1-month Practical Course
Genome Analysis (Integrative Bioinformatics & Genomics)

Lecture 6: Multiple sequence alignment (2) and homology searching (1)

Centre for Integrative Bioinformatics VU (IBIVU)
Vrije Universiteit Amsterdam
The Netherlands
ibivu.nl heringa@cs.vu.nl

Today:

- FASTA
  - Intermezzo: hashing
- BLAST
  - Intermezzo: DFA
- PSI-BLAST

Read in book: Higgs & Attwood
"Bioinformatics And Molecular Evolution"
Chapter 7 (pp. 139-157)

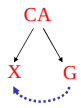
Searching for similarities

- The main question: what is the function of the new gene?
- The "lazy" investigation without doing experiments:
  - Find a set of similar proteins
  - Identify similarities and differences
  - For long proteins it is often good to identify domains first and then compare those separately

Inferring homology from similarity

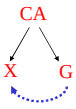
- Homology: sharing a common ancestor
  - a binary property (yes/no)
- Common ancestry makes it more likely that genes share the same function
  - It's a nice tool:

    When (a known gene) G is *homologous* to (an unknown gene) X, we gain a lot of information on X by transferring what we know about G



Can we just transfer information about structure and/or function?

- **Structure (and function) more conserved than sequence**
- **Sequence -> structure ->function**
- So, if the sequences already tell us it's the same thing (homolog), then certainly the structures and functions are supposed to be the same.
- This works most of the time, but there are cases where likely homology does not bear out.



Heuristic Alignment Motivation

- dynamic programming has performance $O(mn)$ which is too slow for large databases with high query traffic
  - MPsrch [Sturrock & Collins, MPsrch version 1.3 (1993) – Massively parallel DP]
- heuristic methods do fast approximation to dynamic programming
  - FASTA [Pearson & Lipman, 1988]
  - BLAST [Altschul *et al.*, 1990]

Heuristic Alignment Motivation

- consider the task of searching SWISS-PROT against a query sequence:
  - say our query sequence is 362 amino-acids long
  - SWISS-PROT release 38 contains 29,085,265 amino acids
- finding local alignments via dynamic programming would entail $O(10^{10})$ matrix operations
- many servers handle thousands of such queries a day (NCBI > 50,000)
- Each database search can be sped up by 'trivial parallelisation"

Sequence database searching – Homology searching

- Profile searching using Dynamic Programming — DP too slow for repeated database searches
- FASTA
- BLAST and PSI-BLAST

} Fast heuristics

- HMMER
- SAM-T99

} Hidden Markov modelling (more recent, slow)

Heuristic Alignment

- Today: FASTA and BLAST are discussed to show you a few of the tricks people have come up with to make alignment and database searching fast, while not losing too much quality.
- Next lecture will cover PSI-BLAST and related topics

FASTA

- First homology searching method (FASTP – Lipman & Pearson, 1985)
- Until gapped-BLAST and PSI-BLAST appeared (1997), FASTA has been the better method
- Compares a given query sequence with a library of sequences and calculates for each pair the highest scoring **local** alignment
- Speed is obtained by delaying application of the dynamic programming technique to the moment where most dissimilar sequences are already discarded by faster and less sensitive techniques (only relatively few putative related sequences left)
- FASTA routine operates in four steps:

FASTA

Operates in four steps:

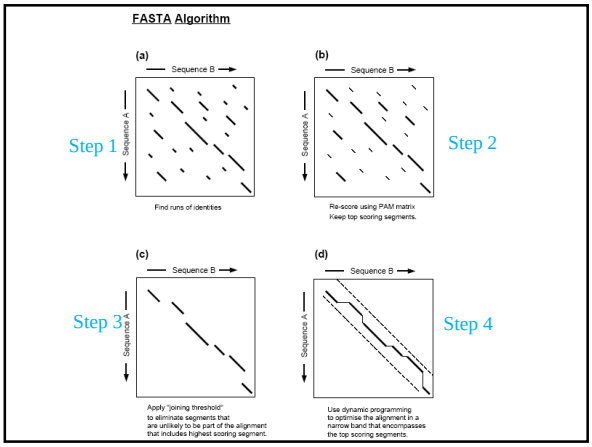
1. Rapid searches for **identical** words of a user specified length occurring in query and database sequence(s) (Wilbur and Lipman, 1983, 1984). For each database sequence the 10 ungapped regions with common words are determined.
2. These 10 regions are rescored using Dayhoff PAM-250 residue exchange matrix (Dayhoff et al., 1983) and the best scoring region of the 10 is reported under *init1* in the FASTA output.
3. Regions scoring higher than a threshold value *T* and being sufficiently near each other in the sequence are joined, now allowing gaps. The highest score of these new fragments can be found under *initn* in the FASTA output. *T* is set such that only a small fraction of database sequences are retained. These sequences are the only ones that are reported to the user. Until here things are quick!

