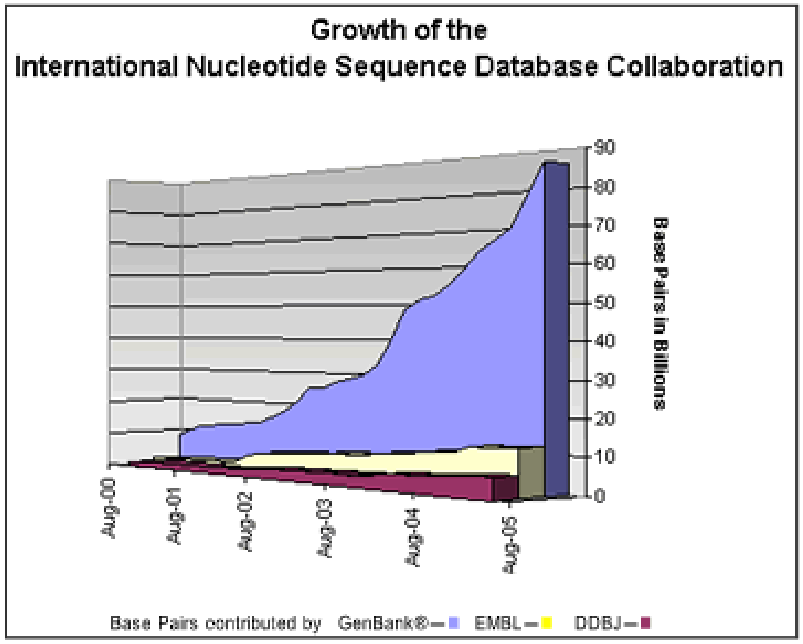
Chapter 7: Rapid alignment methods: FASTA and BLAST

- *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



- For instance, the size of NCBI GenBank in January 2007 was 65,369,091,950 bases (61,132,599 sequences)

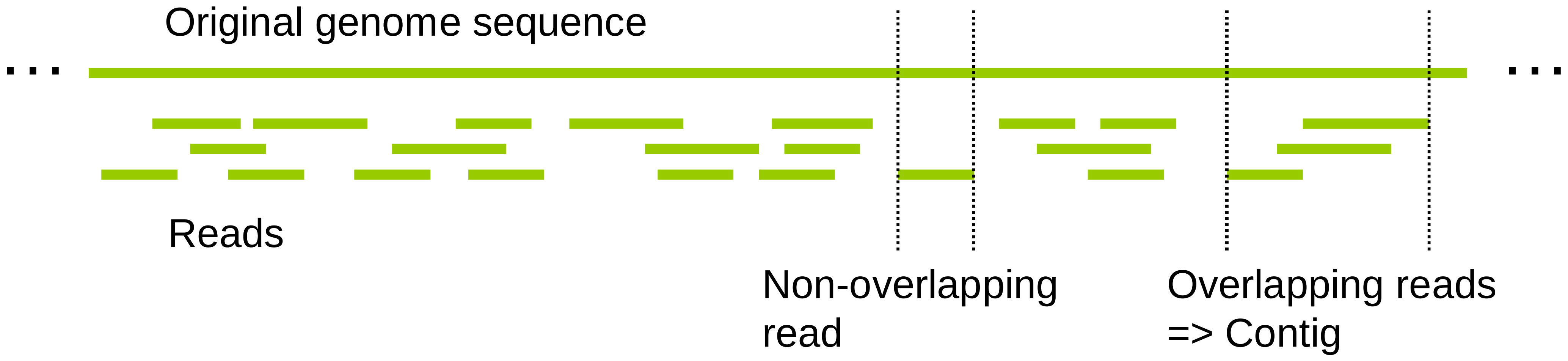
Problem with large amount of sequences

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

Application of sequence alignment: shotgun sequencing

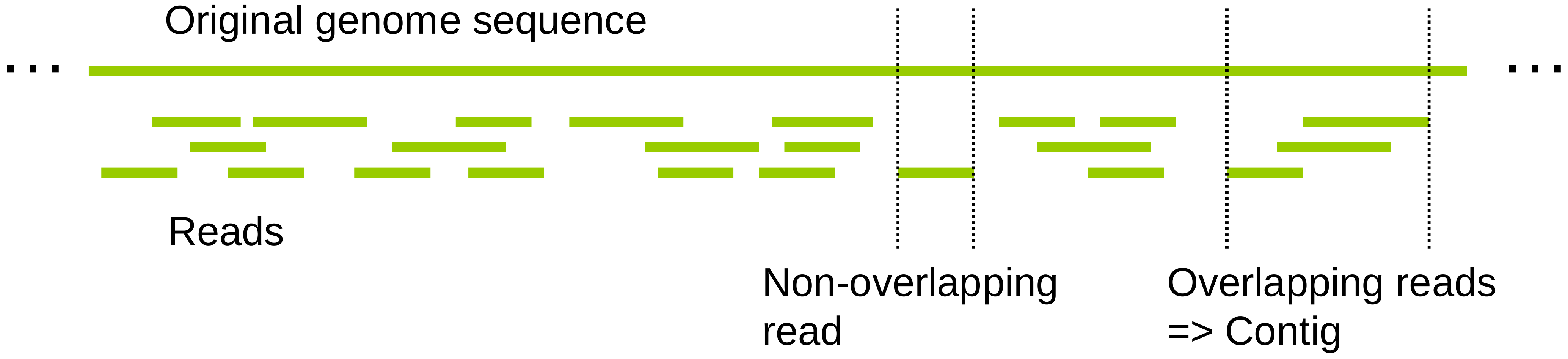
- Shotgun sequencing is a method for sequencing whole-organism genomes
  - First, a large number of short sequences (~500-1000 bp), or *reads* are generated from the genome
  - Reads are contiguous subsequences (substrings) of the genome
  - Due to sequencing errors and repetitions in the reads, the genome has be covered multiple times by reads

