BMEG3102 Bioinformatics

Lecture 3. Sequence Alignment and Searching (2/2)

Qi Dou

Email: qidou@cuhk.edu.hk

Office: Room 1014, 10/F, SHB

BMEG3102 Bioinformatics

The Chinese University of Hong Kong

Lecture outline

1. **Computational complexity of optimal alignment problems**

2. **Heuristic methods**
   - Dot plot
   - Pairwise sequence alignment
     - FASTA
       - The FASTA file format
     - BLAST
       - Statistical significance
     - Variations
   - Multiple sequence alignment

Part 1

Computational Complexity of Optimal Alignment Problems

Computational complexity

- To align two sequences of lengths m and n by dynamic programming, how much time and space do we need?
  - $O(mn)$
  - Can do better, but still expensive

- To find the sequence in a database with ℓ length-n sequences that is closest to a query sequence of length m, how much time and space do we need?
  - Suppose we consider the database sequences one by one
  - $O(\ell mn)$ time
  - $O(mn)$ space

Numbers in real situations

- Scenario 1: Whole-genome alignment between two species
  - *m*, *n* (e.g., *C. elegans* and *C. briggsae*): ~ 100 x 10^6
  - $mn \approx 10^{16}$
  - If a computer can do 3 x 10^9 operations in a second, it would take ~3,000,000 seconds = 926 hours = 38.6 days
  - Still ok if you can wait. But can we find a machine with 10^{16}, i.e., 10PB RAM?

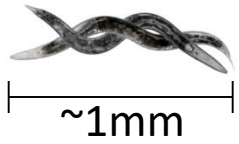


Image credit: wormbase.org

Numbers in real situations

- Scenario 2: Searching for a gene from a database
  - m (e.g., A human gene): ~3,000 on average
  - ℓ (e.g., GenBankv229): 211,281,415 sequences
  - n (e.g., GenBank v229): 285,688,542,186 bases / 211,281,415 sequences = ~1,350
  - mn = ~4,000,000 (manageable)
  - ℓmn = ~857 x 10^{12}
  - If a computer can do 3 x 10^9 operations in a second, it would take ~286,000 seconds = ~80 hours = ~3.3 days
  - If you go to GenBank, it will take only a minute
    - Don't forget it serves many users at the same time

Heuristics

- How to perform alignment faster and with less memory?

1. Quickly identify regions with high similarity
   - By inspection
   - By considering short sub-sequences

2. Combine and refine these initial results
   - The results may not be optimal in terms of alignment score, but the process is usually much faster than dynamic programming

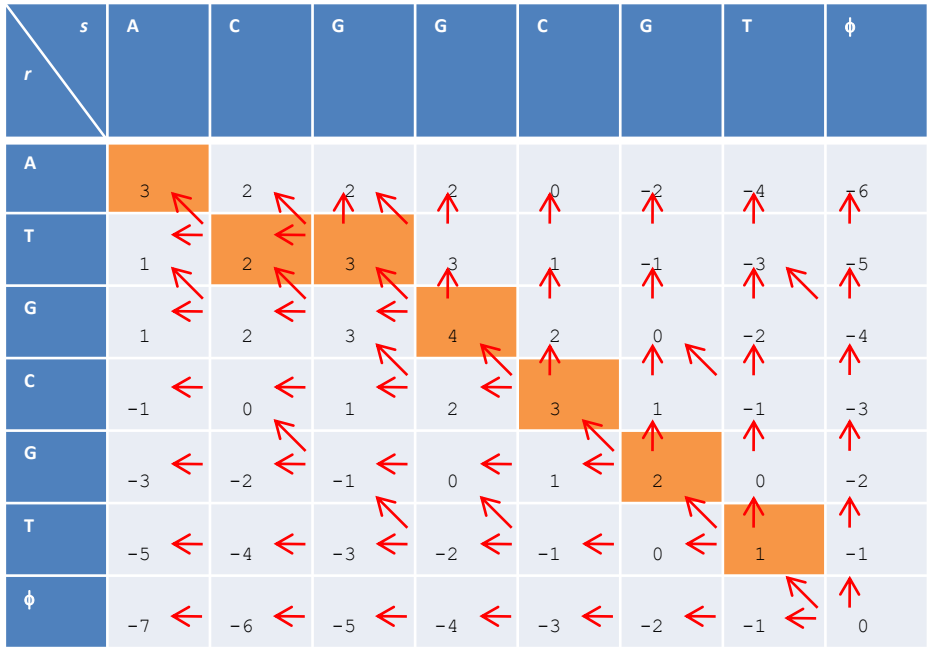
– **These methods are called heuristic methods**

Part 2

Heuristic Methods

Dot plot

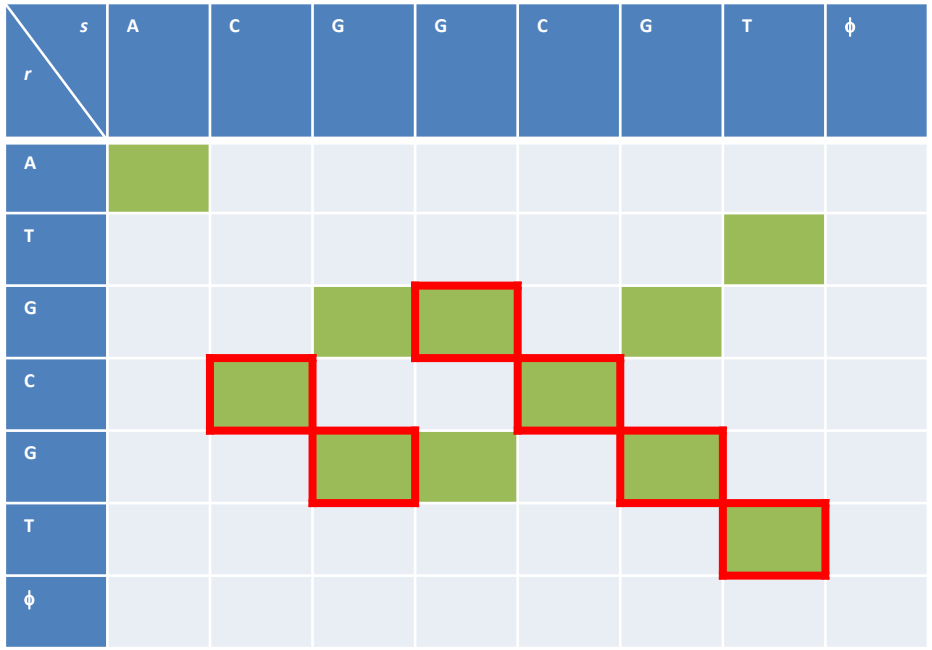
- Consider an alignment that we studied before:



- If we remove the details but only light up the matched characters, what are we going to see?

