Sequence Alignment (chapter 6)

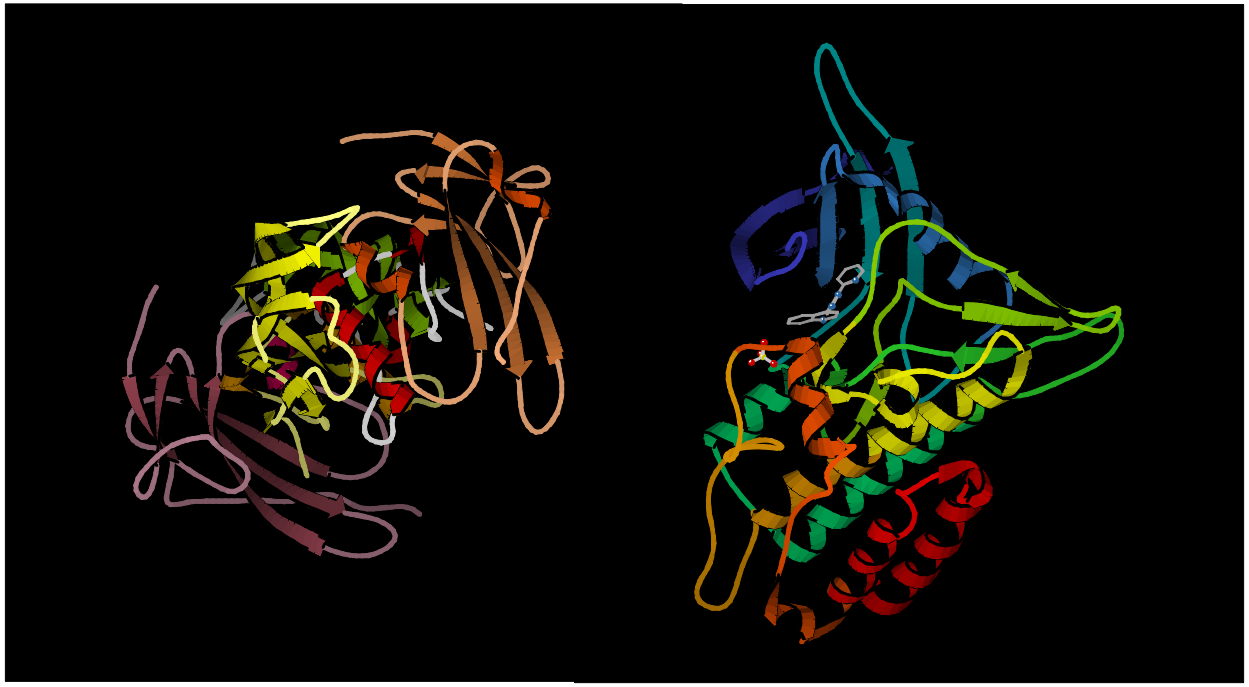
- The biological problem
- Global alignment
- *Local alignment*
- Multiple alignment

Local alignment: rationale

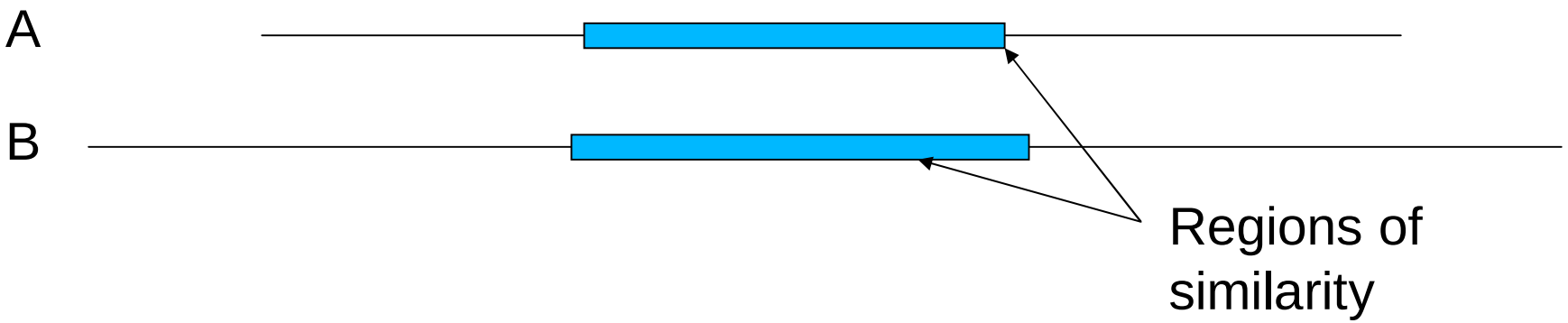
- Otherwise dissimilar proteins may have local regions of similarity
  -> Proteins may share a function

Human bone morphogenic protein receptor type II precursor (left) has a 300 aa region that resembles 291 aa region in TGF-β receptor (right).

The shared function here is protein kinase.



Local alignment: rationale



- Global alignment would be inadequate
- Problem: find the highest scoring *local* alignment between two sequences
- Previous algorithm with minor modifications solves this problem (Smith & Waterman 1981)

From global to local alignment

- Modifications to the global alignment algorithm
  - Look for the highest-scoring path in the alignment matrix (not necessarily through the matrix), or in other words:
  - Allow preceding and trailing indels without penalty

Scoring local alignments

$A = a_1a_2a_3\ldots a_n$, $B = b_1b_2b_3\ldots b_m$

Let I and J be intervals (substrings) of A and B, respectively:

$$I \subset A \quad J \subset B$$

Best local alignment score:

$$M(A, B) = \max\{S(I, J) : I \subset A, J \subset B\}$$

where S(I, J) is the alignment score for substrings I and J.

Allowing preceding and trailing indels

- First row and column initialised to zero:

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

```
b1  b2  b3
-   -   a1
```

		0	1	2	3	4
		-	b_1	b_2	b_3	b_4
0	-	0	0	0	0	0
1	a_1	0				
2	a_2	0				
3	a_3	0				

Recursion for local alignment

- $M_{i,j} = \max \{$
  - $M_{i-1,j-1} + s(a_i, b_i)$,
  - $M_{i-1,j} - \delta$,
  - $M_{i,j-1} - \delta$,
  - 0
  - $\}$

Allow alignment to start anywhere in sequences

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

Finding best local alignment

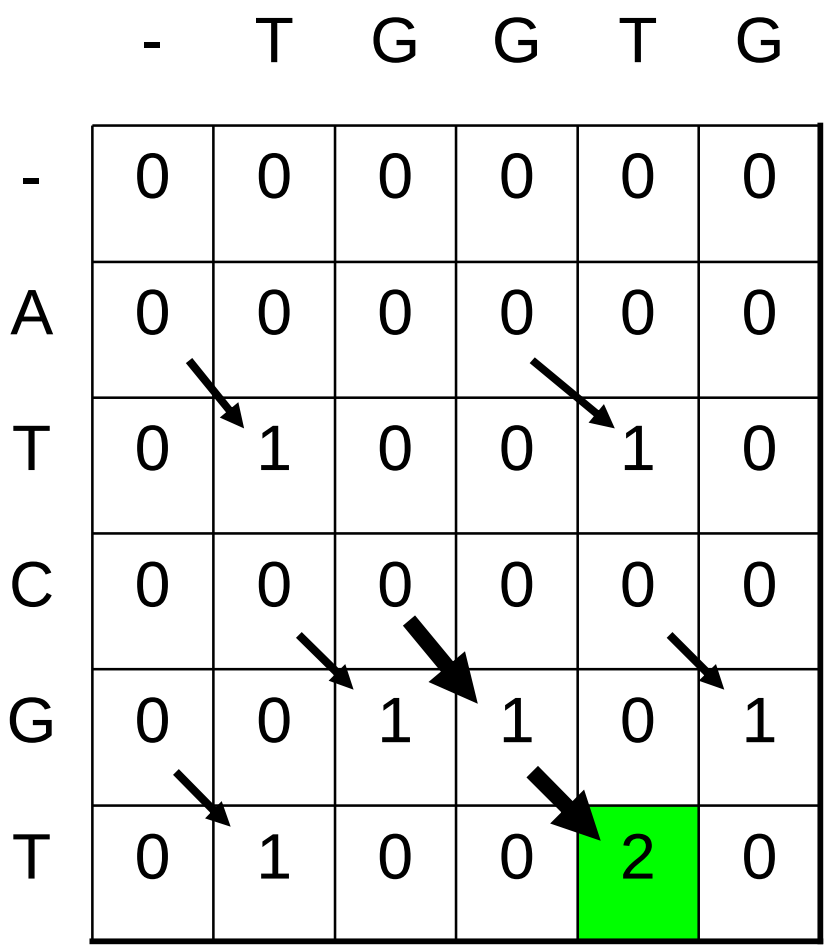
- Optimal score is the highest value in the matrix