Shotgun sequencing



- Ordering of the reads is initially unknown
- Overlaps resolved by aligning the reads
- In a 3×10^9 bp genome with 500 bp reads and 5x coverage, there are ~10^7 reads and ~$10^7(10^7-1)/2$ = ~5×10^{13} pairwise sequence comparisons

Shotgun sequencing



- ~5x10^{13} pairwise sequence comparisons
- Recall that local alignment takes O(nm) time, where n and m are sequence lengths
- Already with n=m=500, the computation cost is prohibitive

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-tuples of I is

```
TGATGATG
 GATGATGA
  ATGATGAA
   TGATGAAG
       …
          GACATCAG
```

Analyzing the word content

- There are n-k+1 k-tuples 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

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, see course book)
- The common k-word technique can be combined with the local alignment algorithm to yield a rapid alignment approach

Chapter 7: Rapid alignment methods: FASTA and BLAST

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

FASTA

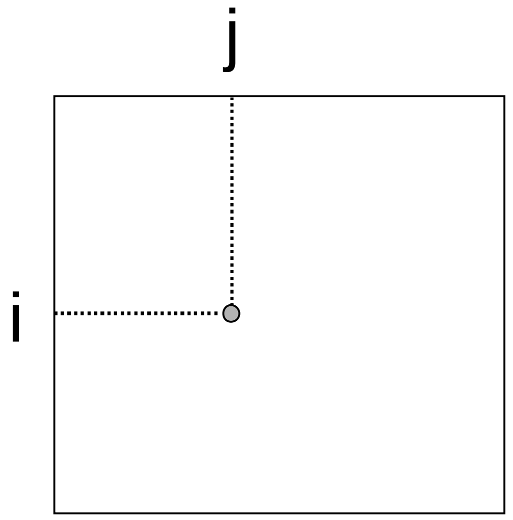
- FASTA is a multistep algorithm for sequence alignment (Wilbur and Lipman, 1983)
- The sequence file format used by the FASTA software is widely used by other sequence analysis software
- Main idea:
  - Choose regions of the two sequences that look promising (have some degree of similarity)
  - Compute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

FASTA outline

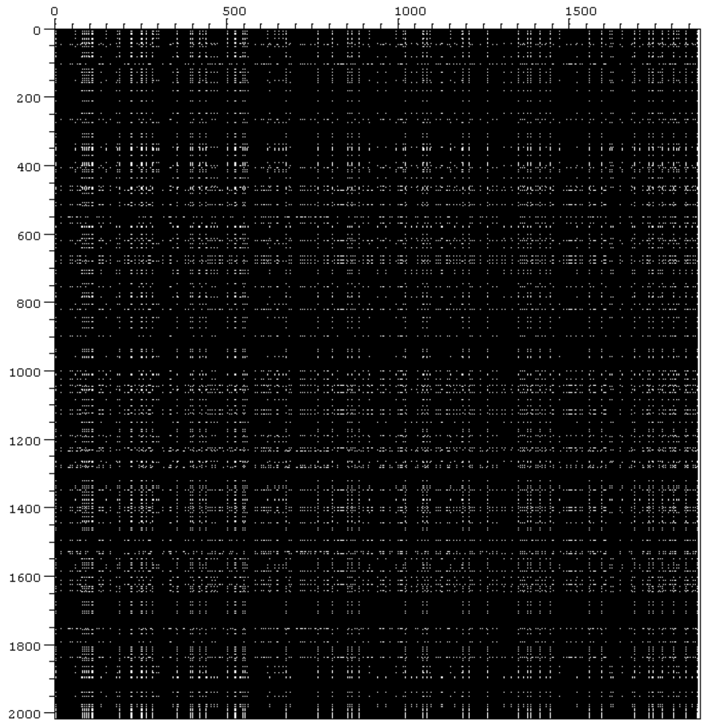
- FASTA algorithm has five steps:
  - 1. Identify common k-words between I and J
  - *2. Score diagonals with k-word matches, identify 10 best diagonals*
  - 3. Rescore initial regions with a substitution score matrix
  - 4. Join initial regions using gaps, penalise for gaps
  - 5. Perform dynamic programming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