Diagonals

- We will see “diagonals”



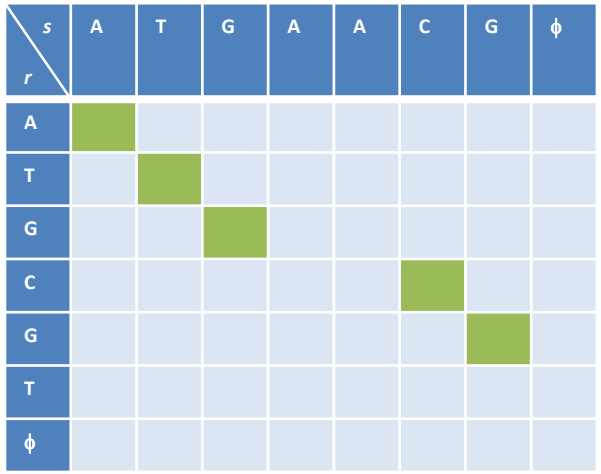
- Each diagonal marks a local exact match

Dot plot

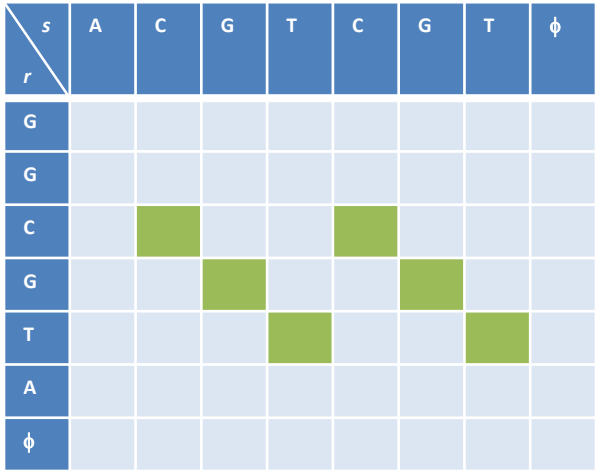
- In general, the dot plot gives us information about:
  - Conserved regions
  - Non-conserved regions
  - Inversions
  - Insertions and deletions
  - Local repeats
  - Multiple matches
  - Translocations

Alignment types

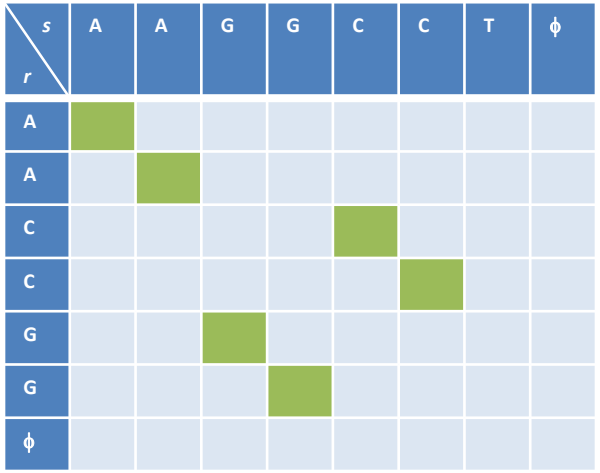
- Information about the aligned sequences based on a dot plot



Insertion/deletion



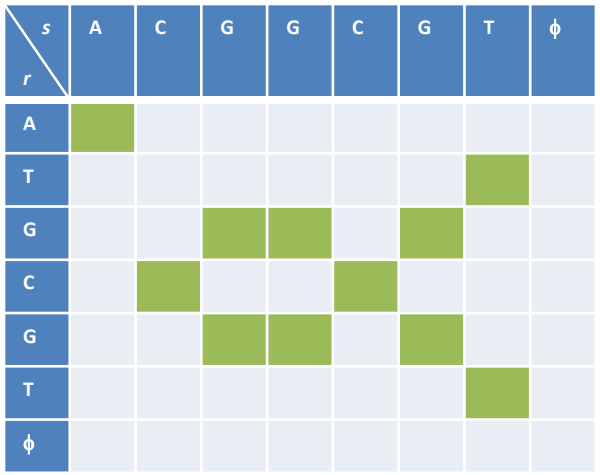
Duplication



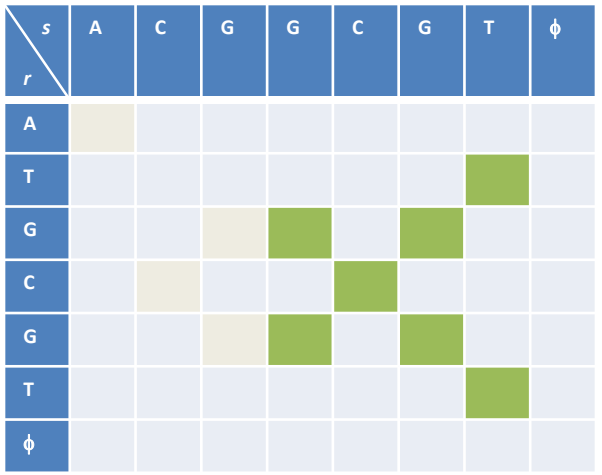
translocation

Resolution

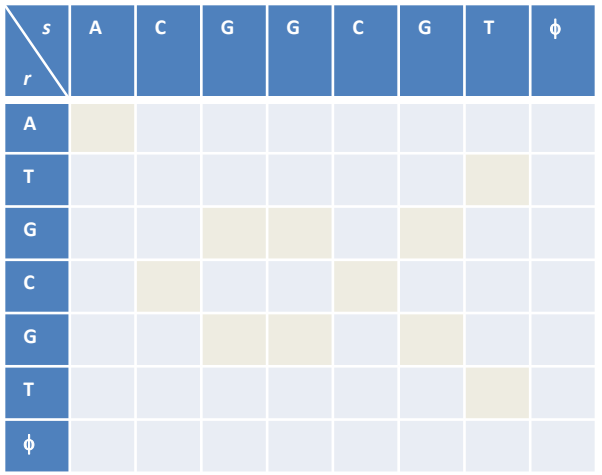
- What we will see if we only highlight diagonal runs of length at least 1, 3, and 5:



Resolution: 1 character



Resolution: 3 characters



Resolution: 5 characters

- Which one is the best?

Limitations

- Must be exact matches
  - –Possible to allow some mismatches, but more computations would be needed
    - We will come back to this topic when we study BLAST
- The whole plot takes a lot of space
- Difficult to determine resolution
  - – Show even one base match: Too many dots
  - – Show only long matches: May miss important signals
- Mainly for visualization, not quantitative

We now study how the ideas of dot plot can help us perform database search

Database search

- Problem: Given a query sequence *r* and a database *D* of sequences, find sequences in *D* that are similar to *r*
  - For each identified sequence *s*, good to return a match score, sim(*r*, *s*)
  - Good to list multiple sequences with high match scores instead of the best one only
  - $|D|$ is usually very large (dynamic programming is not quite feasible)

Heuristic database search

- Two main steps, based on dot-plot ideas:

  1. Instead of aligning the whole *r* with a whole sequence in *D*, we look for short sub-sequences of very high similarity using very quick methods

  2. Combine and extend these initial results to get longer matches

- Notes:
  - We will start with one sequence *s* in *D*
    - To search for high-similarity matches from the whole database, one may simply repeat the two steps for every sequence in the database
    - There are ways to make it faster, by using index structures
  - The two steps do not guarantee to produce the best matches
  - We will cover the high-level ideas

Popular methods

- We now study two popular methods
  - FASTA
  - BLAST

- Both use the mentioned ideas for performing local alignments, but in different ways