$$M(A,B) = \max\{S(I,J) : I \subset A, J \subset B\}$$

$$= \max_{i,j} M_{i,j}$$

- Best local alignment can be found by backtracking from the highest value in M

- What is the best local alignment in this example?



Local alignment: example

$$M_{i,j} = \max \{ M_{i-1,j-1} + s(a_i, b_i),\ M_{i-1,j} - \delta,\ M_{i,j-1} - \delta,\ 0 \}$$

Scoring (for example)
Match: +2
Mismatch: -1
Indel: -2

		0	1	2	3	4	5	6	7	8	9	10
		-	G	G	C	T	C	A	A	T	C	A
0	-	0	0	0	0	0	0	0	0	0	0	0
1	A	0	0									
2	C	0										
3	C	0										
4	T	0										
5	A	0										
6	A	0										
7	G	0										
8	G	0										

Local alignment: example

$$M_{i,j} = \max \{ M_{i-1,j-1} + s(a_i, b_j),\ M_{i-1,j} - \delta,\ M_{i,j-1} - \delta,\ 0 \}$$

Scoring (for example)
Match: +2
Mismatch: -1
Indel: -2

		0	1	2	3	4	5	6	7	8	9	10
		-	G	G	C	T	C	A	A	T	C	A
0	-	0	0	0	0	0	0	0	0	0	0	0
1	A	0	0	0	0	0	0	2				
2	C	0										
3	C	0										
4	T	0										
5	A	0										
6	A	0										
7	G	0										
8	G	0										

Local alignment: example

Optimal local alignment:

C T – A A
C T C A A

		0	1	2	3	4	5	6	7	8	9	10
		-	G	G	C	T	C	A	A	T	C	A
0	-	0	0	0	0	0	0	0	0	0	0	0
1	A	0	0	0	0	0	0	2	2	0	0	2
2	C	0	0	0	2	0	2	0	1	1	2	0
3	C	0	0	0	2	1	2	1	0	0	3	1
4	T	0	0	0	0	4	2	1	0	2	1	2
5	A	0	0	0	0	2	3	4	3	1	1	3
6	A	0	0	0	0	0	1	5	6	4	2	3
7	G	0	2	2	0	0	0	3	4	5	3	1
8	G	0	2	4	2	0	0	1	2	3	4	2

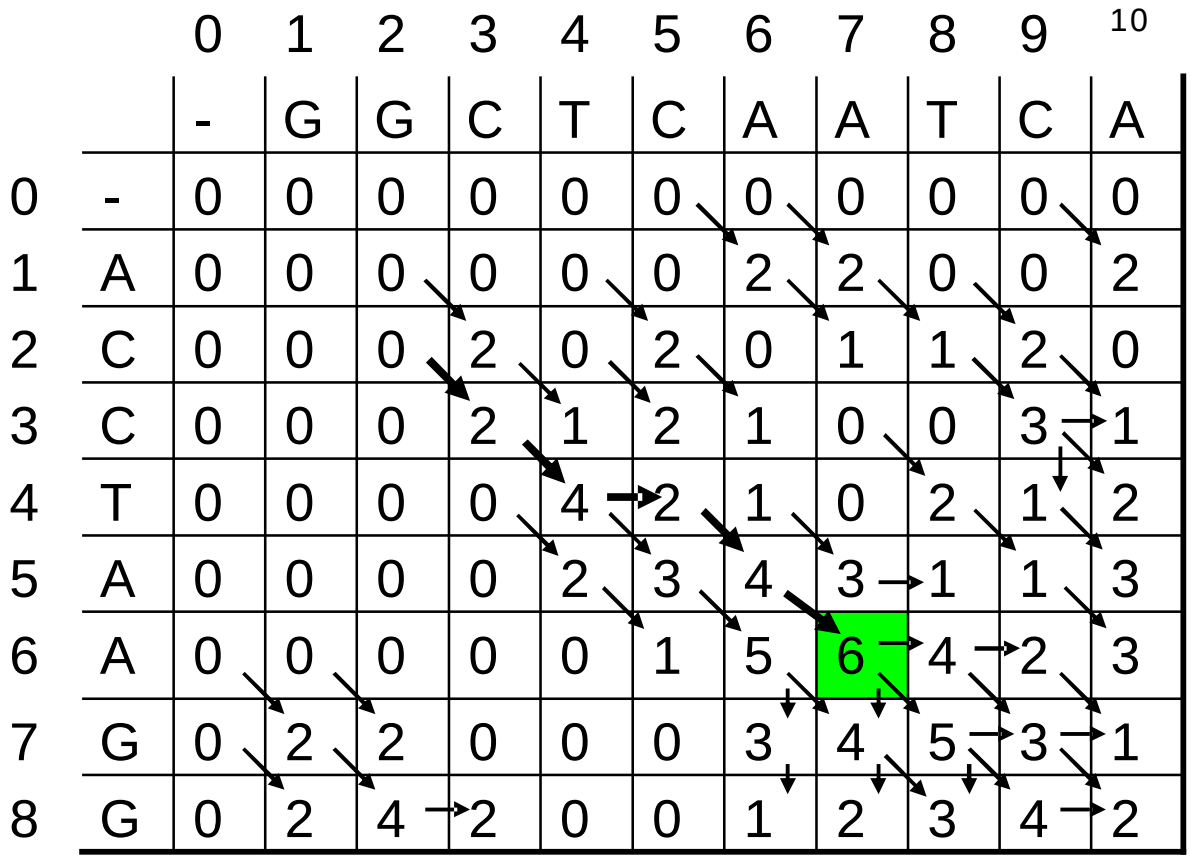
Scoring (for example)
Match: +2
Mismatch: -1
Indel: -2

Multiple optimal alignments
Non-optimal, good-scoring alignments

How can you find

1. Optimal alignments if more than one exist?

2. Non-optimal, good-scoring alignments?



Overlap alignment

- Overlap matrix used by Overlap-Layout-Consensus algorithm can be computed with dynamic programming
- Initialization: $O_{i,0} = O_{0,j} = 0$ for all i, j
- Recursion:

$$O_{i,j} = \max \{ O_{i-1,j-1} + s(a_i, b_i), O_{i-1,j} - \delta, O_{i,j-1} - \delta, \}$$

