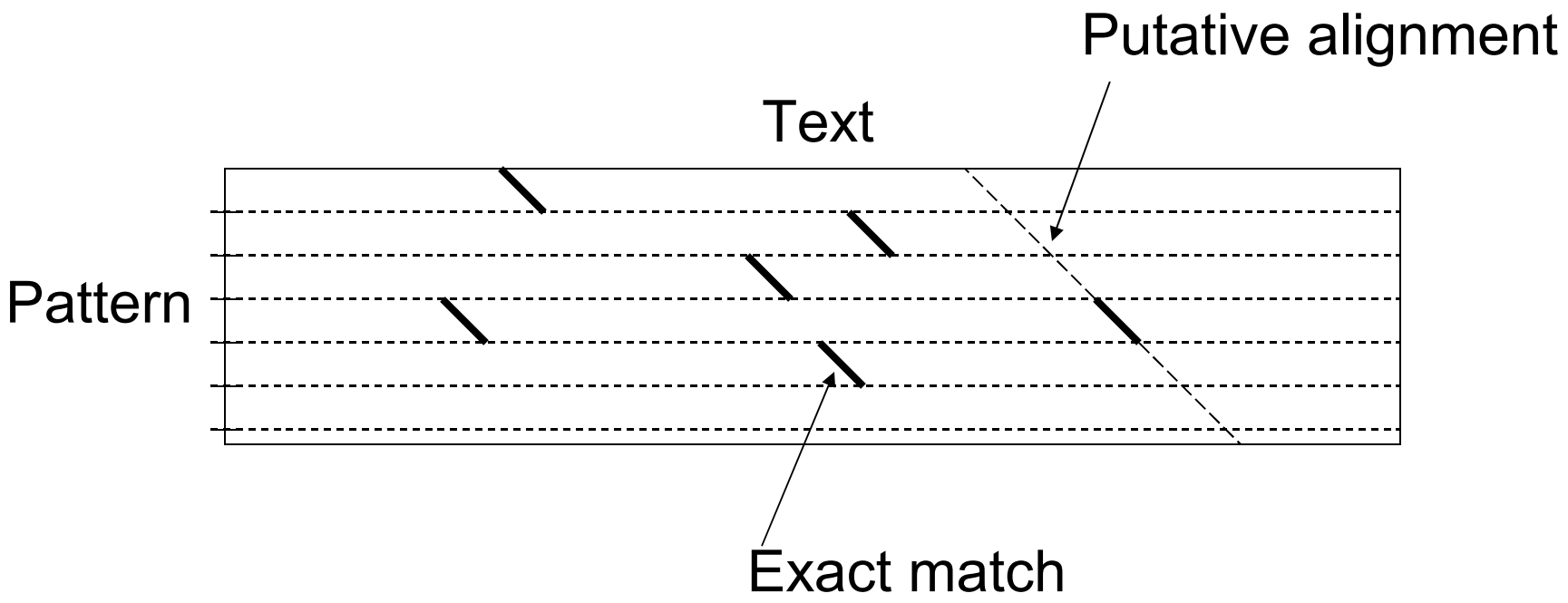
O(km) running time and space

- What if we do not know k? Do binary search to find it

Exclusion methods

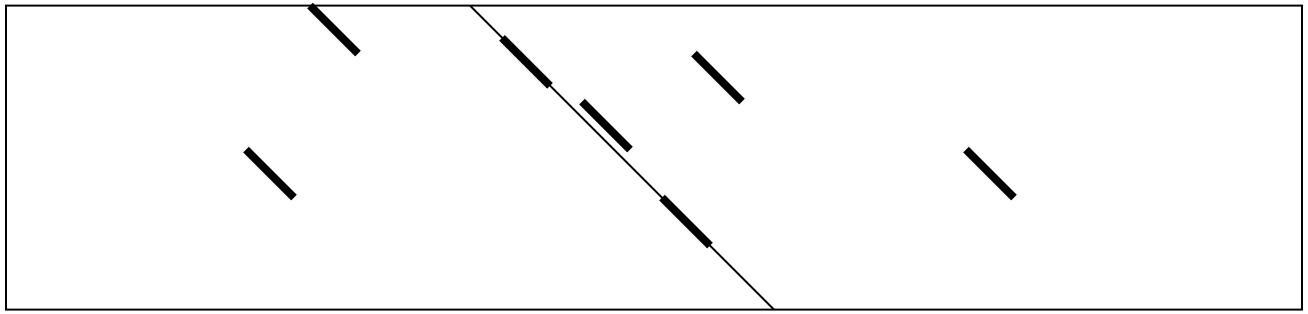
- Assume P must match T with at most k errors. Find places in T where P cannot match.
- Split P into **floor(n/k+1)**-sized chunks.
- If P matches T with less than k errors => at least one chunk matches with no errors
- Use any exact matching algorithm to find places where a chunk matches T, then run dynamic programming in that vicinity.
- Running time, on average O(m)