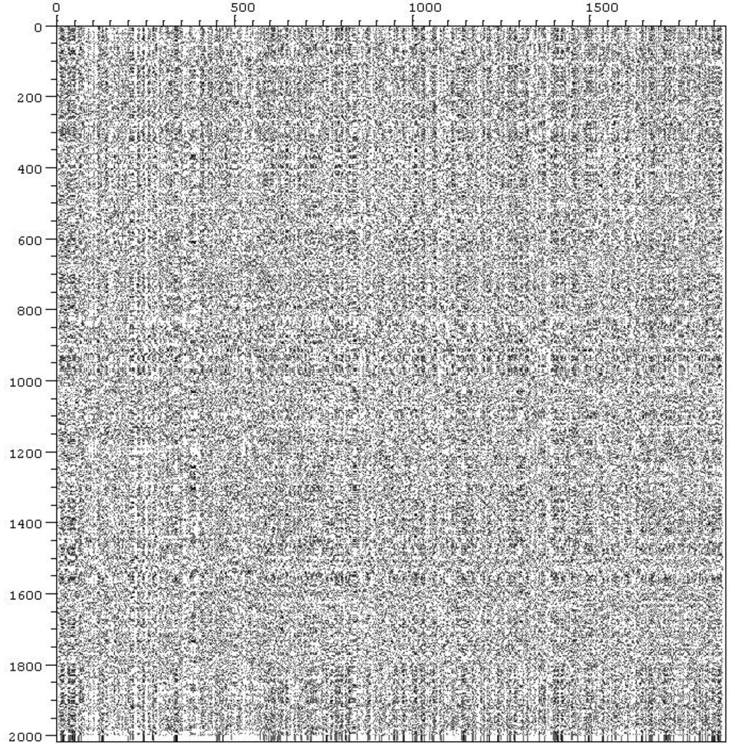
Dot matrix comparisons

- 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

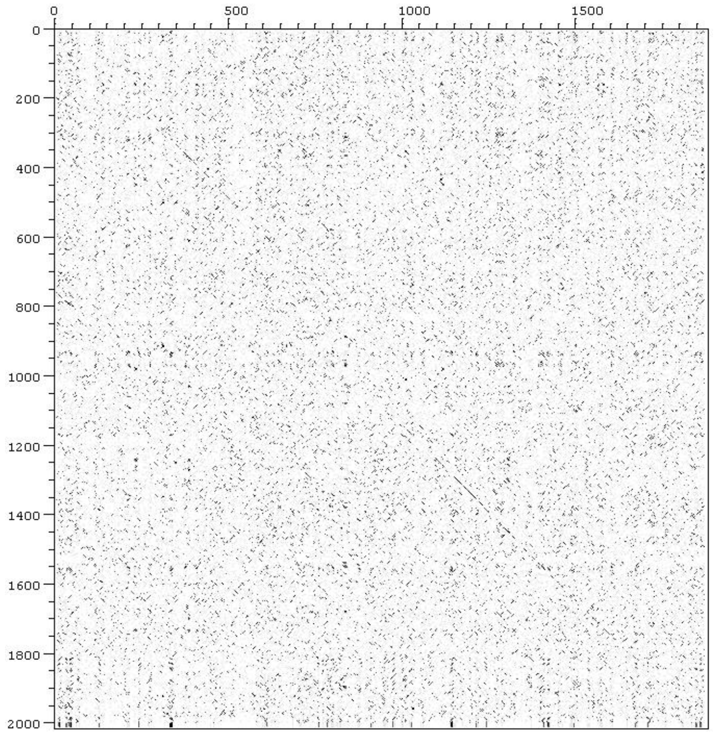


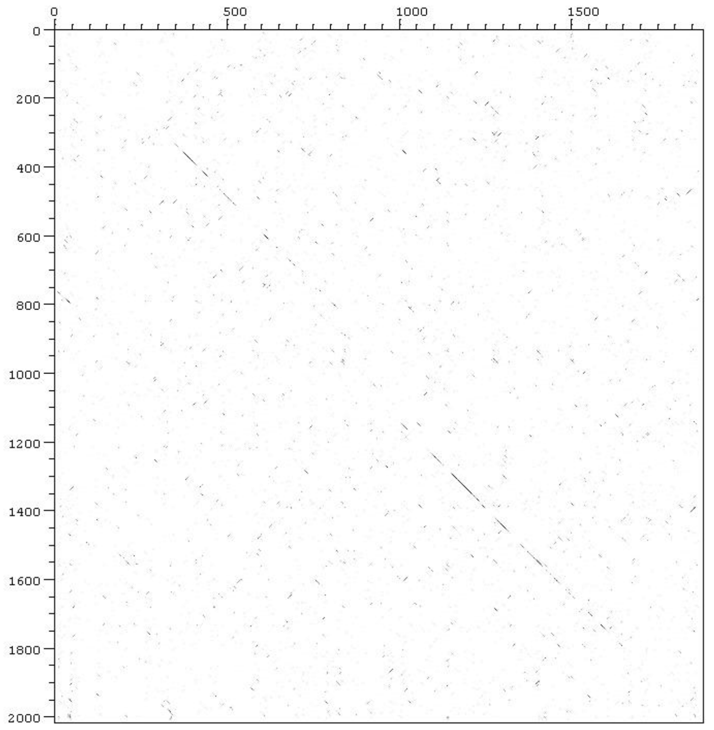
```
                  i
I = … ATCGGATCA …
J = … TGGTGTCGC …
                  j
```