Best overlap: maximum value from rightmost column and bottom row

Non-uniform mismatch penalties

- We used uniform penalty for mismatches:

  s('A', 'C') = s('A', 'G') = … = s('G', 'T') = μ

- Transition mutations (A->G, G->A, C->T, T->C) are approximately twice as frequent than transversions (A->T, T->A, A->C, G->T)
  - use non-uniform mismatch penalties collected into a *substitution matrix*

	A	C	G	T
A	1	-1	-0.5	-1
C	-1	1	-1	-0.5
G	-0.5	-1	1	-1
T	-1	-0.5	-1	1

Gaps in alignment

- Gap is a succession of indels in alignment

```
C T – - - A A
C T C G C A A
```

- Previous model scored a length k gap as $w(k) = -k\delta$
- Replication processes may produce longer stretches of insertions or deletions
  - In coding regions, insertions or deletions of codons may preserve functionality

Gap open and extension penalties (2)

- We can design a score that allows the penalty opening gap to be larger than extending the gap:

$$w(k) = -\alpha - \beta(k - 1)$$

- Gap open cost α, Gap extension cost β
- Alignment algorithms can be extended to use w(k) (not discussed on this course)

Amino acid sequences

- We have discussed mainly DNA sequences
- Amino acid sequences can be aligned as well
- However, the design of the substitution matrix is more involved because of the larger alphabet
- More on the topic in the course Biological sequence analysis

Demonstration of the EBI web site

- European Bioinformatics Institute (EBI) offers many biological databases and bioinformatics tools at http://www.ebi.ac.uk/
  - Sequence alignment: Tools -> Sequence Analysis -> Align

Sequence Alignment (chapter 6)

- The biological problem
- Global alignment
- Local alignment
- *Multiple alignment*

Multiple alignment

- Consider a set of n sequences on the right
  - Orthologous sequences from different organisms
  - Paralogs from multiple duplications
- How can we study relationships between these sequences?

aggcgagctgcgagtgcta
cgttagattgacgctgac
ttccggctgcgac
gacacggcgaacgga
agtgtgcccgacgagcgaggac
gcgggctgtgagcgcta
aagcggcctgtgtgccccta
atgctgctgccagtgta
agtcgagccccgagtgc
agtccgagtcc
actcggtgc

Optimal alignment of three sequences

- Alignment of A = $a_1a_2...a_i$ and B = $b_1b_2...b_j$ can end either in (-, b_j), (a_i, b_j) or (a_i, -)
- $2^2 - 1 = 3$ alternatives
- Alignment of A, B and C = $c_1c_2...c_k$ can end in $2^3 - 1$ ways: (a_i, -, -), (-, b_j, -), (-, -, c_k), (-, b_j, c_k), (a_i, -, c_k), (a_i, b_j, -) or (a_i, b_j, c_k)
- Solve the recursion using three-dimensional dynamic programming matrix: $O(n^3)$ time and space
- Generalizes to n sequences but impractical with even a moderate number of sequences

Multiple alignment in practice

- In practice, real-world multiple alignment problems are usually solved with heuristics
- Progressive multiple alignment
  - Choose two sequences and align them
  - Choose third sequence w.r.t. two previous sequences and align the third against them
  - Repeat until all sequences have been aligned
  - Different options how to choose sequences and score alignments
  - Note the similarity to Overlap-Layout-Consensus

Multiple alignment in practice

- Profile-based progressive multiple alignment: CLUSTALW
  - Construct a distance matrix of all pairs of sequences using dynamic programming
  - Progressively align pairs in order of decreasing similarity
  - CLUSTALW uses various heuristics to contribute to accuracy

Additional material

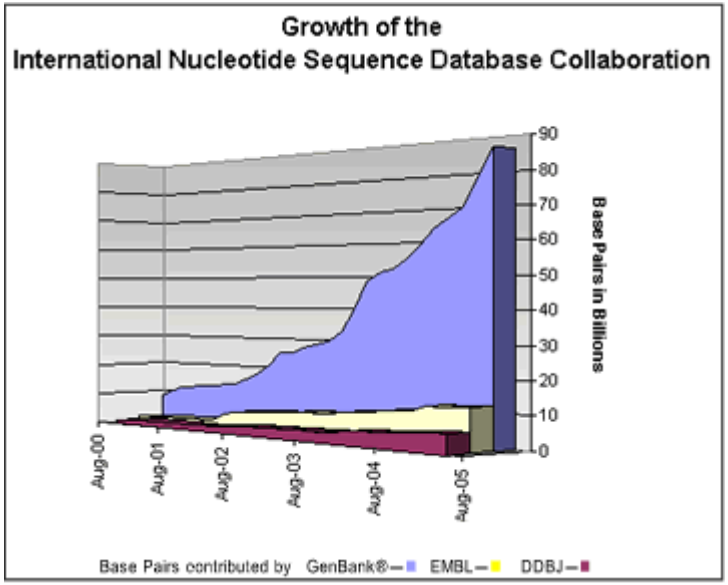
- R. Durbin, S. Eddy, A. Krogh, G. Mitchison: Biological sequence analysis
- N. C. Jones, P. A. Pevzner: An introduction to bioinformatics algorithms
- Course Biological sequence analysis in period II, 2008

Rapid alignment methods: FASTA and BLAST

- *The biological problem*
- *Search strategies*
- FASTA
- BLAST

The biological problem

- Global and local alignment algoritms are slow in practice
- Consider the scenario of aligning a *query sequence* against a large database of sequences
  - New sequence with unknown function



- NCBI GenBank size in January 2007 was 65 369 091 950 bases (61 132 599 sequences)
- Feb 2008: 85 759 586 764 bases (82 853 685 sequences)

Problem with large amount of sequences

- Exponential growth in both number and total length of sequences
- Possible solution: Compare against model organisms only
- With large amount of sequences, chances are that matches occur by random
  - Need for statistical analysis

Rapid alignment methods: FASTA and BLAST

- The biological problem
- Search strategies
- *FASTA*
- BLAST

FASTA

- FASTA is a multistep algorithm for sequence alignment (Wilbur and Lipman, 1983)
- The sequence file format used by the FASTA software is widely used by other sequence analysis software
- Main idea:
  - Choose regions of the two sequences (query and database) that look promising (have some degree of similarity)
  - Compute local alignment using dynamic programming in these regions

FASTA outline

- FASTA algorithm has five steps:
  - *1. Identify common k-words between I and J*
  - 2. Score diagonals with k-word matches, identify 10 best diagonals
  - 3. Rescore initial regions with a substitution score matrix
  - 4. Join initial regions using gaps, penalise for gaps
  - 5. Perform dynamic programming to find final alignments

Search strategies

- How to speed up the computation?
  - Find ways to limit the number of pairwise comparisons
- Compare the sequences at word level to find out common words
  - Word means here a k-tuple (or a k-word), a substring of length k

Analyzing the word content

- Example query string I: TGATGATGAAGACATCAG
- For k = 8, the set of k-words (substring of length k) of I is

TGATGATG

GATGATGA

ATGATGAA

TGATGAAG

...