Exclusion methods



"Famous" approaches

- FASTA (Pearson et al.)
  - Take all k-mers (substrings of length k) from Pattern and identify whether and where they match in the Text
  - Assume the k-mer starting at pos'n i in Pattern matches at position j in Text, remember (j – i) – the diagonal on which the match occured
  - Identify "heavy" diagonals – diagonals where many k-mers match, then refine the diagonals with Smith Waterman
  - Also look for off-diagonal matches to account for gaps

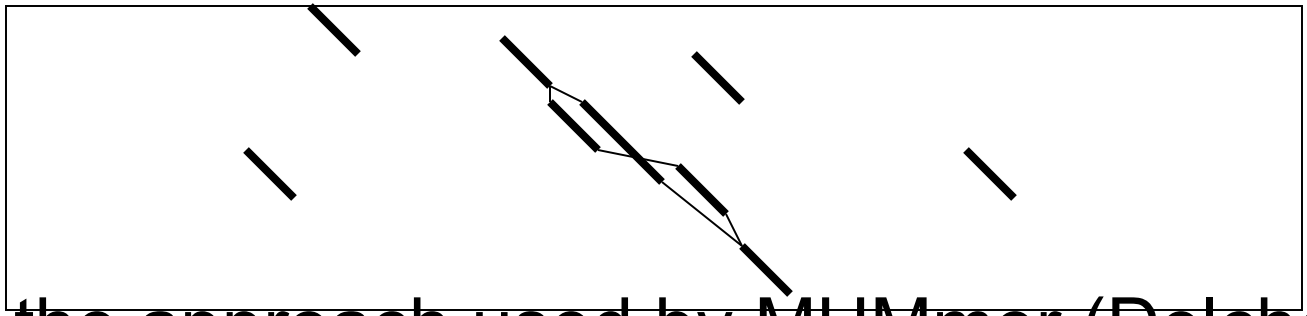


"Famous" approaches

- BLAST (Altschul et al.)
  - Find short k-mer matches
  - Also search for possible inexact matches, e.g. all k-mers within 1 difference from current one.
  - Extend exact matches with Smith-Waterman algorithm
  - Assign probabilistic scores to matches: what is the probability of finding a match with the same S-W alignment score just by chance (e.g. matching a random string)?

Chaining approach

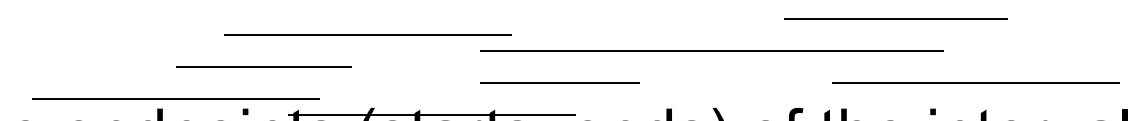
- Extends the FASTA idea
- Search for exact matches
- Find the longest consistent chain of exact matches
- Fill in the gaps in the chain using Smith-Waterman



- This is the approach used by MUMmer (Delcher et al.)
- MUM – maximally unique match (see mummer.sourceforge.net)

Chaining in 1-D

- Input: multiple overlapping intervals on a line
- Output: highest weight set of non-overlapping intervals
- Weight could be length of interval, or Smith-Waterman score, etc.



- Sort the endpoints (starts, ends) of the intervals
- For every interval j, store V[j] – best score of a chain ending in j
- MAX – store highest V[j] seen sofar
- Process endpoints in increasing order of x coordinate
- If we encounter left end (start) of interval j
  - V[j] = weight(j) + MAX
- If we encounter right end (end) of interval j
  - MAX = max{V[j], MAX}
- Running time?