k=1

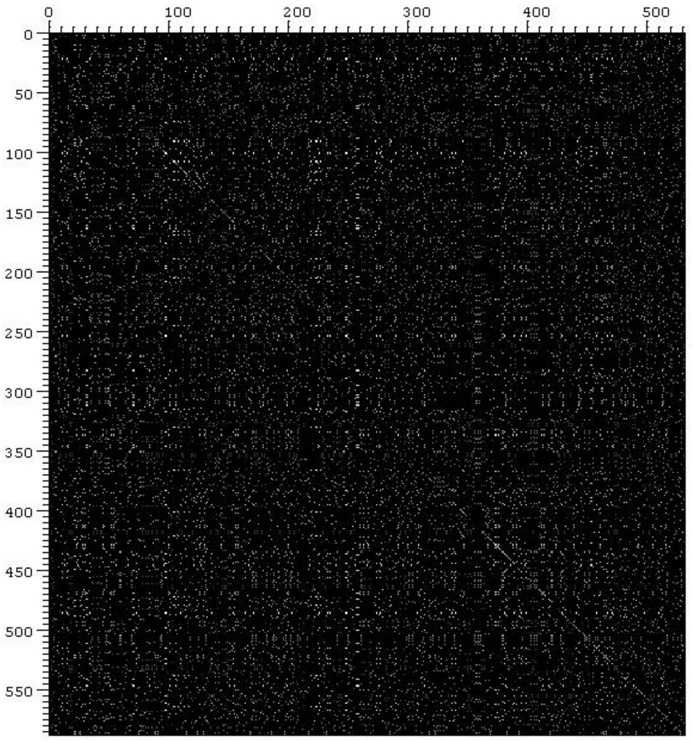

k=4

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

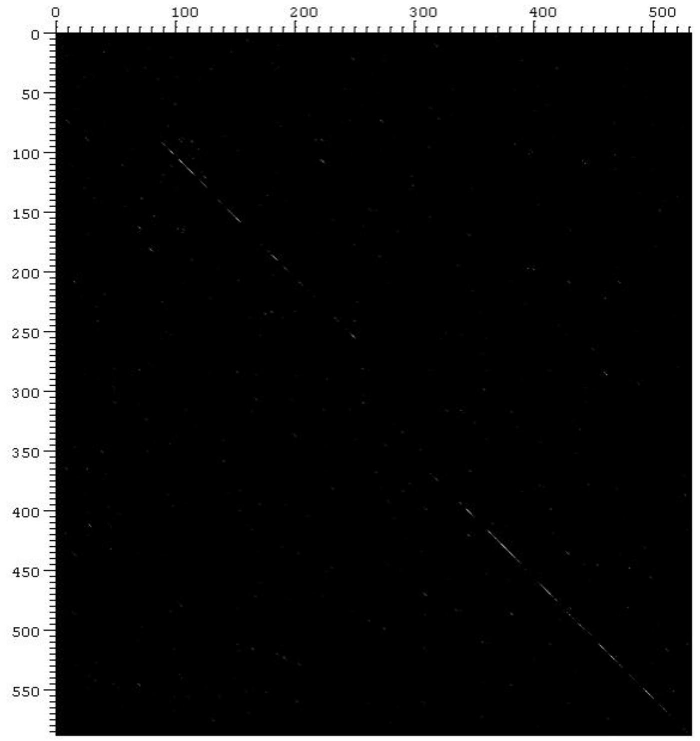

k=8


k=16

k=1

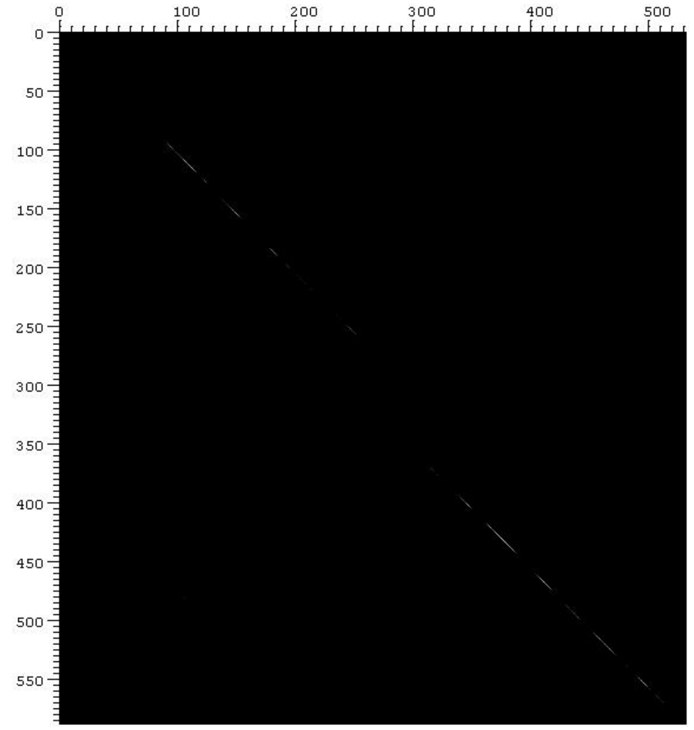


k=4

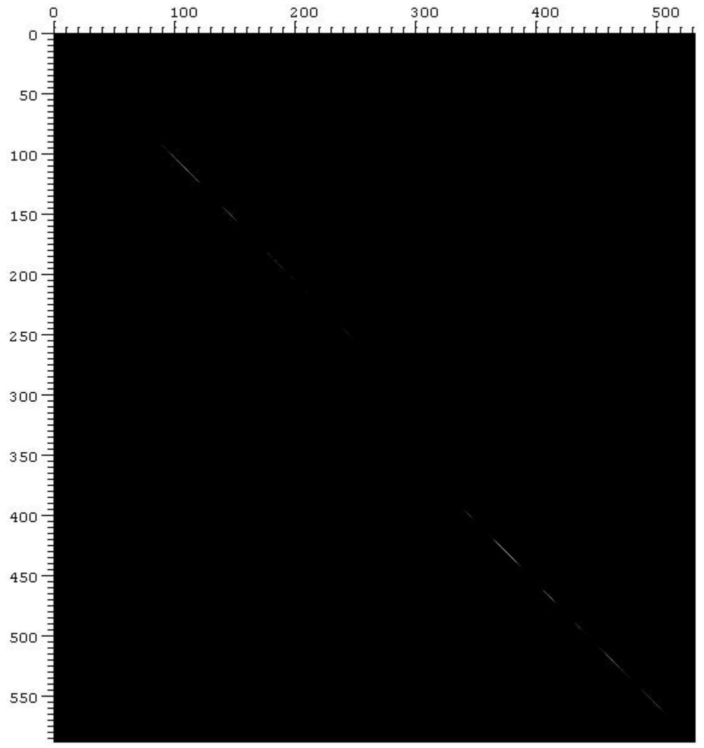


Dot matrix
(k=1,4,8,16) for two
protein sequences
CAB51201.1 (531 aa)
CAA72681.1 (588 aa)