FASTA

- First version for protein sequences (FASTP) proposed by David J. Lipman and William R. Pearson in 1985 (Lipman and Pearson, Rapid and Sensitive Protein Similarity Searches, *Science* 227(4693):1435-1441, 1985)

- Pronounced as “Fast-A” (i.e., **fast** for **a**ll kinds of sequences)

Overview of FASTA

- Step 1:
  a) Find matches with stretches of at least *k* consecutive exact matches. Find the best (e.g., 10) matches using a simple scoring method.
  b) Refine and re-evaluate the best matches using formal substitution matrices.
- Step 2:
  c) Combine the best matches with gaps allowed.
  d) Use dynamic programming (DP) on the combined matches. Banded DP: Considering only a band in the DP table

Let's study more details about a and c in the coming slides

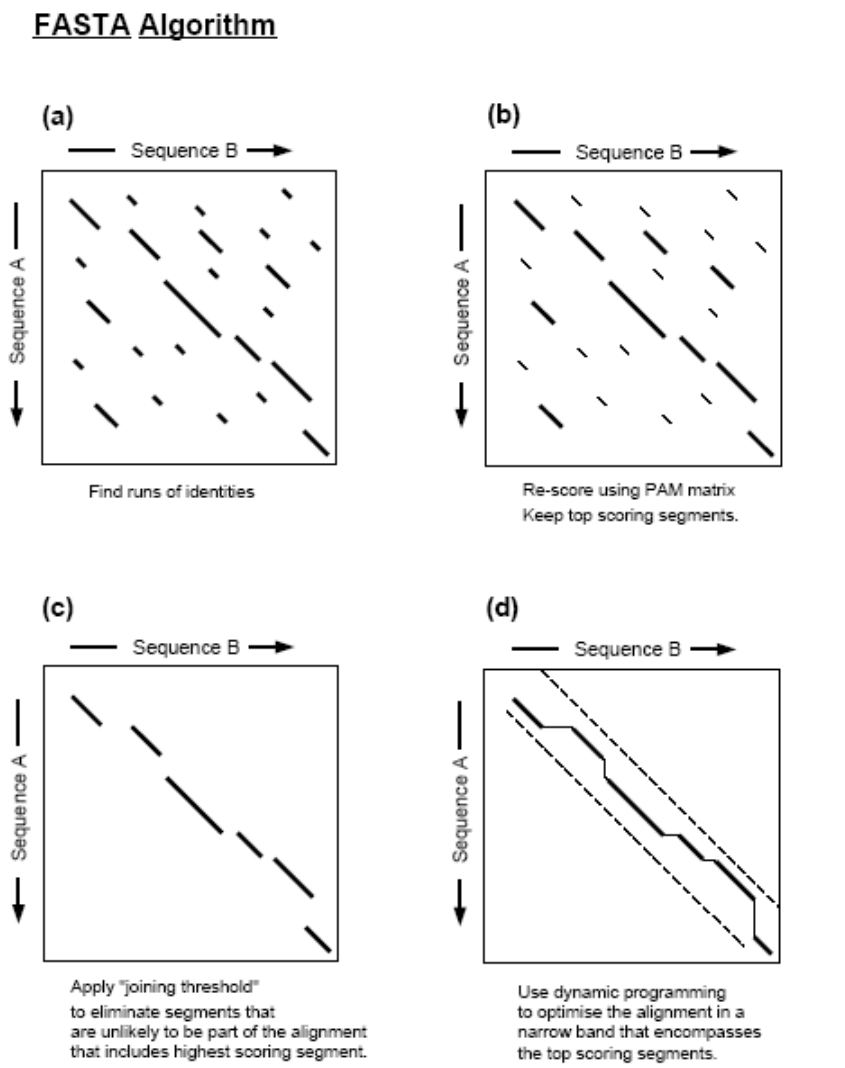


Image credit: Wikipedia

Overview of FASTA

- Step 1:
  a) Find matches with stretches of at least *k* consecutive exact matches. Find the best (e.g., 10) matches using a simple scoring method.
  b) Refine and re-evaluate the best matches using formal substitution matrices.
- Step 2:
  c) Combine the best matches with gaps allowed.
  d) Use dynamic programming (DP) on the combined matches. Banded DP: Considering only a band in the DP table

Let's study more details about a and c in the coming slides

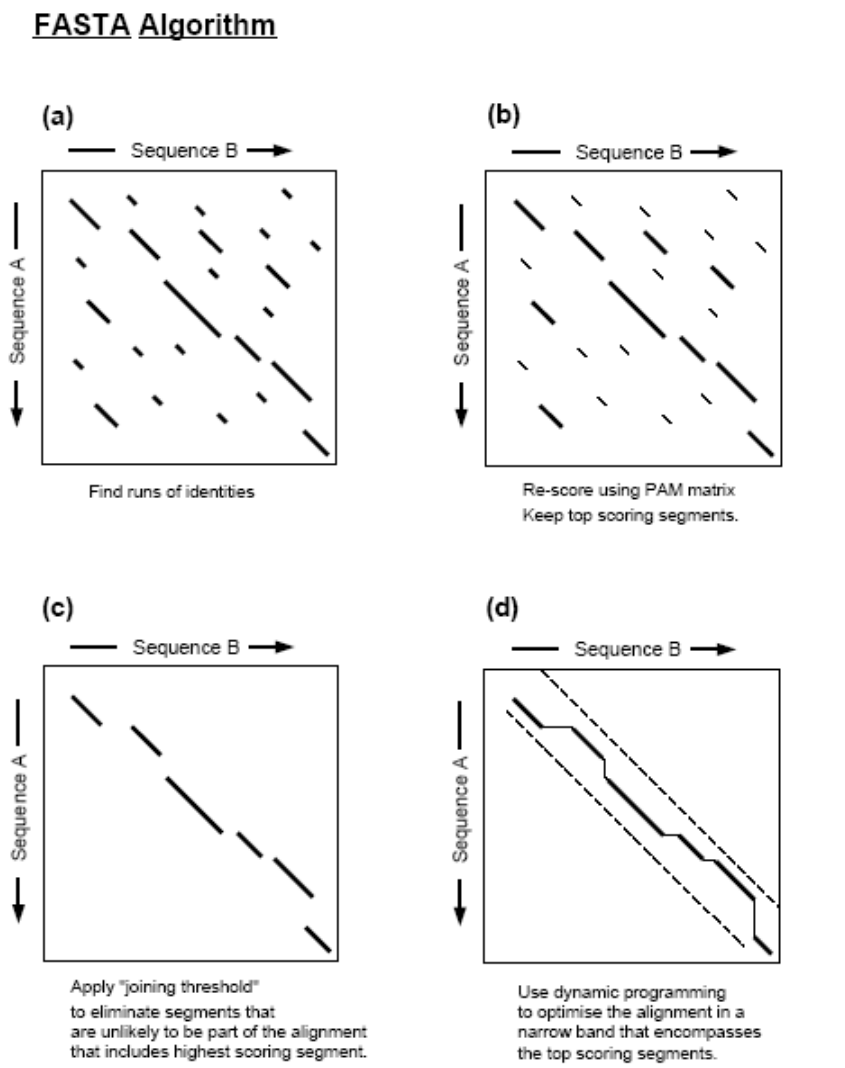


Image credit: Wikipedia

Finding local exact matches

- Finding diagonals of fixed length *k*
  - Usually
    - *k*=1-2 for protein sequences
    - *k*=4-6 for DNA sequences
- Key: Building a lookup table
- Example (new):
  - Sequence *r*: `ACGTTGCT`
  - Sequence *s* in database *D*:

```
0          1
123456789012
GCGTGACTTTCT
```