GACATCAG

Analyzing the word content

- There are n-k+1 k-words in a string of length n
- If at least one word of I is not found from another string J, we know that I differs from J
- Need to consider statistical significance: I and J might share words by chance only
- Let n=|I| and m=|J|

Word lists and comparison by content

- The k-words of I can be arranged into a table of word occurences $L_w(I)$
- Consider the k-words when k=2 and I=GCATCGGC:

GC, CA, AT, TC, CG, GG, GC

AT: 3

CA: 2

CG: 5

GC: 1, 7 ← Start indecies of k-word GC in I

GG: 6

TC: 4

Building $L_w(I)$ takes O(n) time

Common k-words

- Number of common k-words in I and J can be computed using $L_w(I)$ and $L_w(J)$
- For each word w in I, there are $|L_w(J)|$ occurences in J
- Therefore I and J have $\sum_w |L_w(I)||L_w(J)|$ common words
- This can be computed in $O(n + m + 4^k)$ time
  - $O(n + m)$ time to build the lists
  - $O(4^k)$ time to calculate the sum (in DNA strings)

Common k-words

- I = GCATCGGC
- J = CCATCGCCATCG

L_w(I)	L_w(J)	Common words
AT: 3	AT: 3, 9	2
CA: 2	CA: 2, 8	2
	CC: 1, 7	0
CG: 5	CG: 5, 11	2
GC: 1, 7	GC: 6	2
GG: 6		0
TC: 4	TC: 4, 10	2
		10 in total

Properties of the common word list

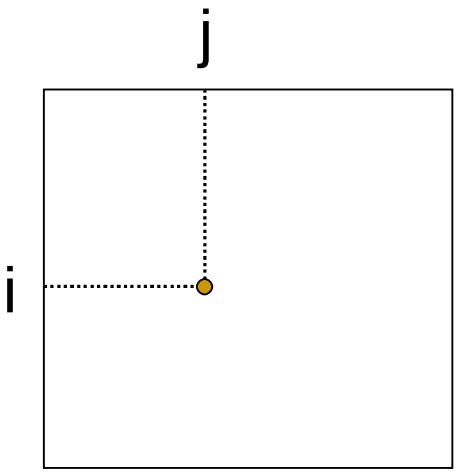
- Exact matches can be found using binary search (e.g., where TCGT occurs in I?)
  - $O(\log 4^k)$ time
- For large k, the table size is too large to compute the common word count in the previous fashion
- Instead, an approach based on merge sort can be utilised (details skipped)
- The common k-word technique can be combined with the local alignment algorithm to yield a rapid alignment approach

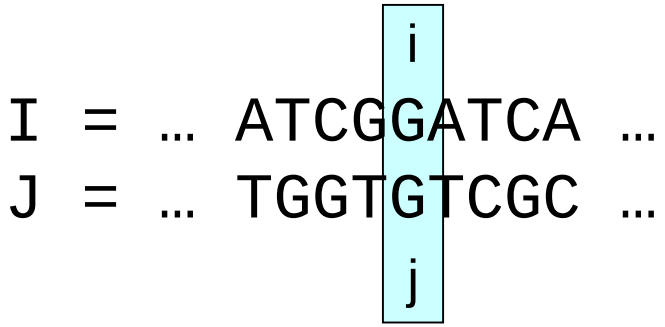
FASTA outline

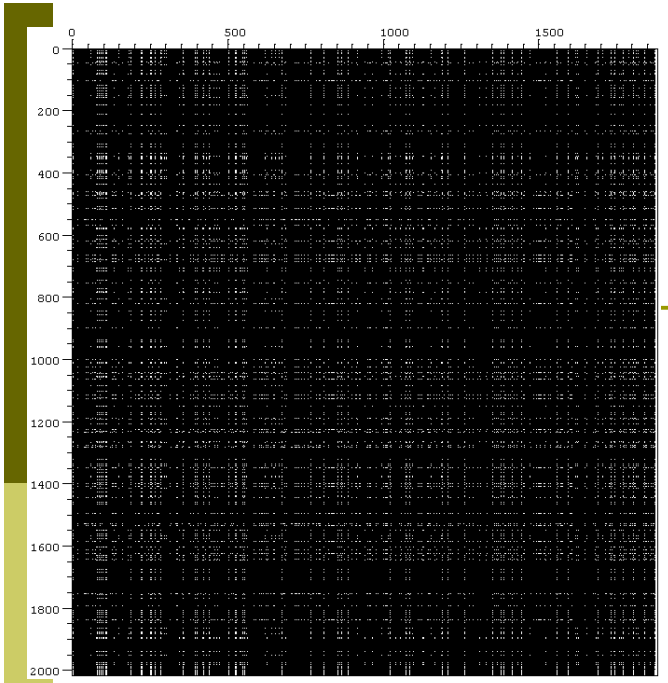
- FASTA algorithm has five steps:
  - 1. Identify common k-words between I and J
  - *2. Score diagonals with k-word matches, identify 10 best diagonals*
  - 3. Rescore initial regions with a substitution score matrix
  - 4. Join initial regions using gaps, penalise for gaps
  - 5. Perform dynamic programming to find final alignments

Dot matrix comparisons

- Word matches in two sequences I and J can be represented as a *dot matrix*
- Dot matrix element (i, j) has ”a dot”, if the word starting at position i in I is identical to the word starting at position j in J
- The dot matrix can be plotted for various k