k=8



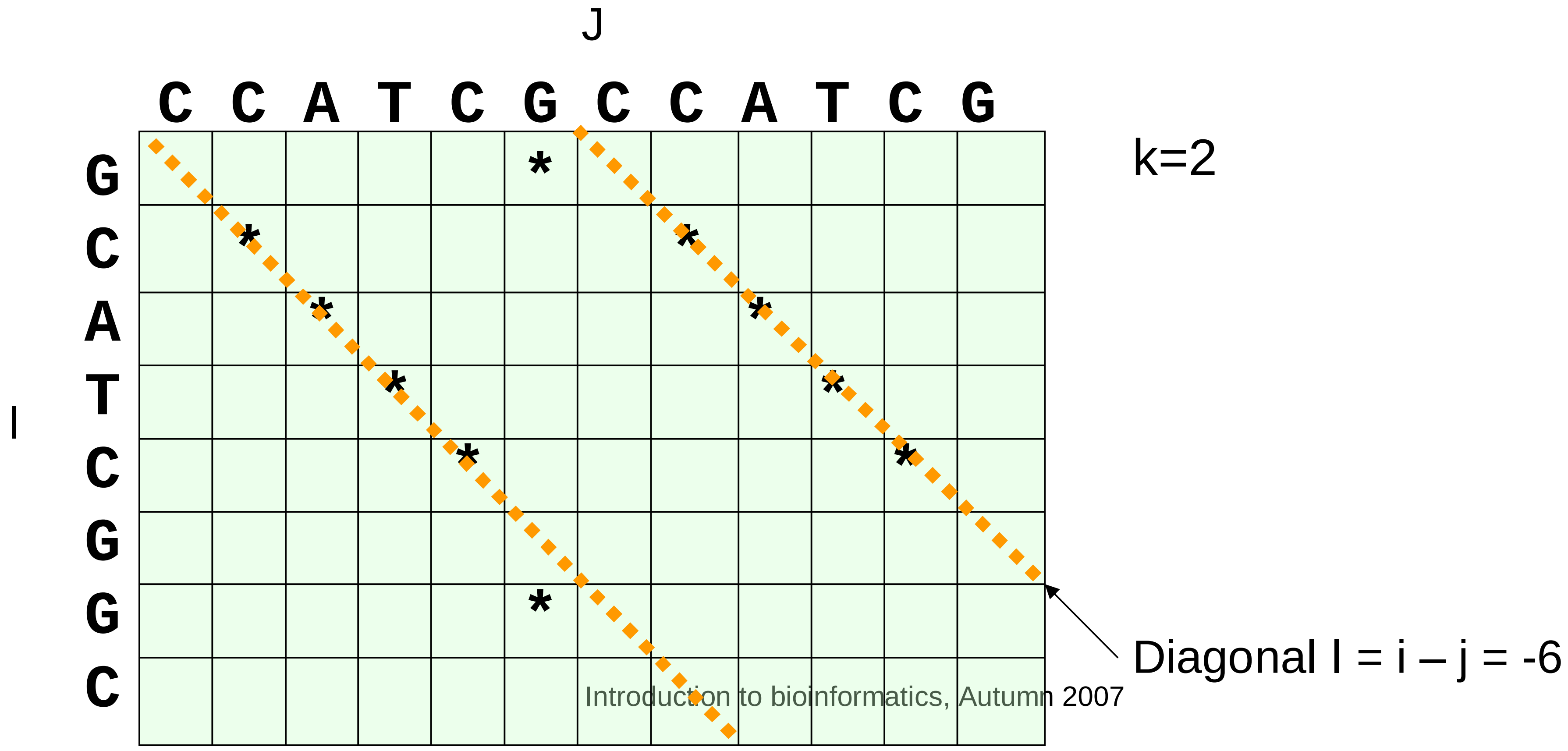
k=16



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



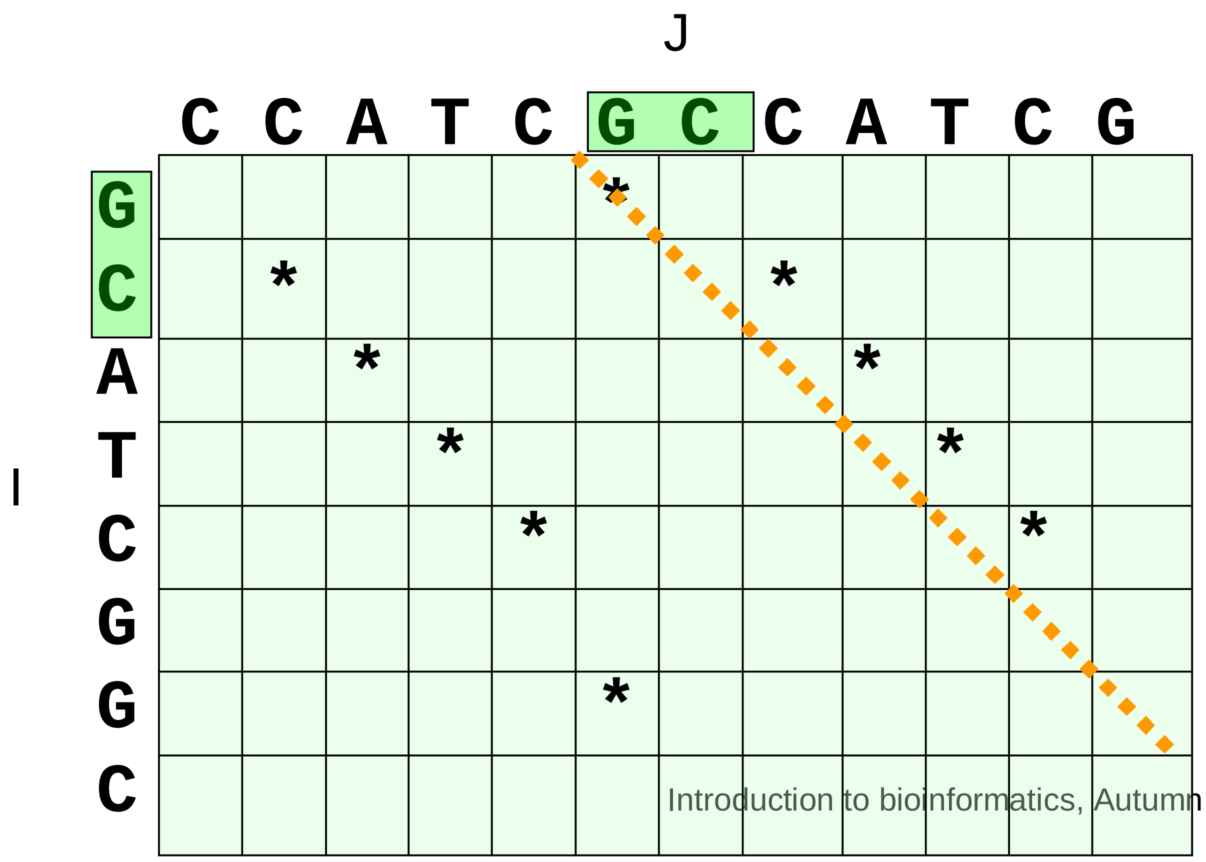
Computing diagonal sums

- 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