  - Let's use *k*=2 here

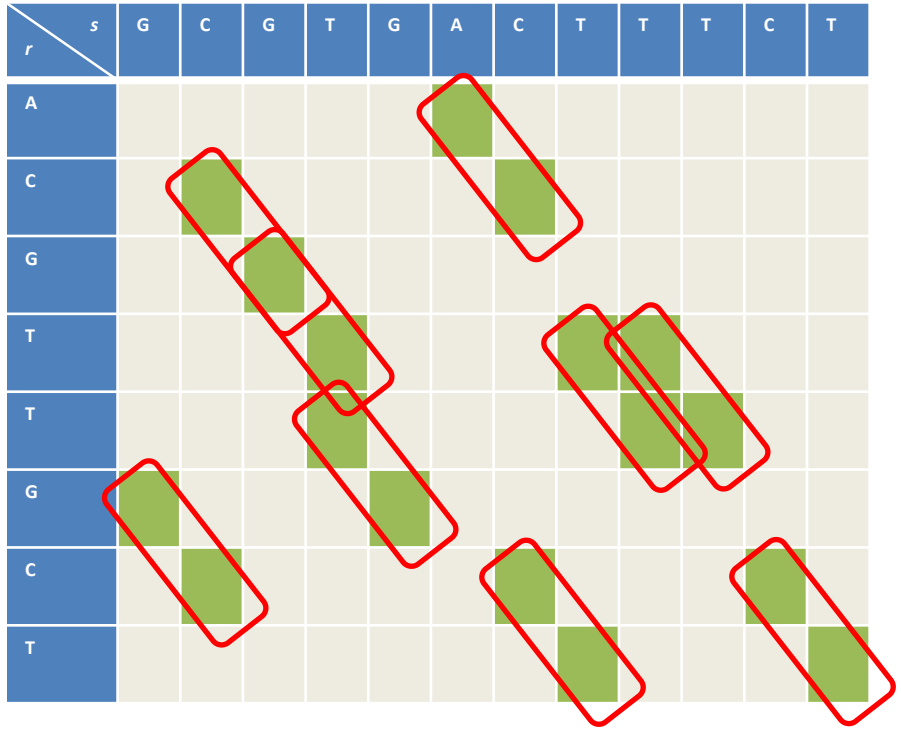
There are different types of lookup tables that can be used. Here we use one that includes every length-2 subsequence (the "2-mers") of *s* sorted lexicographically.

Length-2 subsequences of *s*	Positions
AC	6
CG	2
CT	7, 11
GA	5
GC	1
GT	3
TC	10
TG	4
TT	8, 9

Finding local exact matches

- Sequence *r*: ACGTTGCT
- Relevant sub-sequences:

Length-2 subsequences of *s*	Positions
AC	6
CG	2
CT	7, 11
GA	5
GC	1
GT	3
TC	10
TG	4
TT	8, 9

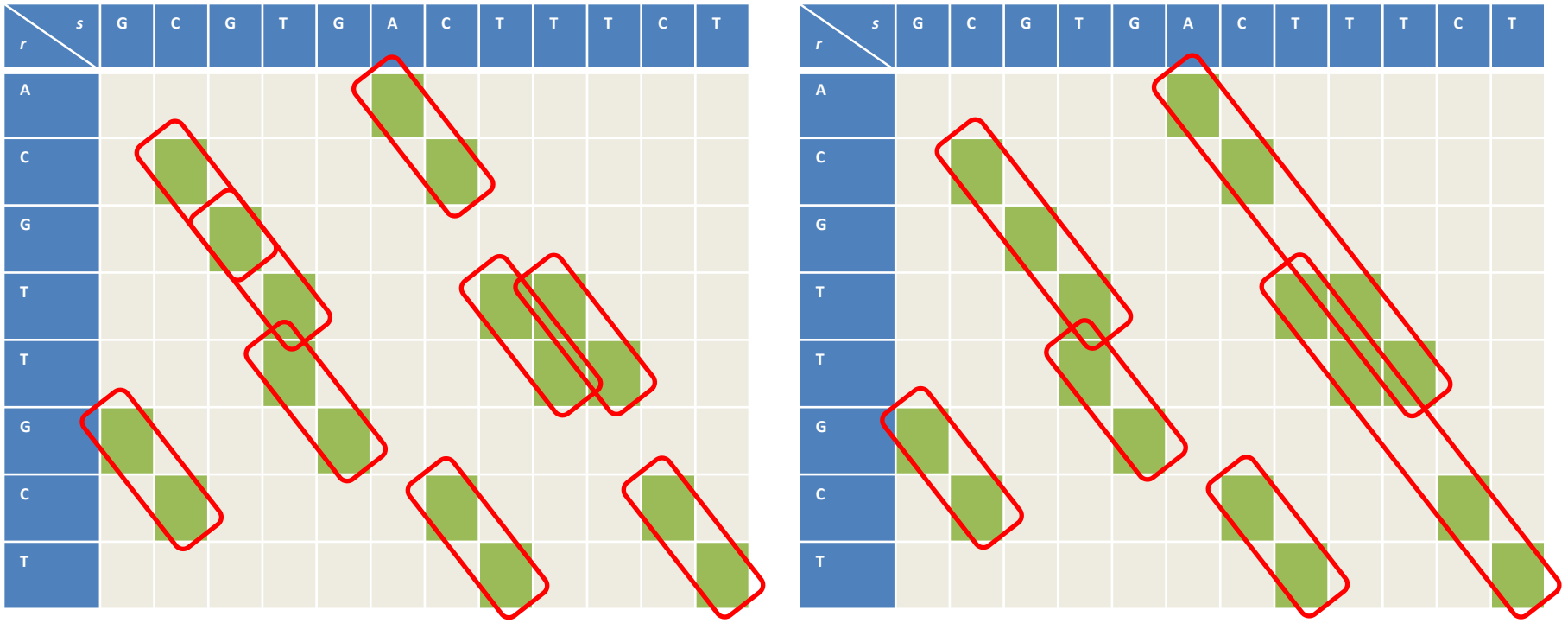


Note: The dot plot is for illustration only. The FASTA program does not need to construct it.

- Quick recap: Why is it not a good idea to construct the dot plot?
- How can the long matches be found without it?

