CENTRE FOR INTEGRATIVE BIOINFORMATICS VU

Master Course

Algorithms in Sequence Alignment

Lecture 7

Homology searching (1)

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 the corresponding (sub)sequences separately
    - A domain is a unit of function
    - Multi-domain proteins have a compound function

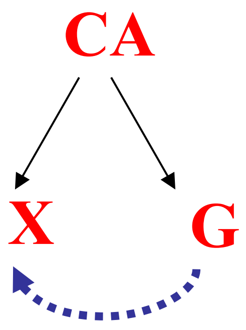
[2] Algorithms in *Sequence Analysis*

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:

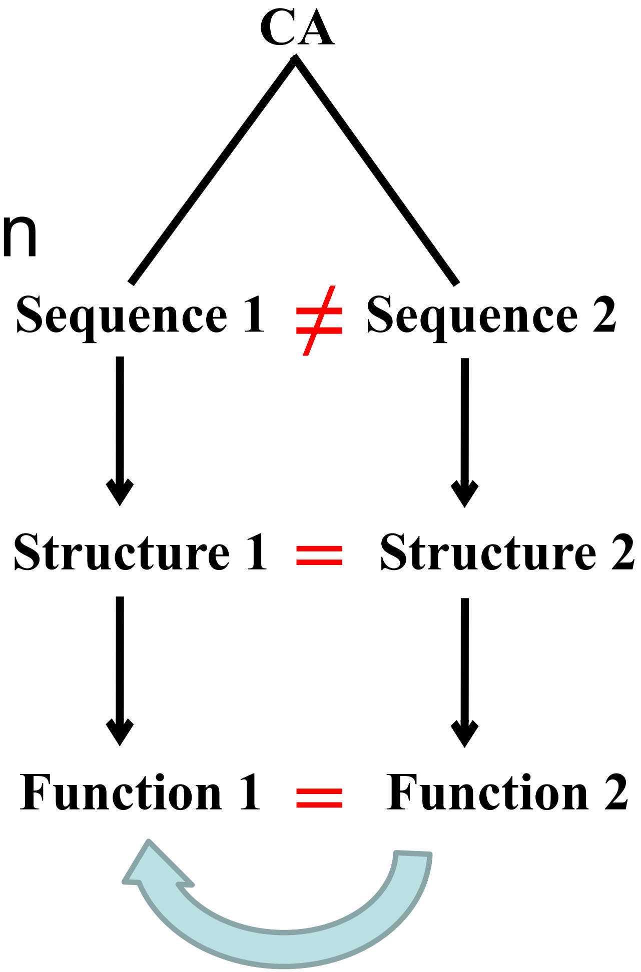
    When (a known gene) G is *homologous* to (an unknown gene) X, we gain a lot of information on X by transferring what we know about G



[3] Algorithms in *Sequence Analysis*

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 some cases where likely homology does not bear out.