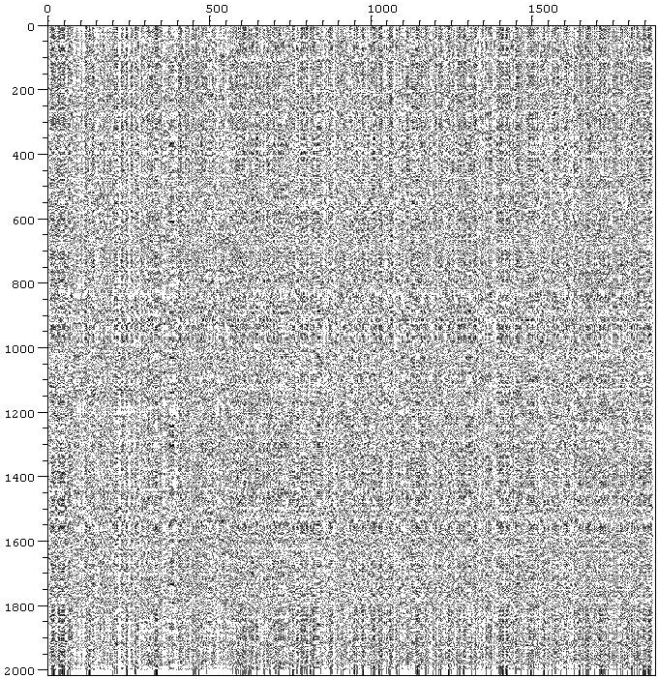




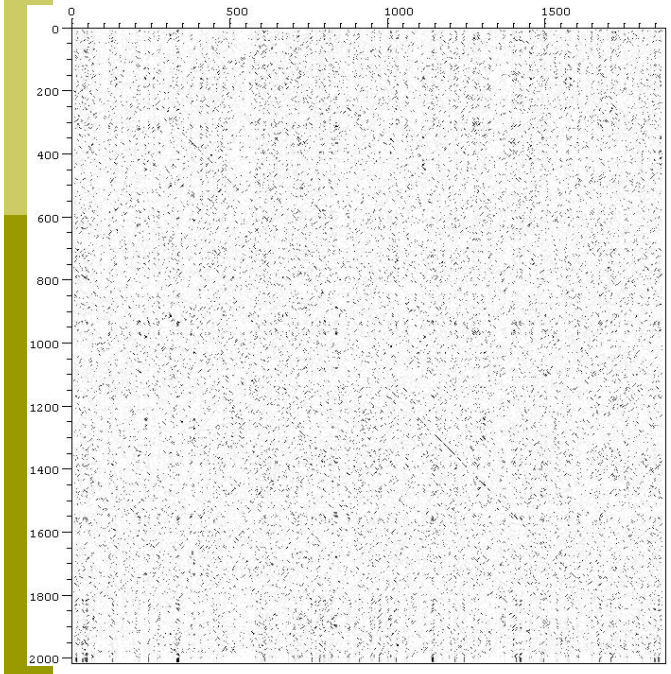


k=1

k=4

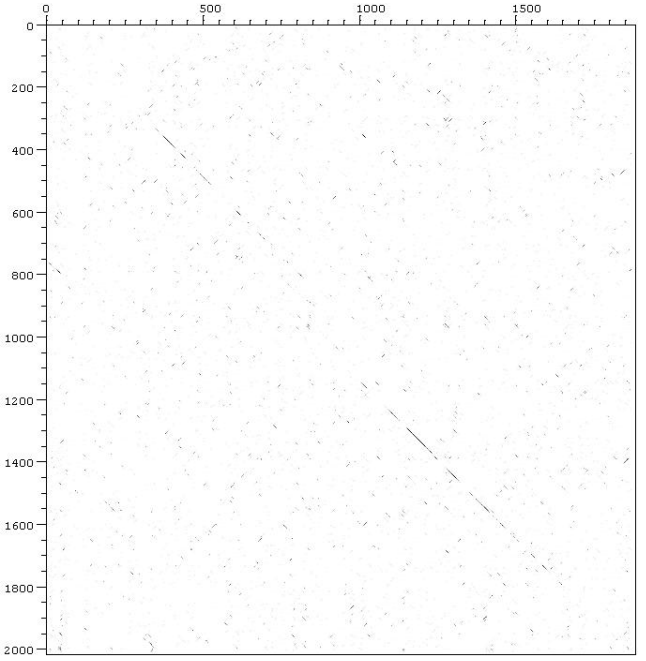


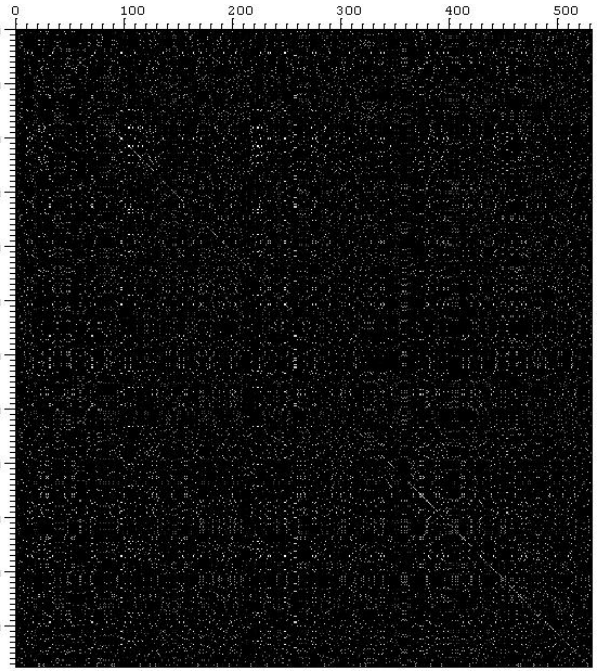
Dot matrix (k=1,4,8,16)
for two **DNA** sequences
X85973.1 (1875 bp)
Y11931.1 (2013 bp)



k=8

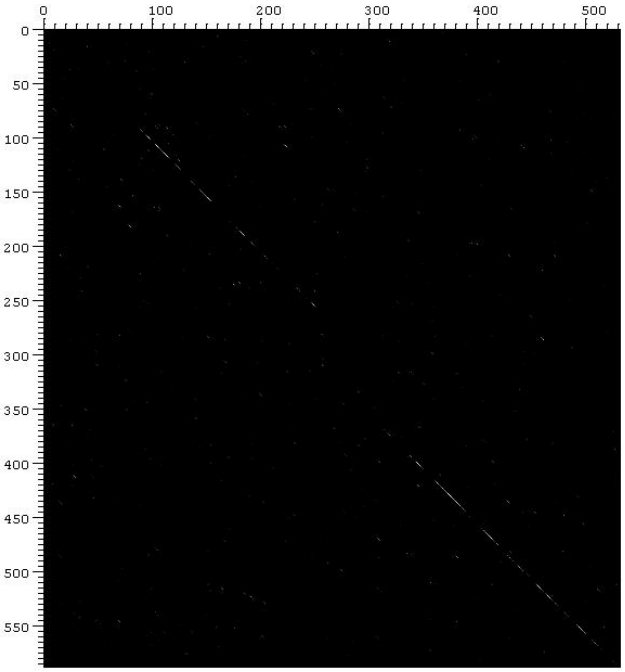
k=16



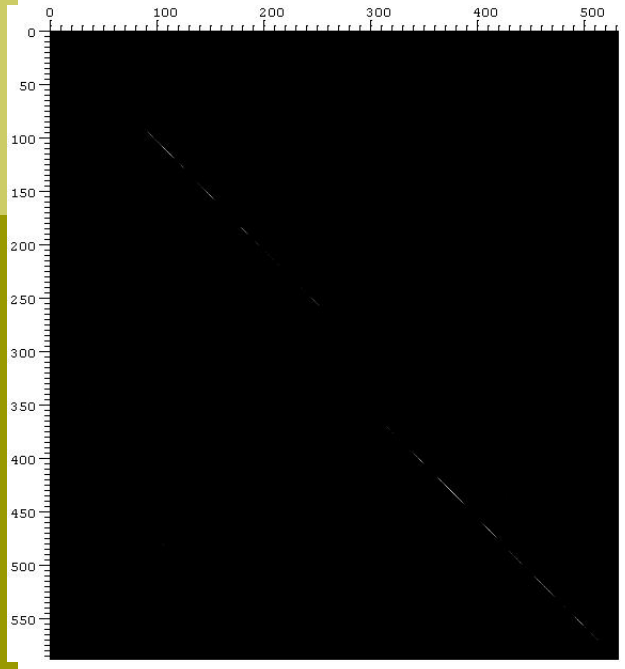


k=1

k=4

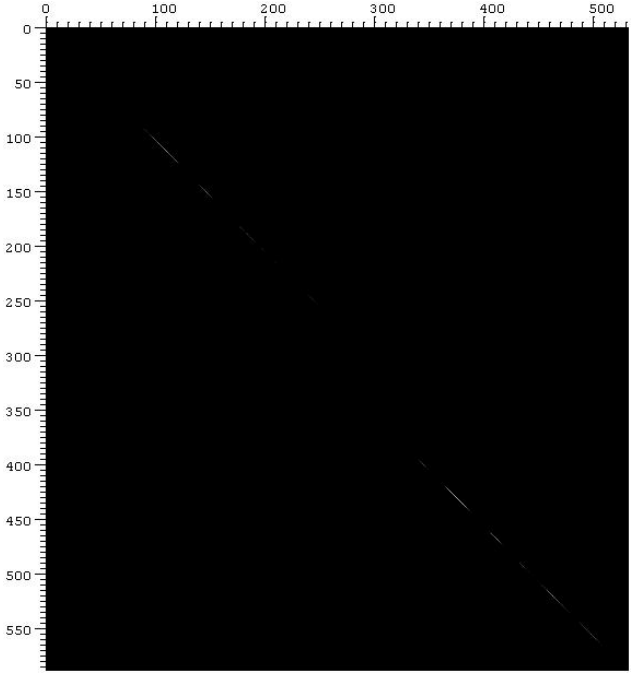


Dot matrix (k=1,4,8,16) for two **protein** sequences
CAB51201.1 (531 aa)
CAA72681.1 (588 aa)