Merging matches

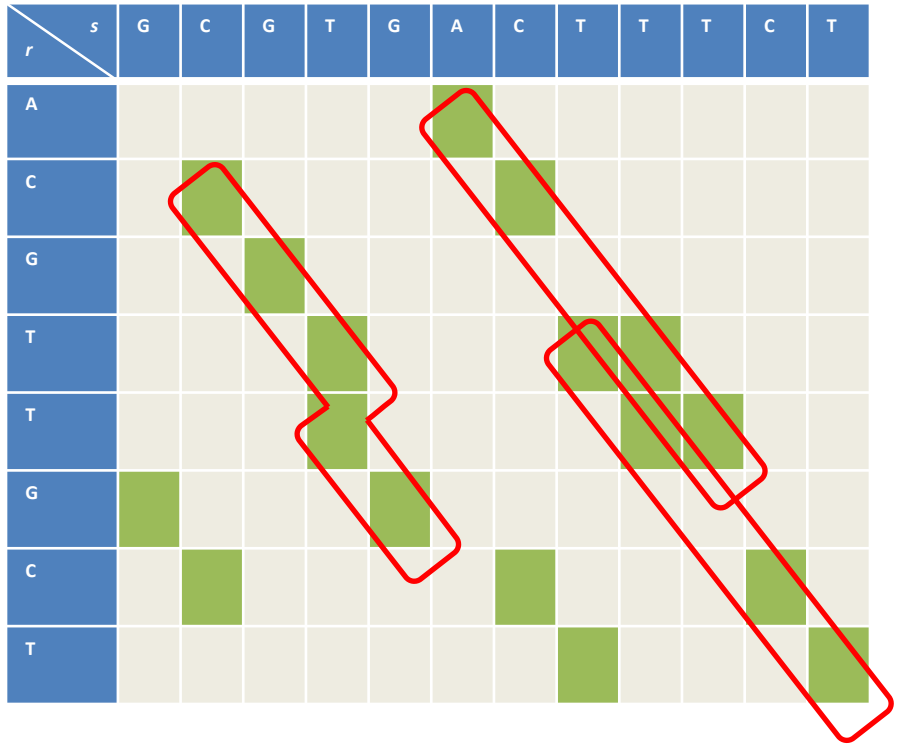
- Merge matches on same diagonal (e.g., $r[2,3]=s[2,3]$ and $r[3,4]=s[3,4]$ imply $r[2,4]=s[2,4]$)
  - More advanced methods also allow gaps



Remaining steps

- Keep some (e.g., 10) high-scoring matches
- Merge matches in different diagonals by allowing indels
- Perform local alignment by dynamic programming

Possible final results:



Missing the optimal alignment

- When will FASTA miss an optimal alignment?

  –Good but not exact local matches (especially for protein sequences) ⇒ Not included in the very first step
    - *k* too large
    - High-scored mismatches, especially for protein sequences

  –Too many local candidates ⇒ The algorithm keeps only a few “best” ones, but it happens that they are not involved in the optimal alignment

Time and space requirements

- How much space is needed?
  - One entry per length-*k* sub-sequence. (*n*-*k*+1) of them for a sequence of length *n*
- How much time is needed?
  - One table lookup per length-*k* sub-sequence
  - In the worst case, it can still take O(*mn*) time
    - Consider matching `AAAAA` with `AAAAAAA`
  - In practice, usually it is much faster
    - There are other types of lookup table that allows finding correct table entries efficiently
- When there are multiple sequences in the database, the lookup tables for different sequences can be combined. Need to record the original sequence of each subsequence in that case.

Indexing multiple sequences

- Suppose we have the following two sequences s_1 and s_2 in the database D:
  - s_1:

```
 0          1
 123456789012
 GCGTGACTTTCT
```

  - s_2:

```
 0         1
 1234567890
 CTGGAGCTAC
```

Lookup table:

Length-2 subsequences	Sequences and positions
AC	s_1:6, s_2:9
AG	s_2:5
CG	s_1:2
CT	s_1:7, s_1:11, s_2:1, s_2:7
GA	s_1:5, s_2:4
GC	s_1:1, s_2:6
GG	s_2:3
GT	s_1:3
TA	s_2:8
TC	s_1:10
TG	s_1:4, s_2:2
TT	s_1:8, s_1: 9

Overview of FASTA

- Step 1:
  a) Find matches with stretches of at least *k* consecutive exact matches. Find the best (e.g., 10) matches using a simple scoring method.
  b) Refine and re-evaluate the best matches using formal substitution matrices.
- Step 2:
  c) Combine the best matches with gaps allowed.
  d) Use dynamic programming (DP) on the combined matches. Banded DP: Considering only a band in the DP table

Let's study more details about a and c in the coming slides

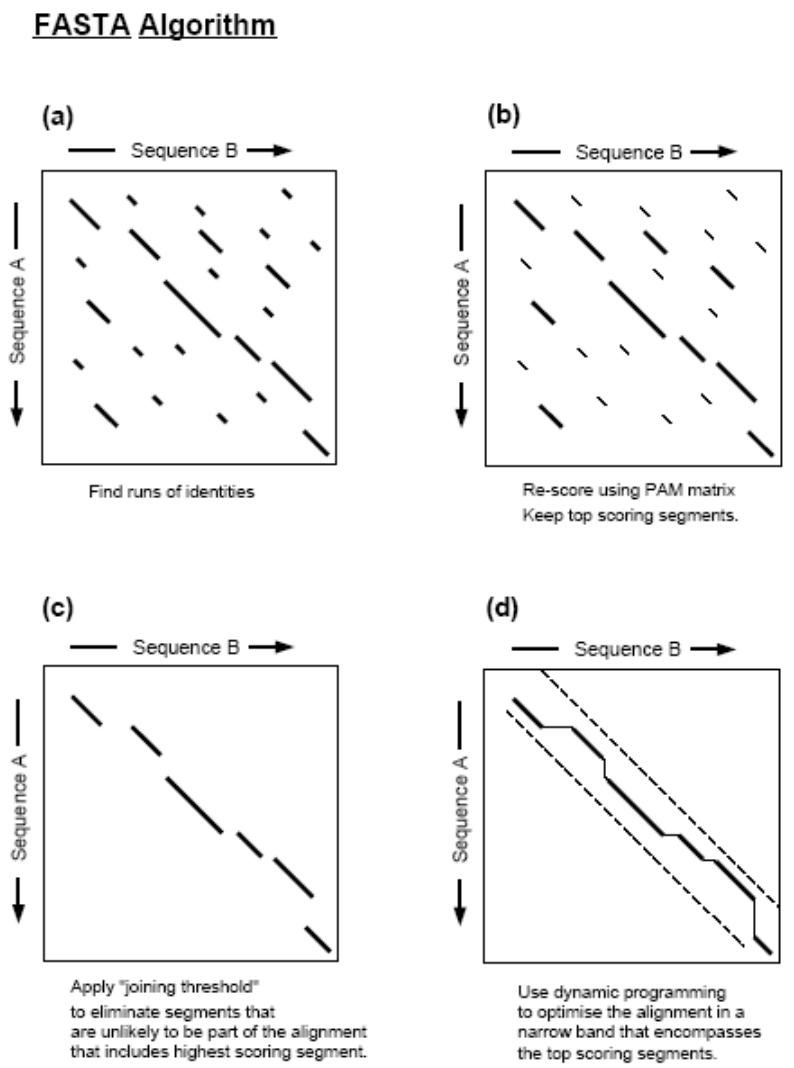


Image credit: Wikipedia

The FASTA file format

- FASTA may not be the most frequently used heuristic sequence alignment method, but the FASTA file format is probably the most frequently used format for sequence data
- The format (see http://en.wikipedia.org/wiki/FASTA_format):
  - Text-based
  - Can store multiple sequences, one after another
  - For each sequence:
    - One line that starts with '>', stating the metadata (e.g., ID) of the sequence
    - One or more lines for the actual sequence. Usually, each line contains no more than 80 characters to fit screen width
  - Can add comment lines that start with ';'