CA

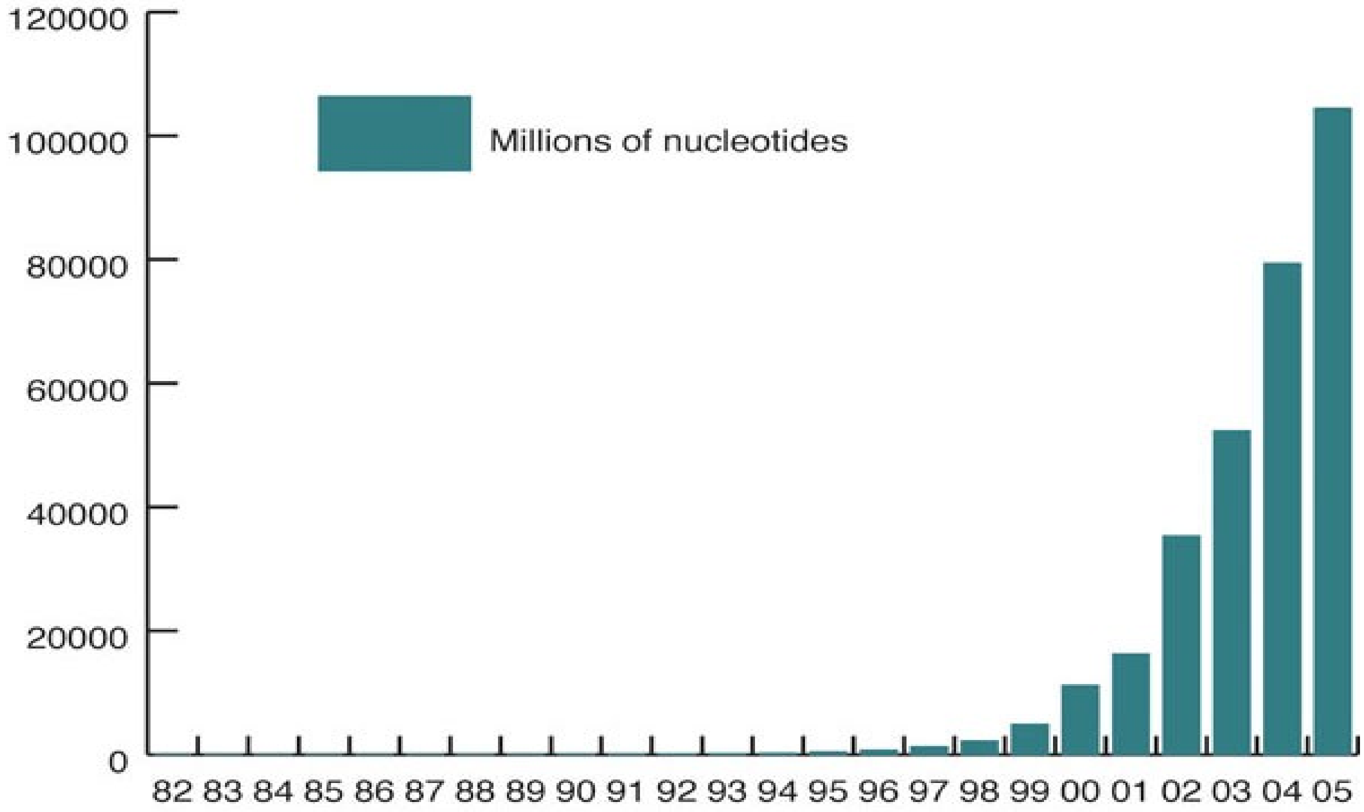
Sequence 1 ≠ Sequence 2

Structure 1 = Structure 2

Function 1 = Function 2

Sequence searching - challenges

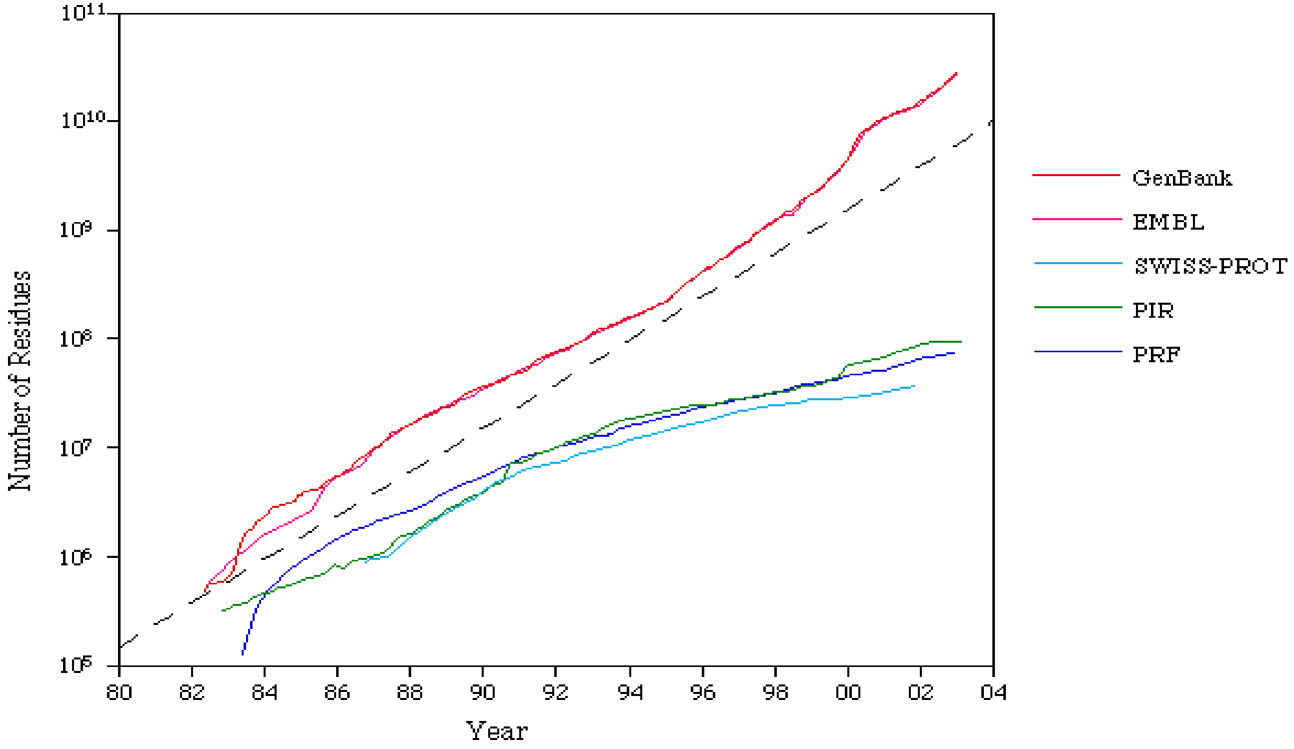
- Exponential growth of databases



[5] Algorithms in *Sequence Analysis*

Sequence searching - challenges

- **Exponential growth of databases**



[6] Algorithms in Sequence Analysis

Bioinformatics justification

- ***"Mind the Gap"***
- There are far more sequence data than structural/functional data
- We need to fill this gap by analysis and prediction pipelines

Sequence searching - definition

- Task:
  - Query: short, new sequence (~1000 residues)
  - Database (searching space): very many sequences
  - Goal: find seqs related to query
- We want:
  - fast tool
  - primarily a filter: most sequences will be unrelated to the query
  - fine-tune the alignment later

Heuristic Alignment Motivation

- heuristic methods perform fast approximation of dynamic programming
  - FASTA [Pearson & Lipman, 1988]
  - BLAST [Altschul *et al.*, 1990]

- the dynamic programming algorithm has complexity $O(mn)$, which is too slow for large databases with high query traffic

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 > 10^6/day)
- Using the DP algorithm for this is clearly prohibitive

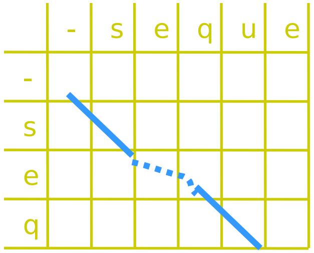
- Note: each database search can be sped up by ‘trivial parallelisation’ (each query-db comparison is independent)

Heuristic Alignment

- Today: the methods **BLAST**, **PSI-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.
- The earlier method **FASTA** is also discussed as a reference

What is BLAST

- **B**asic **L**ocal **A**lignment **S**earch **T**ool