FASTA

Operates in four steps (continued):

4. full dynamic programming alignment (Chao *et al.*, 1992) over the final region which is widened by 32 residues at either side, of which the score is written under *opt* in the FASTA output.

This is slow – $O(n^2)$



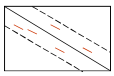
FASTA output example

```
DE METAL RESISTANCE PROTEIN YCF1 (YEAST CADMIUM FACTOR 1). . . .
SCORES Init1: 161 Initn: 161 Opt: 162 z-score: 229.5 E(): 3.4e-06
Smith-Waterman score: 162; 35.1% identity in 57 aa overlap

                                                10        20        30
test.seq                                      MQRSPLEKASVVSKLFFSWTRPILRKGYRQRLE
                                                :| :|::| |:::|:|:|||::|: |
YCFI_YEAST      CASILLLEALPKKPLMPHQHIHQTLTRRKPNPYDSANIFSRITFSWMSGLMKTGYEKYLV
             180       190       200       210       220       230

                    40        50        60
test.seq        LSDIYQIPSVDSADNLSEKLEREWDRE
                 :|:|::|   |:::||:|||::|: |
YCFI_YEAST      EADLYKLPRNFSSEELSQKLEKNWENELKQKSNPSLSWAICRTFGSKMLLAAFFKAIHDV
             240       250       260       270       280       290
```

FASTA

More about step 1 - Rapid identical word searches:

- Searching for *k*-tuples of a certain size within a specified bandwidth along search matrix diagonals.



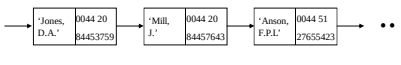
- For not-too-distant sequences (> 35% residue identity), little sensitivity is lost while speed is greatly increased.
- Technique employed is known as hash coding or hashing: a lookup table is constructed for all words in the query sequence, which is then used to compare all encountered words in each database sequence.

HASHING (general)

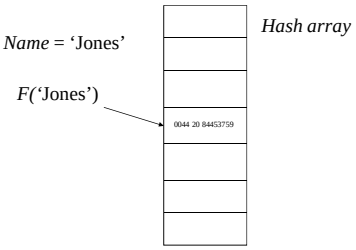
- rapid identical word searches
- a lookup table is constructed for all words in the query sequence, which is then used to compare all encountered words in each database sequence
- Example of hashing: the telephone book to find persons' phone numbers (names are ordered)
  -you do not need to search through all names until you find the person you want
  -In computer speak: find a function *f* such that *f*(*name*) can be directly assigned to an address in the computer memory, where the telephone number is then stored

HASHING (cont.)

This takes too long……

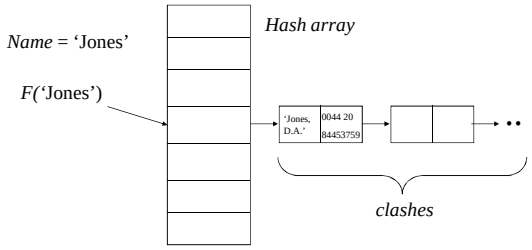


HASHING (cont.)



For sequences:
-name is *subword* in database sequence
-telephone number is biological *score* of subword

HASHING (cont.)



Hash function should avoid clashes:
-clashes take more time
-but need less memory for hash array

HASHING (cont.)

Example of hash function:

Take position of letter in alphabet (p(a)=1, p(b)=2, p(c)=3,..)