Example

```
>SEQUENCE_1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDFDKAVQLLREKGLGKAAKKADRLAAEG
LVSVKVSDDFTIAAMRPSYLSYEDLDMTFVENEYKALVAELEKENEERRRLKDPNKPEHK
IPQFASRKQLSDAILKEAEEKIKEELKAQGKPEKIWDNIIPGKMNSFIADNSQLDSKLTL
MGQFYVMDDKKTVEQVIAEKEKEFGGKIKIVEFICFEVGEGLEKKTEDFAAEVAAQL
>SEQUENCE_2
SATVSEINSETDFVAKNDQFIALTKDTTAHIQSNSLQSVEELHSSTINGVKFEEYLKSQI
ATIGENLVVRRFATLKAGANGVVNGYIHTNGRVGVVIAAACDSAEVASKSRDLLRQICMH
```

Image credit: Wikipedia

BLAST

- **B**asic **L**ocal **A**lignment **S**earch **T**ool
- Proposed by Altschul et al. in 1990

  (Altschul et al., *J. Mol. Biol.* 215(3):403-410, 1990)
- Probably the most frequently used (and most well-known) algorithm in bioinformatics

BLAST vs. FASTA

- BLAST also uses the two main ideas (finding local matches, then extending and combining them)
- Main differences between the original ideas of BLAST and FASTA:
  1. FASTA considers exact matches in the first step. BLAST allows high-scoring inexact matches
  2. BLAST tries to extend local matches regardless of the presence of local matches in the same diagonal
  3. BLAST contains a way to evaluate statistical significance of matched sequences
- In later versions the two share more common ideas
- Let’s study these differences in more details

1. Local matches

- Again, consider the query sequence *r*: ACGTTGCT
- Suppose *k*=3, the first sub-sequence ("word") is ACG
- FASTA looks for the locations of ACG in the sequences in the database
- BLAST looks for the locations of ACG and other similar length-3 sequences
  - If match has +1 score, mismatch has -1 score, and we only consider sub-sequences with score ≥ 1, we will consider these sub-sequences:

ACG
CCG
GCG
TCG
AAG
AGG
ATG
ACA
ACC
ACT

Faster or slower?

- For the same word length, BLAST needs to search for more related sub-sequences

- However, BLAST is usually faster than FASTA because
  - BLAST uses a larger *k*, and so there are fewer matches (for DNA, usually BLAST uses 11 while FASTA uses 6-8)
  - Couldn't FASTA also use a large *k*? No, because it only considers exact matches. Many local matches would be missed if *k* is too large

2. Extending and combining matches

- For each local match, BLAST extends it by including the adjacent characters in the two ends until the match score drops below a threshold

- Second version of BLAST (BLAST2) also tries to combine matches on the same diagonal

3. Statistical significance

- Besides being faster, another main contribution of BLAST is evaluating the statistical significance of search results
- Statistical significance: the “E-value”

3. Statistical significance

- Besides being faster, another main contribution of BLAST is evaluating the statistical significance of search results
- Statistical significance: the “E-value”
  - Suppose the query sequence *r* has length *m*, a sequence *s* in the database has length *n*, and a match has score *Q*. What is the expected (i.e., mean) number of matches with score *Q* or larger for a pair of random sequences of lengths *m* and *n* respectively?
  - What is the expected number in the whole database?
    - This expected number in the whole database is called the E-value

3. Statistical significance

- Besides being faster, another main contribution of BLAST is evaluating the statistical significance of search results
- Statistical significance: the “E-value”
  - –Suppose the query sequence *r* has length *m*, a sequence *s* in the database has length *n*, and a match has score *Q*. What is the expected (i.e., mean) number of matches with score *Q* or larger for a pair of random sequences of lengths *m* and *n* respectively?
  - –What is the expected number in the whole database?
    - This expected number in the whole database is called the E-value
  - –A small E-value means it is unlikely to happen by chance, thus suggesting potential biological meaning
    - Logic: There must be a reason behind this high similarity. For example, *r* and *s* may be evolutionarily or functional related.

Statistical significance

- An illustration:
  - *r*=A
  - *s*=AC
  - Best match: exact match, match score = 1

Statistical significance

- An illustration:
  - *r*=A
  - *s*=AC
  - Best match: exact match, match score = 1
- How many matches are there with score ≥ 1 for random *r* and *s* of lengths 1 and 2, respectively?
  - 0 matches:
    - *r*=A, *s*=CC, CG, CT, GC, GG, GT, TC, TG, TT (9 cases)
  - 1 match:
    - *r*=A, *s*=AC, AG, AT, CA, GA, TA (6 cases)
  - 2 matches:
    - *r*=A, *s*=AA (1 case)

Statistical significance

- An illustration:
  - *r*=A
  - *s*=AC
  - Best match: exact match, match score = 1
- How many matches are there with score ≥ 1 for random *r* and *s* of lengths 1 and 2, respectively?
  - 0 matches:
    - *r*=A, *s*=CC, CG, CT, GC, GG, GT, TC, TG, TT (9 cases)
  - 1 match:
    - *r*=A, *s*=AC, AG, AT, CA, GA, TA (6 cases)
  - 2 matches:
    - *r*=A, *s*=AA (1 case)
  - Expected number assuming equal chance of all cases (due to symmetry, no need to consider *r*=C, *r*=G and *r*=T):
    - (0x9 + 1x6 + 2x1) / 16 = 0.5 – statistically not quite significant (usually call it significant if <0.05 or <0.01)
    - In reality, need to estimate chance of each case from some large databases instead of assuming uniform distribution

Computing statistical significance [optional]

- For large *m* and *n*, we cannot list all cases to find the expected number

- Fortunately, the match score of two sub-sequences is the maximum score among all possible local alignments
  - When *m* and *n* are large, the match score tends to follow an *extreme value distribution*
  - There are known formulas to compute E-values

- For a match between *r* and *s* from database *D*, the size of *D* (and the length of its sequences) should be included in the calculation
  - See http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html for some more details

Variations

- Depending on the type of sequences (query-database):
  - –Nucleotide-nucleotide BLAST (blastn)
  - –Protein-protein BLAST (blastp)
  - –Nucleotide 6-frame translation-protein BLAST (blastx)
    - Perform 6-frame translation of nucleotide query, then compare with protein sequences in database
  - –Protein-nucleotide 6-frame translation BLAST (tblastn)
    - Compare query protein sequence with 6-frame translation of nucleotide sequences in database
  - –Nucleotide 6-frame translation-nucleotide 6-frame translation BLAST (tblastx)
    - Perform 6-frame translation of query and database nucleotide sequences, then perform comparisons