- Bad news: it is only a heuristic
  - Heuristics: A rule of thumb that often helps in solving a certain class of problems, but makes no guarantees.
    Perkins, DN (1981) The Mind's Best Work
  - *Also see* http://en.wikipedia.org/wiki/Heuristic
- Basic idea:
  - Discard putatively unrelated sequences fast
  - High scoring segments have well conserved (almost identical) part
  - As well-conserved segments are identified, extend these to the *real* alignment



What does *well conserved* mean in BLAST?

- BLAST works with *k-words* (words of length k)
  - *k* is a parameter
  - different for DNA (>10) and proteins (2..4), default k values are 11 and 3, resp.
- word w_1 is *T-similar* to w_2 if the sum of pair scores is at least *T* (e.g. *T*=8)

Similar 3-words

W_1:	R	K	P	
W_2:	R	R	P	
Score:	5	2	7	$\Sigma = 14$

BLAST algorithm 3 basic steps

1) Preprocess
2) Scan
3) Extend

1) Preprocess the query sequence: extract all the *k-words*

2) Scan for *T-similar* matches in database (fast step)

3) Extend these to alignments (slow step)

BLAST, Step 1: Preprocess the query

1) Preprocess
2) Scan
3) Extend

- Take the query (e.g. `LVNRKPVVP`)
- Chop it into overlapping k-words (k=3 in this case)

```
Query:     LVNRKPVVP
Word1:     LVN
Word2:      VNR
Word3:       NRK
…
```

- *For each word find all similar words* (scoring at least *T*)
  - Build a table of similar words for each query 3-word (3-mer)
- E.g. for `RKP` the following 3-words are similar:

```
QKP   KKP   RQP   REP   RRP   RKP
```

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=15, ELN=12, HLN=10, QMD=9,…
LNF	LNF=16, LDF=11, FNY=9, LDY=8, …
NFS	NFS=16, NYS=13, AFS=8, DFT=8,…
…	
SAG	none
...	