k=8

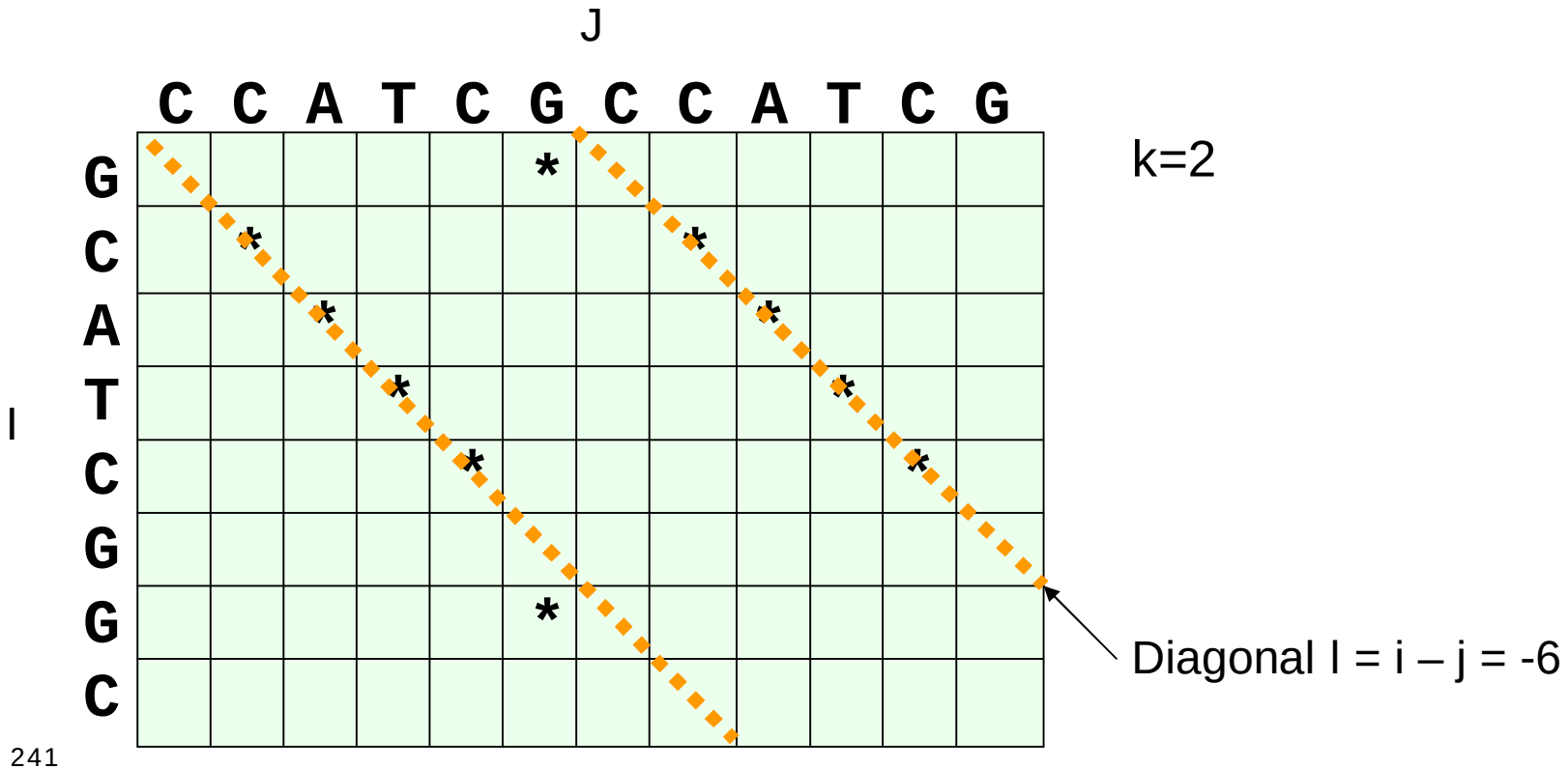
k=16



Shading indicates now the match score according to a score matrix (Blosum62 here)

Computing diagonal sums

- We would like to find high scoring diagonals of the dot matrix
- Lets index diagonals by the offset, l = i - j



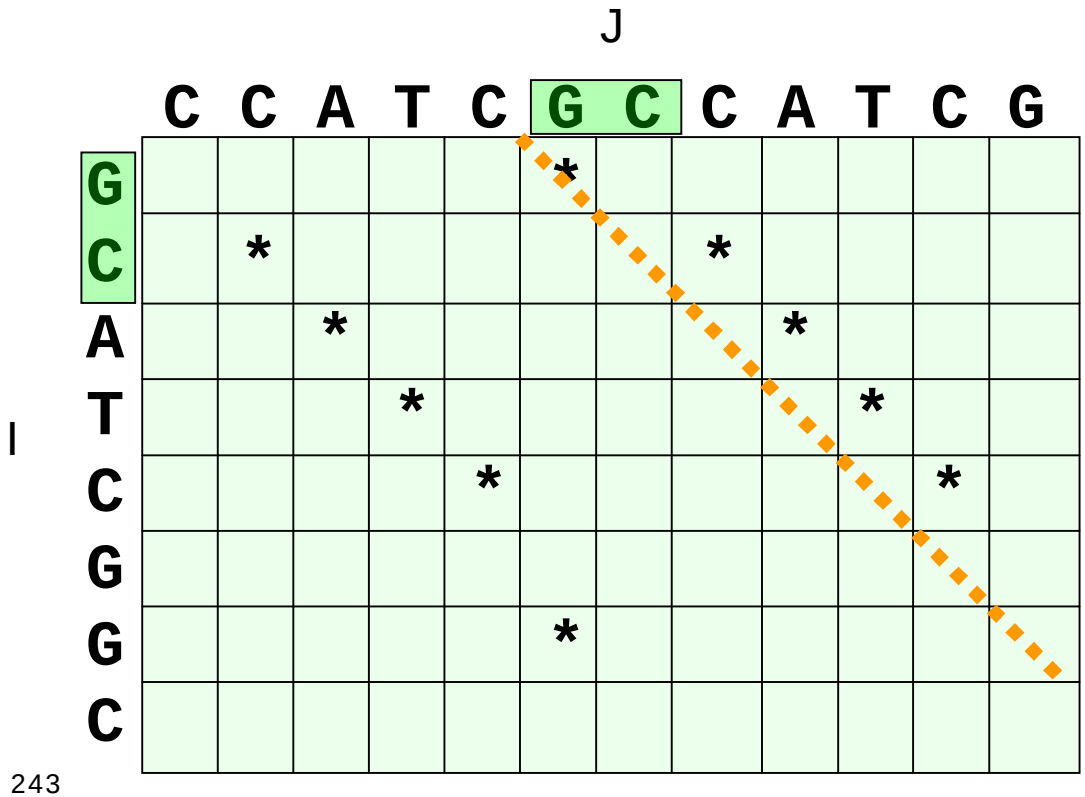
Computing diagonal sums

- As an example, lets compute diagonal sums for I = GCATCGGC, J = CCATCGCCATCG, k = 2
- 1. Construct k-word list $L_w(J)$
- 2. Diagonal sums S_l are computed into a table, indexed with the offset and initialised to zero

l	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
S_l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Computing diagonal sums