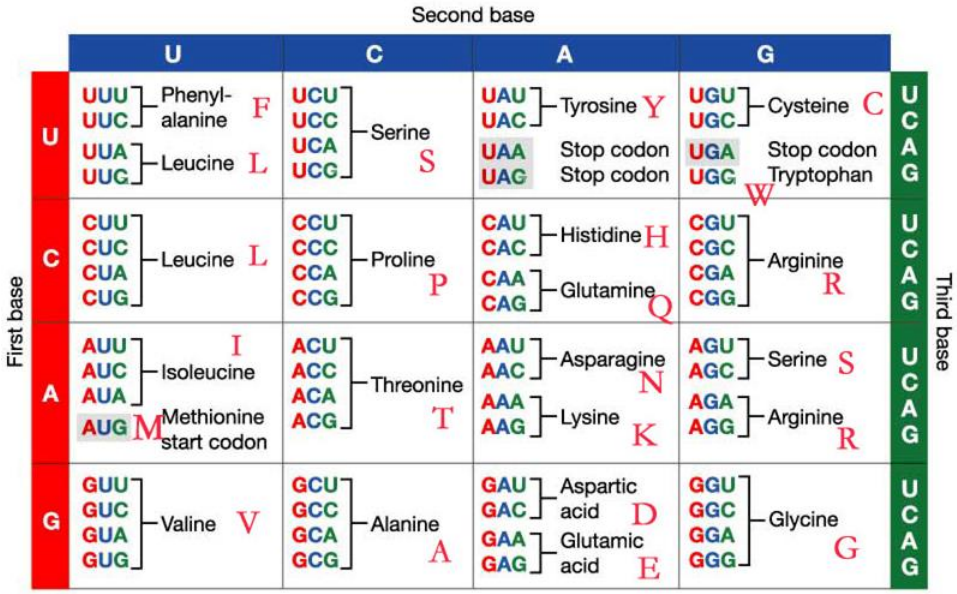
Variations

Tool	Query sequence	Database sequences	Comparison
blast**n**	**N**ucleotide	**N**ucleotide	Nucleotide-nucleotide
blast**p**	**P**rotein	**P**rotein	Protein-protein
blast**x**	**N**ucleotide	Protein	6FT-protein
tblastn	Protein	**N**ucleotide	Protein-6FT
tblast**x**	**N**ucleotide	**N**ucleotide	6FT-6FT

Six-frame translation revisited

Reading frame

```
+3       L   V   R   T
+2     T   C   S   Y
+1   N   L   F   V
   5′-AACTTGTTCGTACA-3′
   3′-TTGAACAAGCATGT-5′
-1       K   N   T   C
-2     S   T   R   V
-3       V   Q   E   Y
```



Variations

Tool	Query sequence	Database sequences	Comparison
blastn	Nucleotide	Nucleotide	Nucleotide-nucleotide
blastp	Protein	Protein	Protein-protein
blastx	Nucleotide	Protein	6FT-protein
tblastn	Protein	Nucleotide	Protein-6FT
tblastx	Nucleotide	Nucleotide	6FT-6FT

- When do you want to use blastn and when to use tblastx?
  - –blastn: If conservation is expected at nucleotide level (e.g., ribosomal RNA)
  - –tblastx: If conservation is expected at the protein level (e.g., coding exons)

Iterative database search

- Suppose you have a sequence and would want to find a group of similar sequences in a database

- However, you are not sure whether your sequence has all the key properties of the group

- You can do an iterative database search

Illustration

- Suppose there is a group of related sequences with two properties:
  1. Mainly C's in the first half
  2. Mainly T's in the second half
- You have one of the sequences

  r: CCCCTATG

  –It has perfect signature for #1, but not very clear for #2
- Suppose a database contains the following sequences from the same group:

  s_1: CCCCTTTT

  s_2: CCGCATTT

  s_3: GCTCTTTT

  s_4: AACCTTTT

  –If we use *r* to query the database, probably we can only get the first one or two

How to get all four?

1. First, use BLAST to get the highly similar sequences (let's say you get s_1 and s_2)
2. Then, construct a profile of these sequences
   — E.g., `CC[CG]C[AT][AT]T[GT]`
3. Use the model to BLAST again
   — Probably can get s_3 and/or s_4 now as the profile contains more `T`'s in the second half than *r*
4. Repeat #2 and #3 above until no more new sequences are returned

r: `CCCCTATG`
s_1: `CCCCTTTT`
s_2: `CCGCATTT`
s_3: `GCTCTTTT`
s_4: `AACCTTTT`

- This is similar to the Position-Specific Iterative BLAST (PSI-BLAST) algorithm

Multiple sequence alignment (MSA)

- In general, given a set of sequences, we want to align them all at the same time so that related characters are put in the same column

- As mentioned before, with 3 or more sequences, it quickly becomes infeasible to get the optimal solution by dynamic programming
  - Again, we need heuristics

- Now let's study these topics:
  - How to evaluate the goodness of a MSA (i.e., computing the alignment score of an MSA)
  - How to form a good MSA

Alignment score

- Suppose we have already got an alignment with 3 or more sequences. We want to evaluate how good it is. How can we compute an alignment score?

- Two possible ideas:
  - All pairs (e.g. average of 1 vs. 2, 1 vs. 3 and 2 vs. 3)
  - Compare each with a profile
    - Consensus sequence
    - Position weight matrix (PWM)
    - …

Example

- Suppose we have this alignment:

  r_1 : ACGGCT
  r_2 : GCGGTT
  r_3 : TGGG_T
  r_4 : TCGG_T

- Match: +1 score, mismatch/indel: -1 score

All pairs

- Scoring matrix:

	r_1	r_2	r_3	r_4
r_1	6	2	0	2
r_2	2	6	0	2
r_3	0	0	5	3
r_4	2	2	3	5

r_1 : ACGGCT
r_2 : GCGGTT
r_3 : TGGG_T
r_4 : TCGG_T

- Average alignment score = (2 + 0 + 0 + 2 + 2 + 3) / 6 = 9 / 6 = 1.5
- Note: the alignment between sequences r_3 and r_4 involves a "gap only" column – we simply ignore it

Consensus sequence

- Suppose we represent the alignment by the consensus sequence `TCGGCT`

- Alignment scores between each input sequence and consensus:
  - r_1: 4
  - r_2: 2
  - r_3: 2
  - r_4: 4
  - Average = (4 + 2 + 2 + 4) / 4 = 12 / 4 = 3

r_1 : `ACGGCT`
r_2 : `GCGGTT`
r_3 : `TGGG_T`
r_4 : `TCGG_T`

Performing multiple sequence alignment

- Many methods:
  - Clustal (ClustalW, ClustalX, Clustal Omega, etc.)
  - T-Coffee
  - MAFFT
  - MUSCLE
  - …
- We will study the main ideas behind Clustal

Clustal

- First proposed by Giggins and Sharp in 1988
- The popular version ClustalW (Clustal **w**eighted) was proposed by Thompson et al in 1994

- Main steps:
  - Compute distance matrix between all pairs of sequences
  - Construct a tree that captures the relationship between the sequences according to the distance matrix
  - Progressively align the sequences based on the tree

Distance matrix

- Distance matrix: similar to a scoring matrix, but larger number means more dissimilar