F('Jones') = p(J)+p(o)+p(n)+p(e)+p(s) = 10+15+14+5+19=63

So, 'Jones' goes to slot 63 in Hash array

What do you think about this function? Will there be clashes?

HASHING in FASTA

Sequence positions in query are hashed

Preprocessing the query sequence: make the hash table only once

Query: ERLFERLAC ……

DB: MERIFERLAC ……

Query hash table:

Word	*Position*
ER	1, 5
RL	2, 6
LF	3
FE	4
LA	7
AC	8
….	…

*You only need to go through each DB sequence once: for each word encountered (ME, ER, RI, IF, ..), check the query hash list for the word. If found, you immediately have the query sequence positions of the word. You also know the position you are at in the DB sequence, and so you can fill in the m*n matrix with diagonals (see earlier slide step 1).*

*Algorithmic speed therefore is linear with (DB) sequence length or O(n). Compare this to finding all word match positions without a hash list (complexity is O(m*n)).*

Hashing in FASTA: The Hash Array and the Chaining Array

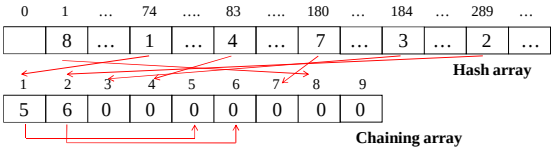
Query: ERLFERLAC One-let codes: ACDEFGHIKLMNPQRSTVWY
123456789 0 5 10 15

The Hash address is calculated using a 20-based numbering system to ensure that each dipeptide gets a separate slot in the hash array

Query hash table:

Word	*Hash*	*Position*
ER	3*20+14	1, 5
RL	14*20+9	2, 6
LF	9*20+4	3
FE	4*20+3	4
LA	9*20+0	7
AC	0*20+1	8

E=3, R=14
h('ER')=3×20+14



FASTA

- The k-tuple length (step 1) is user-defined and is usually 1 or 2 for protein sequences (i.e. either the positions of each of the individual 20 amino acids or the positions of each of the 400 possible dipeptides are located).
- For nucleic acid sequences, the k-tuple is 5-20 (often 11), and should be longer because short k-tuples are much more common due to the 4 letter alphabet of nucleic acids. The larger the k-tuple chosen, the more rapid but less thorough, a database search.

BLAST

- **B**asic **L**ocal **A**lignment **S**earch **T**ool
- BLAST heuristically finds *high scoring segment pairs* (HSPs):
  - It identifies segments in the query sequence and database sequences with statistically significant match scores
  - These are ungapped local alignments
- key trade-off: sensitivity vs. speed
- Sensitivity = number of significant matches detected/ number of significant matches in DB

Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990). Basic local alignment search tool. J. Mol. Biol. 215:403-410.

BLAST step 1: Determining Query Words

- Given:
  - query sequence: QLNFSAGW
  - word length w = 3 (Blast default)
  - word score threshold T = 8
- Step 1.1: determine all words of length w in query sequence

  QLN LNF NFS FSA SAG AGW

BLAST step 1: Determining Query Words

- Step 1.2: determine all words that score at least T when compared to a word in the query sequence:

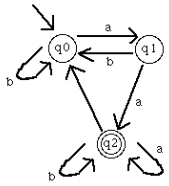
words from sequence	*query words w/ T=8*
QLN	QLN=11, QMD=9, HLN=8, ZLN=9,…
LNF	LNF=9, LBF=8, LBY=8, FNW=8,…
NFS	NFS=12, AFS=8, NYS=8, DFT=10,…
…	
SAG	none
...	

Scoring is done using the BLOSUM62 amino acid exchange matrix

BLAST step 2: Scanning the Database - DFA

- search database for all occurrences of query words
- can be a massive task
- approach:
  - build a DFA (deterministic finite-state automaton) that recognizes all query words
  - run DB sequences through DFA
  - remember hits

Scanning the Database - DFA



Moore paradigm: the alphabet is (a, b), the states are q0, q1, and q2, the start state is q0 (denoted by the arrow coming from nowhere), the only accepting state is q2 (denoted by the double ring around the state), and the transitions are the arrows. The machine works as follows. Given an input string, we start at the start state, and read in each character one at a time, jumping from state to state as directed by the transitions. When we run out of input, we check to see if we are in an accept state. If we are, then we accept. If not, we reject.