- 3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums

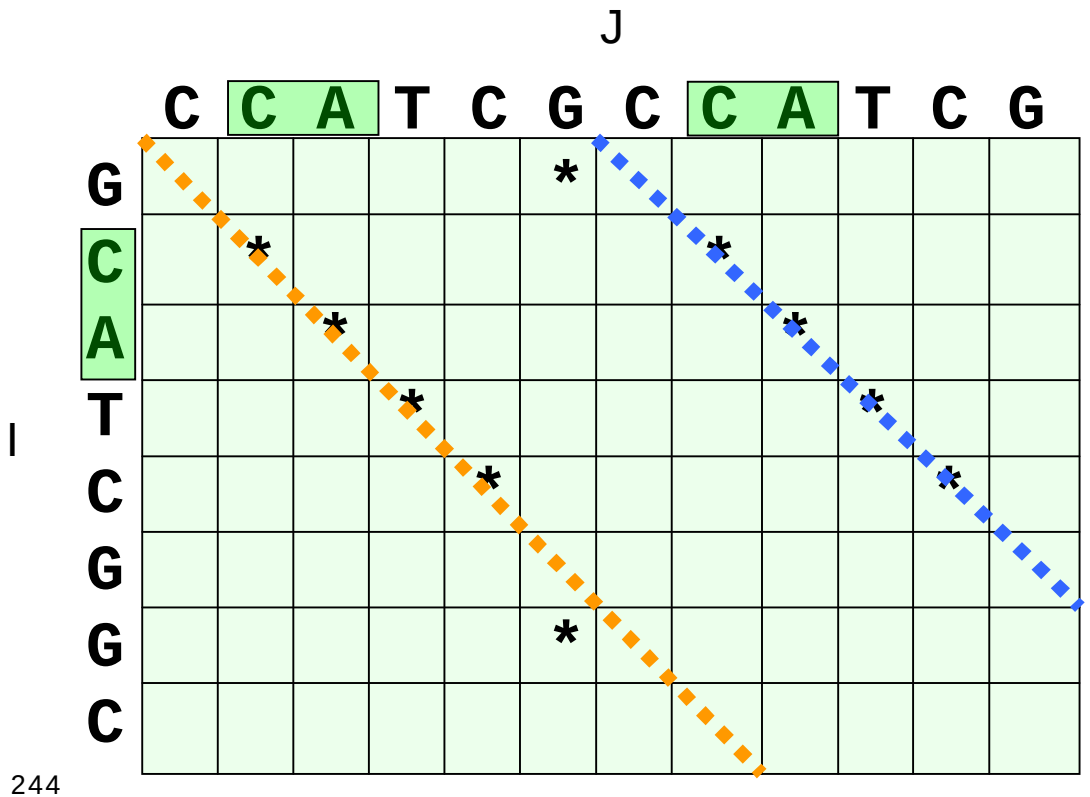


For the first 2-word in I, GC, $L_{GC}(J) = \{6\}$.

We can then update the sum of diagonal $I = i - j = 1 - 6 = -5$ to $S_{-5} := S_{-5} + 1 = 0 + 1 = 1$

Computing diagonal sums

- 3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums



Next 2-word in I is CA, for which $L_{CA}(J) = \{2, 8\}$.

Two diagonal sums are updated:

$I = i - j = 2 - 2 = 0$

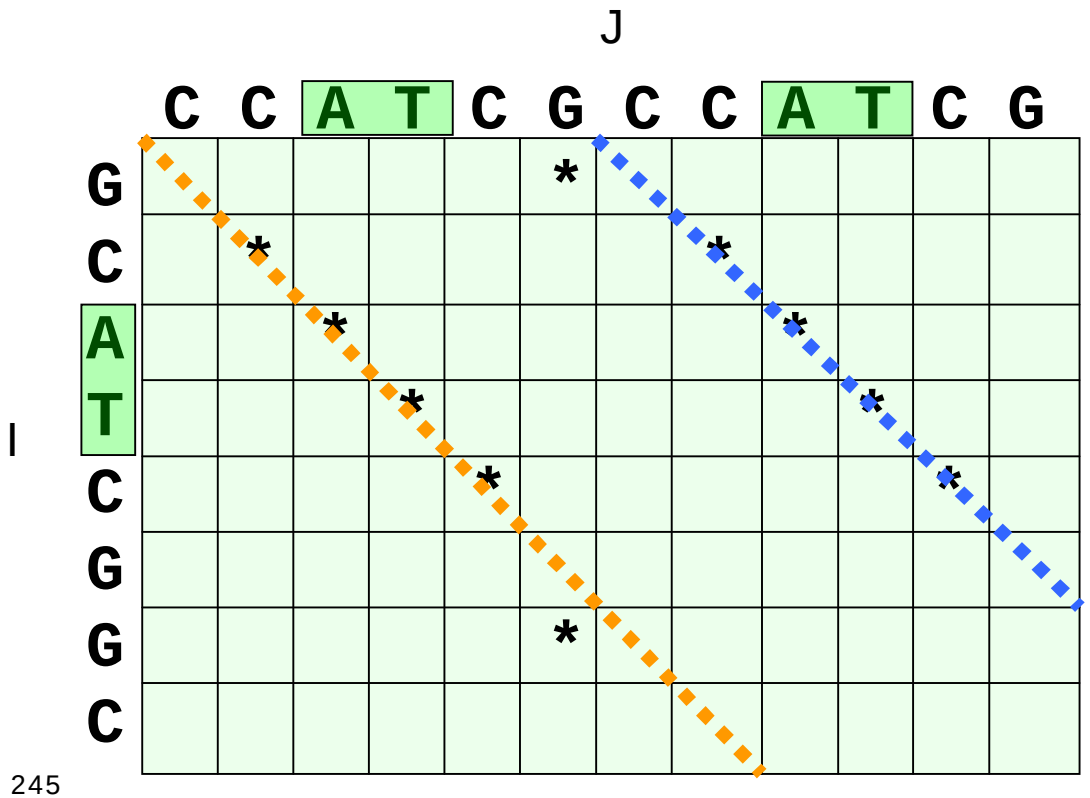
$S_0 := S_0 + 1 = 0 + 1 = 1$

$I = i - j = 2 - 8 = -6$

$S_{-6} := S_{-6} + 1 = 0 + 1 = 1$

Computing diagonal sums

- 3. Go through k-words of I, look for matches in $L_w(J)$ and update diagonal sums



Next 2-word in I is AT, for which $L_{AT}(J) = \{3, 9\}$.

Two diagonal sums are updated:

$I = i - j = 3 - 3 = 0$

$S_0 := S_0 + 1 = 1 + 1 = 2$

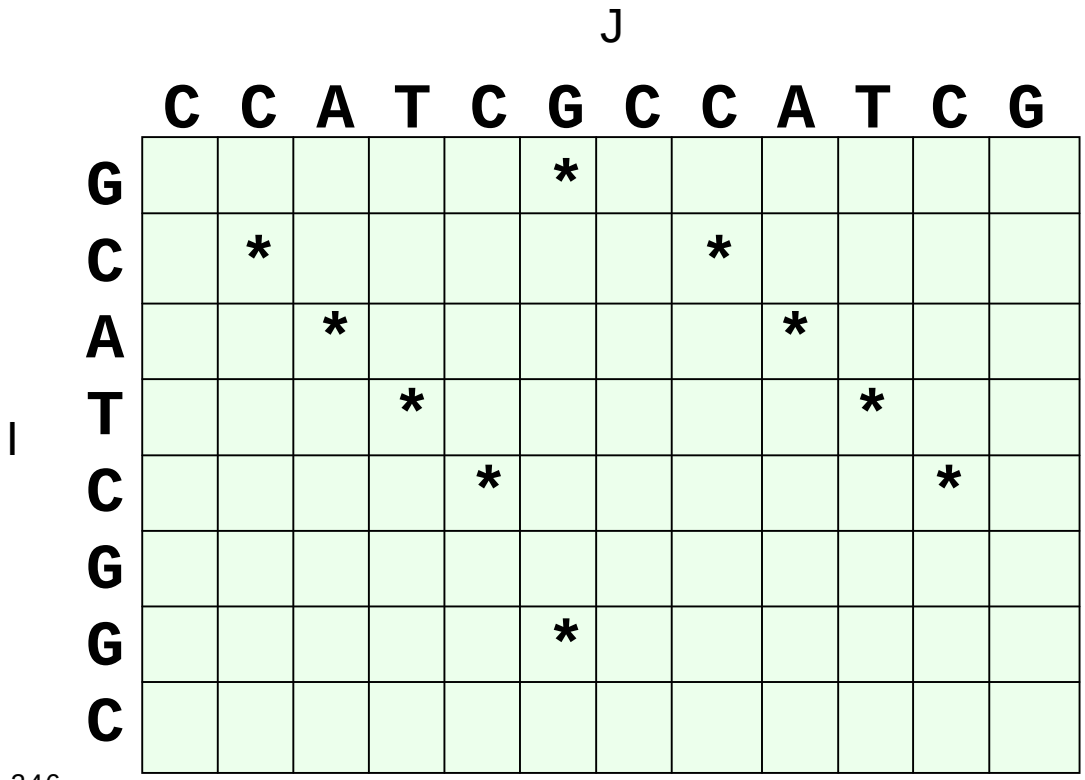
$I = i - j = 3 - 9 = -6$

$S_{-6} := S_{-6} + 1 = 1 + 1 = 2$

Computing diagonal sums

After going through the k-words of I, the result is:

l	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
S_l	0	0	0	0	4	1	0	0	0	0	4	1	0	0	0	0	0



Algorithm for computing diagonal sum of scores

$S_l := 0$ for all $1 - m \leq l \leq n - 1$

Compute $L_w(J)$ for all words w

for $i := 1$ to $n - k - 1$ do

  $w := I_i I_{i+1} \ldots I_{i+k-1}$

  for $j \in L_w(J)$ do

    $l := i - j$

    $S_l := S_l + 1$ ← Match score is here 1

  end

end

FASTA outline

- FASTA algorithm has five steps:
  - 1. Identify common k-words between I and J
  - 2. Score diagonals with k-word matches, identify 10 best diagonals
  - *3. Rescore initial regions with a substitution score matrix*
  - *4. Join initial regions using gaps, penalise for gaps*
  - 5. Perform dynamic programming to find final alignments

Rescoring initial regions

- Each high-scoring diagonal chosen in the previous step is rescored according to a score matrix
- This is done to find subregions with identities shorter than k
- Non-matching ends of the diagonal are trimmed

```
I: C C A T C G C C A T C G
J: C C A A C G C A A T C A
```

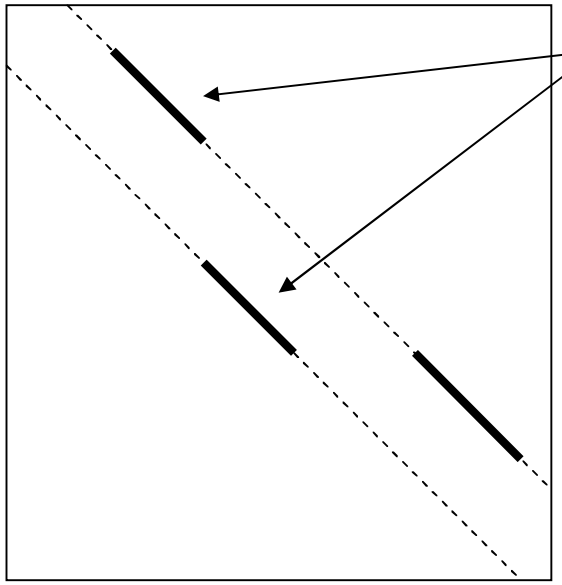
75% identity, no 4-word identities

```
I': C C A T C G C C A T C G
J': A C A T C A A A T A A A
```

33% identity, one 4-word identity

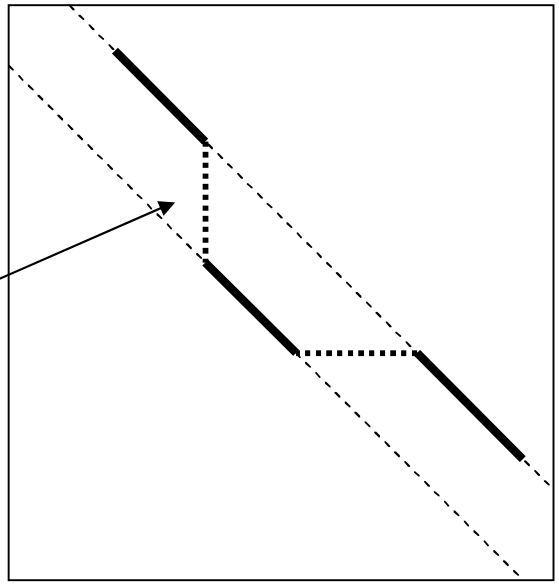
Joining diagonals

- Two offset diagonals can be joined with a gap, if the resulting alignment has a higher score
- Separate gap open and extension are used
- Find the best-scoring combination of diagonals



High-scoring diagonals

Two diagonals joined by a gap

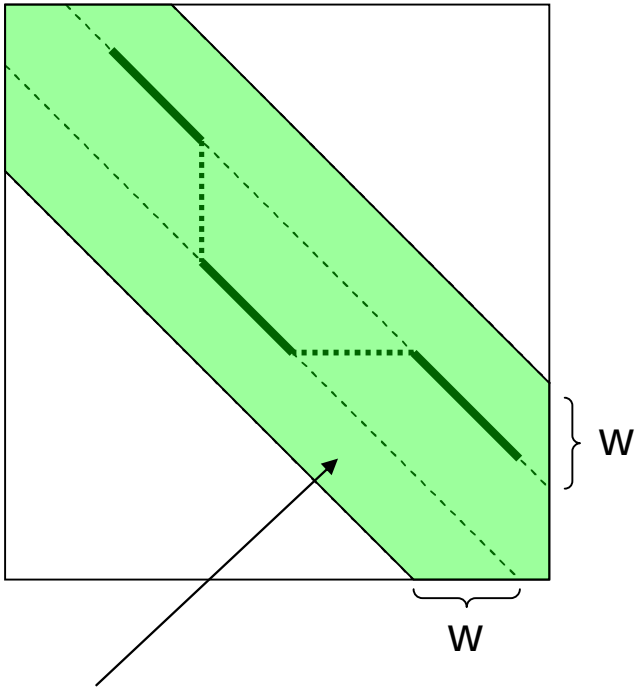


FASTA outline

- FASTA algorithm has five steps:
  - 1. Identify common k-words between I and J
  - 2. Score diagonals with k-word matches, identify 10 best diagonals
  - 3. Rescore initial regions with a substitution score matrix
  - 4. Join initial regions using gaps, penalise for gaps
  - *5. Perform dynamic programming to find final alignments*

Local alignment in the highest-scoring region

- Last step of FASTA: perform local alignment using dynamic programming around the highest-scoring
- Region to be aligned covers –w and +w offset diagonal to the highest-scoring diagonals
- With long sequences, this region is typically very small compared to the whole n x m matrix



Dynamic programming matrix
M filled only for the green region

Properties of FASTA

- Fast compared to local alignment using dynamic programming only
  - Only a narrow region of the full matrix is aligned
- Increasing parameter k decreases the number of hits:
  - Increases specificity
  - Decreases sensitivity
  - Decreases running time
- FASTA can be very specific when identifying long regions of low similarity
  - Specific method does not find many incorrect results
  - Sensitive method finds many of the correct results

Properties of FASTA

- FASTA looks for initial exact matches to query sequence
  - Two proteins can have very different amino acid sequences and still be biologically similar
  - This may lead into a lack of sensitivity with diverged sequences

Demonstration of FASTA at EBI

- http://www.ebi.ac.uk/fasta/
- Note that parameter ktup in the software corresponds to parameter k in lectures