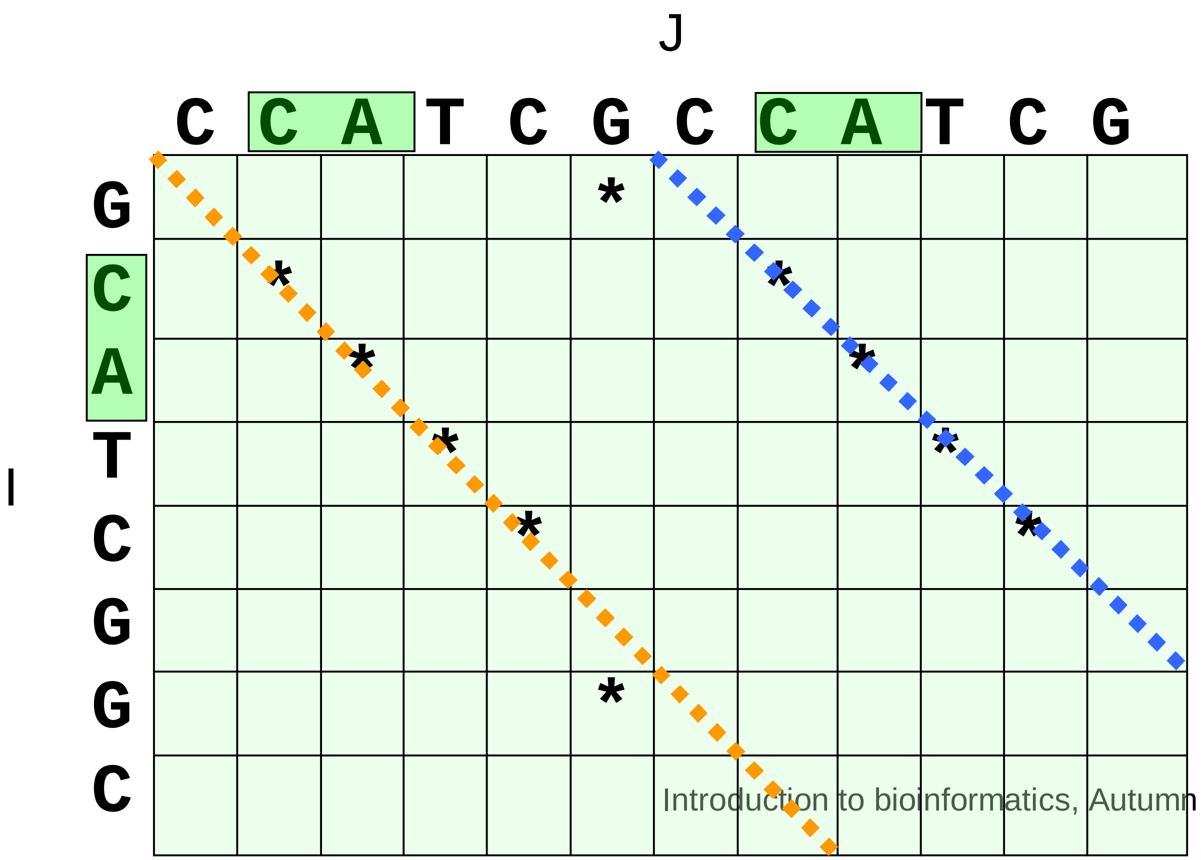


For the first 2-word in I, GC, $L_{GC}(J) = \{6\}$.

We can then update the sum of diagonal $l = i - j = 1 - 6 = -5$ 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:

$l = i - j = 2 - 2 = 0$