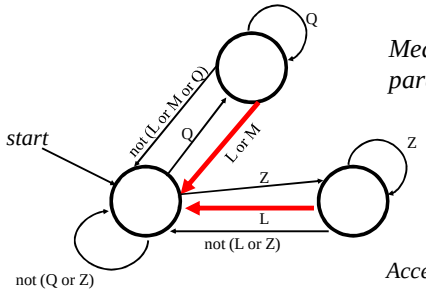
Moore paradigm: accept/reject states

Mealy paradigm: accept/reject transitions

Example:

- consider a DFA to recognize the query words: QL, QM, ZL
- All that a DFA does is read strings, and output "accept" or "reject."
- use Mealy paradigm (accept on transitions) to save space and time

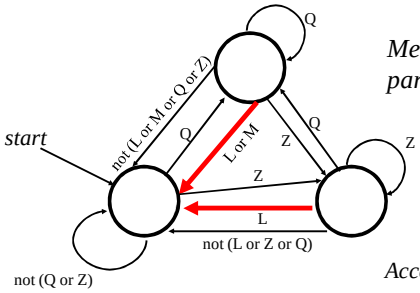
a DFA to recognize the query words: QL, QM, ZL in a fast way



Mealy paradigm

*Accept on **red** transitions*

a DFA to recognize the query words: QL, QM, ZL in a fast way



Mealy paradigm

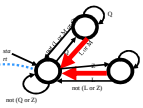
*Accept on **red** transitions*

Can you spot and justify the differences with the last slide?

1) Preprocess
2) Scan
3) Extend

BLAST, Step 2: Find "near-exact" matches with scanning

- Use all the *T*-similar *k*-words to build the Finite State Machine
- Scan for exact matches



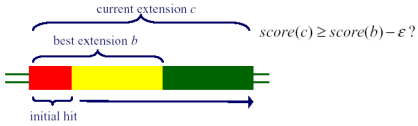
QKP
KKP
RQP
REP
RRP
RKP
...

movement

...VLQKPLKKPPLVKRQPCCEVVRKPLVKVIRCLA...

BLAST step 3: Extending Hits

- extend hits in both directions (without allowing gaps)
- terminate extension in one direction when score falls certain distance below best score for shorter extensions



- return segment pairs scoring at least *S*

1) Preprocess
2) Scan
3) Extend

BLAST, Step 3: Extending hits

- Having the list of matches (hits) we extend alignment in both directions

```
Query:      L  V  N  R  K  P  V  V  P
T-similar:            R  R  P
Database:   G  V  C  R  R  P  L  K  C
Score:     -3  4 -3  5  2  7  1 -2 -3
```

- …till the sum of scores drops below some level ε from the best known

Sensitivity *versus* Running Time

- the main parameter controlling the sensitivity vs. running-time trade-off is *T* (threshold for what becomes a query word)
  - small *T*: greater sensitivity, more hits to expand
  - large *T*: lower sensitivity, fewer hits to expand

BLAST 'flavours'

- **blastp** compares an amino acid query sequence against a protein sequence database
- **blastn** compares a nucleotide query sequence against a nucleotide sequence database
- **blastx** compares the six-frame conceptual protein translation products of a nucleotide query sequence against a protein sequence database
- **tblastn** compares a protein query sequence against a nucleotide sequence database translated in six reading frames
- **tblastx** compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.

BLAST (recap)

- Generates all tripeptides from a query sequence and for each of those the derivation of a table of similar tripeptides: number is only fraction of total number possible.
- Quickly scans a database of protein sequences for ungapped regions showing high similarity, which are called **high-scoring segment pairs** (HSP), using the tables of similar peptides. The initial search is done for a word of length *W* that scores at least the threshold value *T* when compared to the query using a substitution matrix.
- Word hits are then extended in either direction in an attempt to generate an alignment with a score exceeding the threshold of *S*, and as far as the cumulative alignment score can be increased.