- Let's say we use Needleman-Wunsch to get optimal alignment and distance = length of alignment – alignment score

  –Here we only have the raw sequences and don't have the MSA yet

r_1 : ACGGCT
r_2 : GCGGTT
r_3 : TGGGT
r_4 : TCGGT

	r_1	r_2	r_3	r_4
r_1	0	4	6	4
r_2	4	0	6	4
r_3	6	6	0	2
r_4	4	4	2	0

From distance matrix to tree

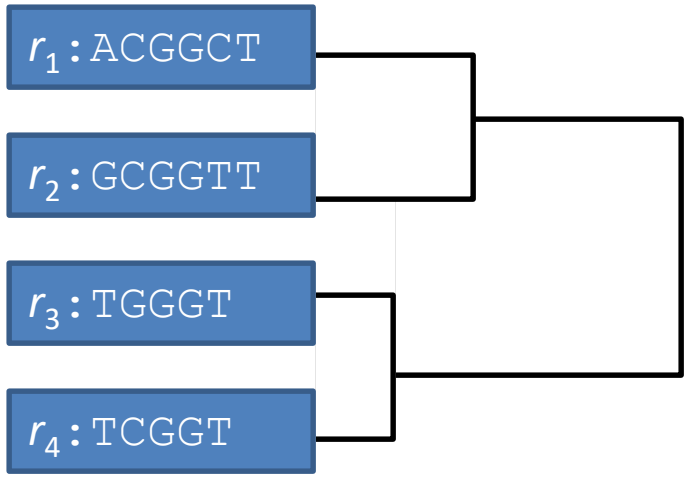
- Tree: Close sequences are put close to each other in the tree, branch length indicates distance

- Forming a tree (one possible way): repeatedly group the two closest sequences together (hierarchical clustering)
  - We will study more about tree construction later

	r_1	r_2	r_3	r_4
r_1	0	4	6	4
r_2	4	0	6	4
r_3	6	6	0	2
r_4	4	4	2	0

From distance matrix to tree

- A possible tree:



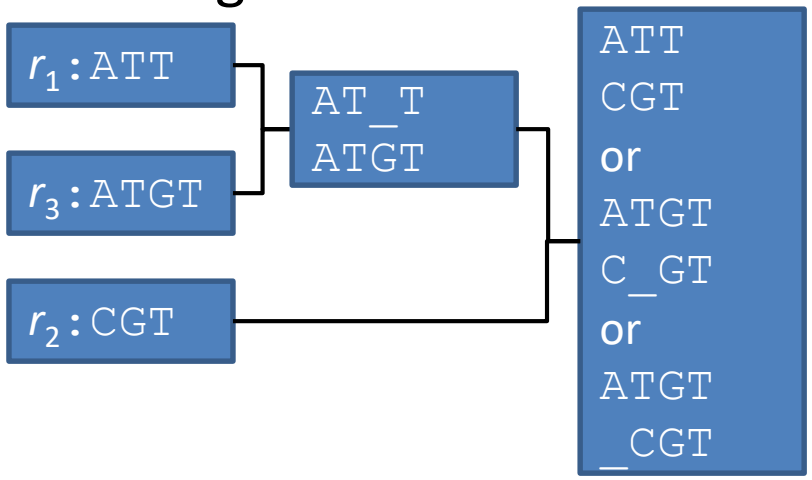
r_1 : ACGGCT
r_2 : GCGGTT
r_3 : TGGGT
r_4 : TCGGT

	r_1	r_2	r_3	r_4
r_1	0	4	6	4
r_2	4	0	6	4
r_3	6	6	0	2
r_4	4	4	2	0

A complete example

- r_1 = ATT, r_2 = CGT, r_3 = ATGT
- Match: 1; mismatch: -1; indel: -2
- Best alignment between r_1 and r_2:
  ATT
  CGT
  (score = -1, distance = 4)
- Best alignment between r_1 and r_3:
  AT_T
  ATGT
  (score = 1, distance = 3)
- Best alignments between r_2 and r_3:
  C_GT and _CGT
  ATGT ATGT
  (score = -1, distance = 5)
- Consensus between r_1 and r_3:
  r_{13} = ATT or ATGT
- Best alignments between r_{13} and r_2:
  ATT ATGT ATGT
  CGT or C_GT or _CGT
  (for ATT) (for ATGT)

Resulting tree:




```
r1 AT_T
r2 C_GT
r3 ATGT
```

Multiple sequence alignments:

```
AT_T  AT_T  AT_T
CG_T  C_GT  _CGT
ATGT or ATGT or ATGT
```

Epilogue

Case Study, Summary and Further Readings

Case study: High-impact work

- The 1990 BLAST paper by Altschul et al. has been cited 38,000 times, ranked 12th in the most highly-cited papers of all time by ISI Web of Science in 2014
  - PSI-BLAST was the 14th, with ~36,000 citations
  - (Check out more details about the list by yourself at http://www.nature.com/news/the-top-100-papers-1.16224)

- The method itself is one of the most used ones in bioinformatics.
  - The work is not only well-received in academia, but also heavily used in practice.

Case study: High-impact work

- Why the success?
  - Exponential growth in the amount of sequencing data
  - Optimal methods are too slow
    - BLAST is much faster
    - Seldom necessary to find “optimal” solution – mathematically optimal does not guarantee biological significance
  - E-value
    - Interpretability: What cutoff score would we use to define a “good” alignment?
    - Statistical basis

Case study: High-impact work

- Some ingredients of high-impact work:
  - Real needs
    - Not only now, but also future
    - No good solutions exist yet
  - Balance between theoretical elegance and practicality
  - User-friendliness
    - Easy-to-interpret inputs and outputs
  - Availability, stability and scalability
  - An appropriate name

Summary

- We need heuristic alignment methods because dynamic programming is infeasible for very long sequences and/or many sequences

- For pairwise alignment, FASTA and BLAST first find local matches, then extend/combine them to get longer matches
  - There are ways to evaluate statistical significance of matches

- For multiple sequence alignment, one way is to perform a series of pairwise alignments in a greedy manner

Further readings

- Chapter 4 of *Algorithms in Bioinformatics: A Practical Introduction*
  - More about E-values of BLAST
  - Additional searching algorithms
  - Free slides available
- Chapter 5 of *Algorithms in Bioinformatics: A Practical Introduction*
  - More details and additional methods
  - Free slides available
- Chapter 6 of *Algorithms in Bioinformatics: A Practical Introduction*
  - Methods for aligning whole genomes
  - Free slides available

Further readings

- Kent, BLAT – The BLAST-like Alignment Tool. *Genome Research* 12(4): 656-664, (2002)
  - Claimed to be 500 times faster for aligning DNA/mRNA and 50 times faster for aligning proteins than existing tools at that time
  - Due to indexing all non-overlapping k-mers in the genome and keeping it in memory
  - Major differences from BLAST:
    - BLAST indexes the query sequence, BLAT indexes the database
    - BLAST extends only when there are two proximal hits, BLAT can extend on any number of perfect or near-perfect hits
    - BLAST returns each local alignment separately, BLAT tries to stitch them together into a larger alignment
      - Particularly useful for handling exons and introns