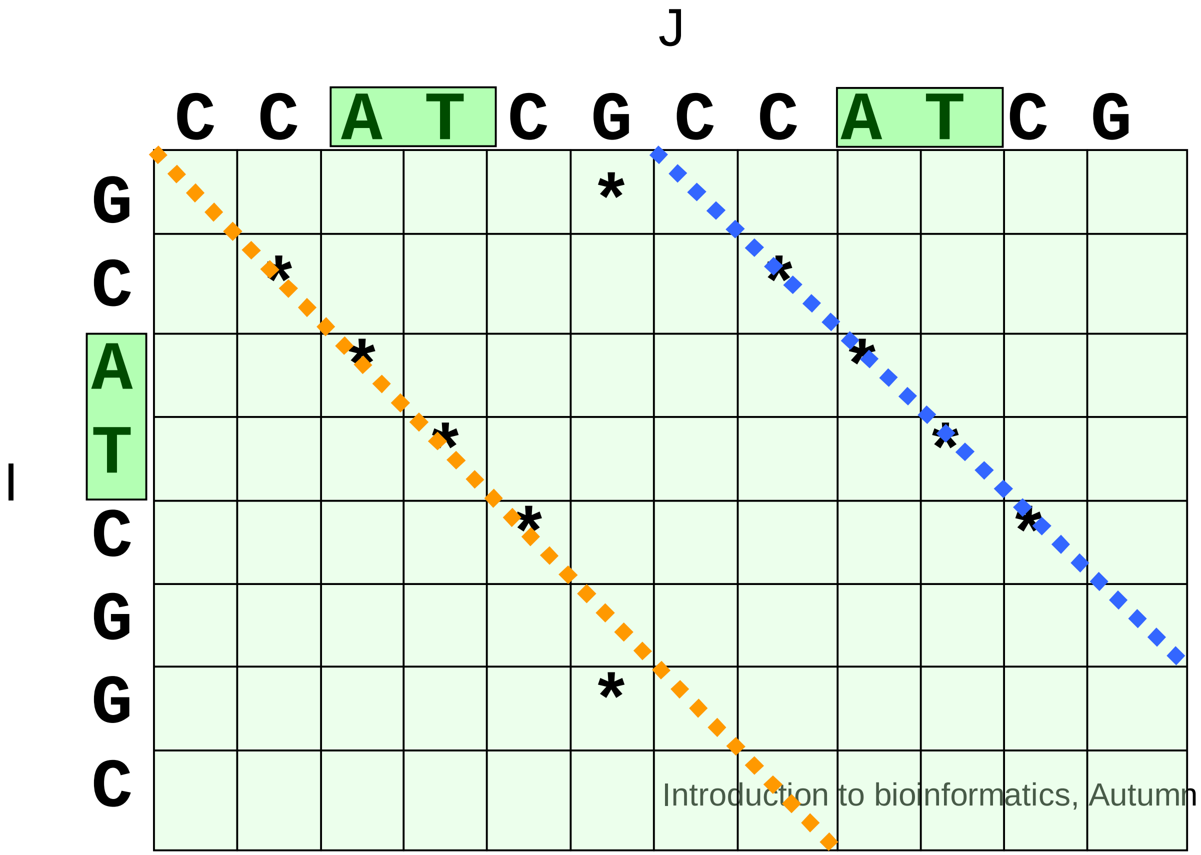
$S_0 := S_0 + 1 = 0 + 1 = 1$

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

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

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:

$l = i - j = 3 - 3 = 0$

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

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

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

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

J

I

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

Algorithm for computing diagonal sum of scores

```
$S_l$ := 0 for all $1 – m \le l \le n – 1$
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
```

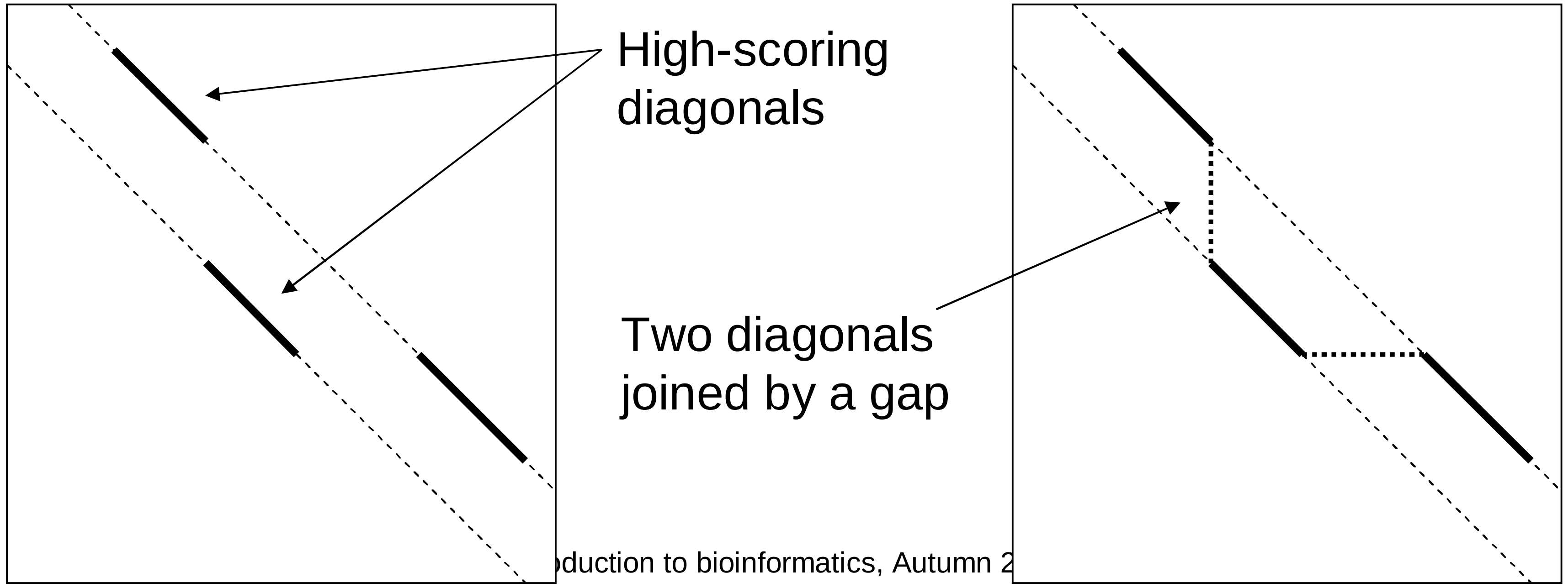
75% identity, no 4-word identities

```
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

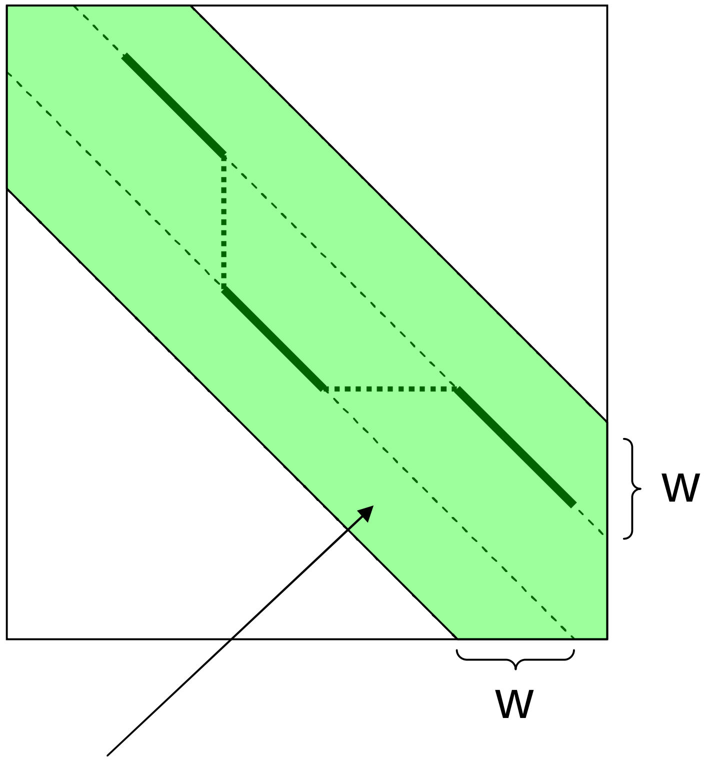


FASTA outline

- 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
- FASTA can be very specific when identifying long regions of low similarity
  - Specific method does not produce many incorrect results
  - Sensitive method produces 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