BLAST (recap)

- Extension of the word hits in each direction are halted
  - when the cumulative alignment score falls off by a quantity ε from its maximum achieved value (see earlier picture)
  - the cumulative score goes to zero or below due to the accumulation of one or more negative-scoring residue alignments
  - upon reaching the end of either sequence
- The *T* parameter is the most important for the speed and sensitivity of the search resulting in the high-scoring segment pairs
- A **Maximal-scoring Segment Pair (MSP)** is defined as the highest scoring of all possible segment pairs produced from two sequences.

BLAST Notes

- may fail to find all HSPs
  - may miss seeds if *T* is too stringent
  - extension is greedy
- empirically, 10 to 50 times faster than Smith-Waterman
- large impact:
  - NCBI's BLAST server handles more than 200,000 queries a day
  - most widely used bioinformatics program

More Recent BLAST Extensions

- the two-hit method
- gapped BLAST
- PSI-BLAST

all are aimed at increasing sensitivity while keeping run-times minimal

Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25:3389-3402.

New BLAST step 3 (extension): The Two-Hit Method

- extension step typically accounts for 90% of BLAST's execution time
- key idea: do extension only when there are two hits on the same diagonal within distance A of each other
- to maintain sensitivity, lower *T* parameter
  - more single hits found
  - but only small fraction have associated 2nd hit

The Two-Hit Method

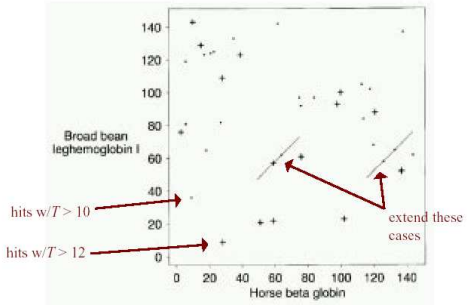


Figure from: Altschul et al. Nucleic Acids Research 25, 1997

Gapped BLAST

- trigger gapped alignment if two-hit extension has a sufficiently high score
  - So you first need to have two-hits on diagonal close enough together
- find length-11 segment with highest score; use central pair in this segment as seed
- run DP process both forward & backward from seed
- prune cells when local alignment score falls a certain distance below best score yet (same as extension in older version)

Gapped BLAST

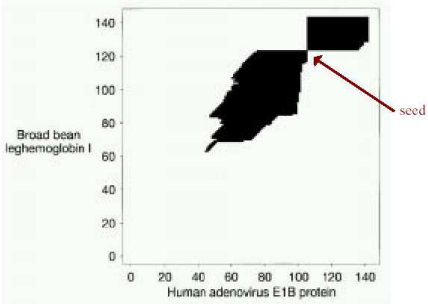


Figure from: Altschul et al. Nucleic Acids Research 25, 1997

The full new extension step: Combining the two-hit method and Gapped BLAST

- **Before:**
  - relatively high *T* threshold for 3-letter word (hashed) lists
  - two-way hit extension (see earlier slide)
- **Current (gapped) BLAST:**
  - Lower *T*: ungapped words (hits) made of 3-letter words are going to be longer (more 3-letter words accepted as match)
  - Relatively few hits (diagonal elements) will be on same matrix diagonal within a given distance *A*
  - 2-way local Dynamic Programming

*The new way is a little faster on average, **and** gives better (gapped) alignments and better alignment scores!*

Iterative homology searching using PSI-BLAST

PSI (*Position Specific Iterated*) BLAST

- basic idea
  - use results from BLAST query to construct a *profile matrix*
  - search database with profile instead of query sequence
- iterate

A Profile Matrix (Position Specific Scoring Matrix – PSSM)

sequence positions

amino acids	1	2	3	4	5	6	7	8
A			-2.4					
R			1.2					
D			0.5					
N			-0.2					
C			-3.1					

This is the same as a profile without position-specific gap penalties

Example: Profile calculation using frequency normalisation and log conversion (A) and scoring a sequence against PSSM (B)

(A)

S1 **GCTCC**
S2 **AATCG**
S3 **TACGC**
S4 **GTGTT**
S5 **GTAAA**
S6 **CGTCC**

	1	2	3	4	5	Overall
A	.17	.33	.17	.17	.17	6/30 = .20
C	.17	.17	.17	.50	.50	9/30 = .30
G	.50	.17	.17	.17	.17	7/30 = .23
T	.17	.33	.50	.17	.17	8/30 = .27

Normalise by dividing by overall frequencies

profile

	1	2	3	4	5
A	-0.23	0.72	-0.23	-0.23	-0.23
C	-0.81	-0.81	-0.81	0.74	0.74
G	1.11	-0.43	-0.43	-0.43	-0.43
T	-0.66	0.29	0.89	-0.66	-0.66

Convert to log to base of 2

	1	2	3	4	5	Overall
A	.85	1.65	.85	.85	.85	6/30 = .20
C	.57	.57	.57	1.67	1.67	9/30 = .30
G	2.17	.74	.74	.74	.74	7/30 = .23
T	.63	1.22	1.85	.63	.63	8/30 = .27

(B)

Match **GATCA** to PSSM

Find nucleotides at corresponding positions

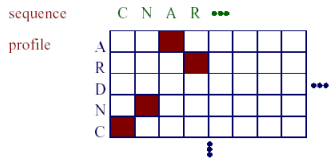
	1	2	3	4	5
A	-0.23	0.72	-0.23	-0.23	-0.23
C	-0.81	-0.81	-0.81	0.74	0.74
G	1.11	-0.43	-0.43	-0.43	-0.43
T	-0.66	0.29	0.89	-0.66	-0.66

Sum corresponding log odds matrix scores

Score = 1.11 + 0.72 + 0.89 + 0.74 - 0.23 = **3.23**

PSI BLAST

- Searching with a Profile
- aligning profile matrix to a simple sequence
  - like aligning two sequences
  - except score for aligning a character with a matrix position is given by the matrix itself
  - not a substitution matrix



PSI BLAST: Constructing the Profile Matrix

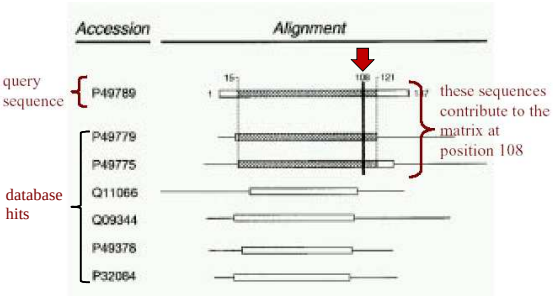
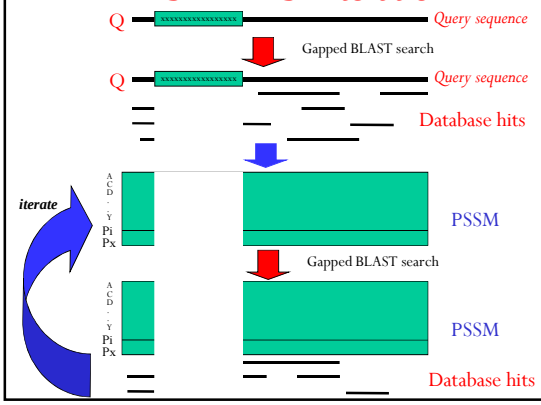
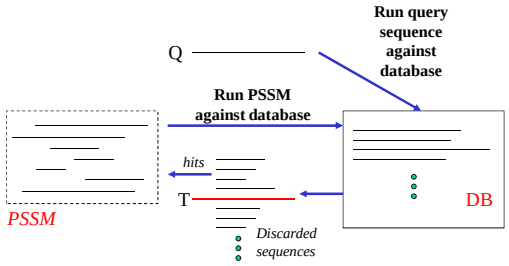


Figure from: Altschul et al. Nucleic Acids Research 25, 1997

PSI-BLAST iteration



Another PSI-BLAST iteration graphic…



PSI-BLAST steps in words

- The program initially operates on a single query sequence by performing a gapped BLAST search
- Then, the program takes significant local alignments (hits) found, constructs a multiple alignment (master-slave alignment) and abstracts a position-specific scoring matrix (PSSM) from this alignment.
- PSI-BLAST then rescans the database in a subsequent round, using the PSSM, to find more homologous sequences. Iteration continues until user decides to stop or search has converged, i.e. no more sequences can be added to the master-slave alignment.