Scoring is done using the BLOSUM62 amino acid exchange matrix

BLAST step 1: Determining Query Words

- Step 1.2: determine all *T* words for QLN

Word Alignment: `QLN Query`

`|||`

`YYY Database`

words w/ T=8 :

QLN

QLN 5+4+6 = 15

ELN 2+4+6 = 12

HLN 0+4+6 = 10

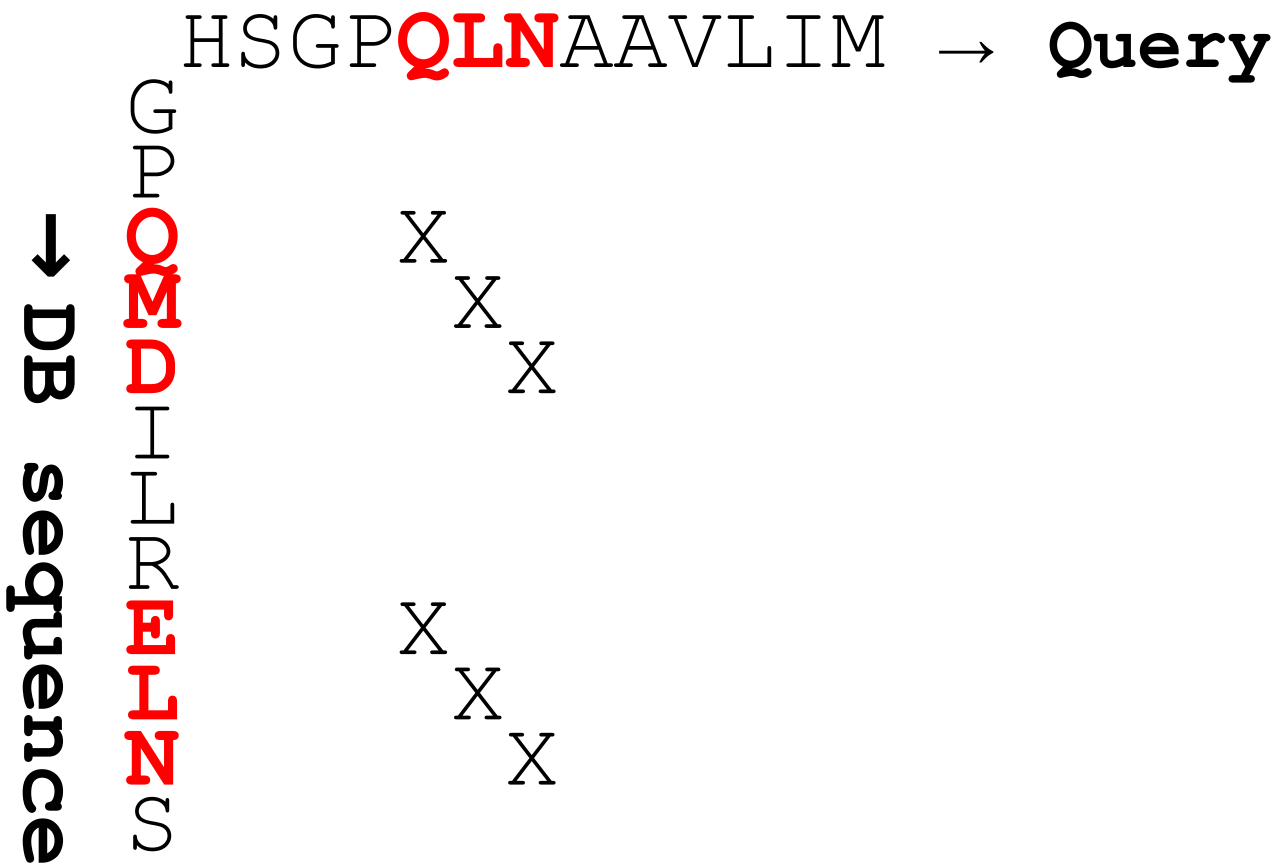
A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V

Blosum62

Scoring is done using the BLOSUM62 amino acid exchange matrix

BLAST step 1: Determining Query Words

For every neighbor of each k-mer, the naive way is to look all the k-mers in the database and check whether the neighbor matches anything



HSGPQLNAAVLIM → Query

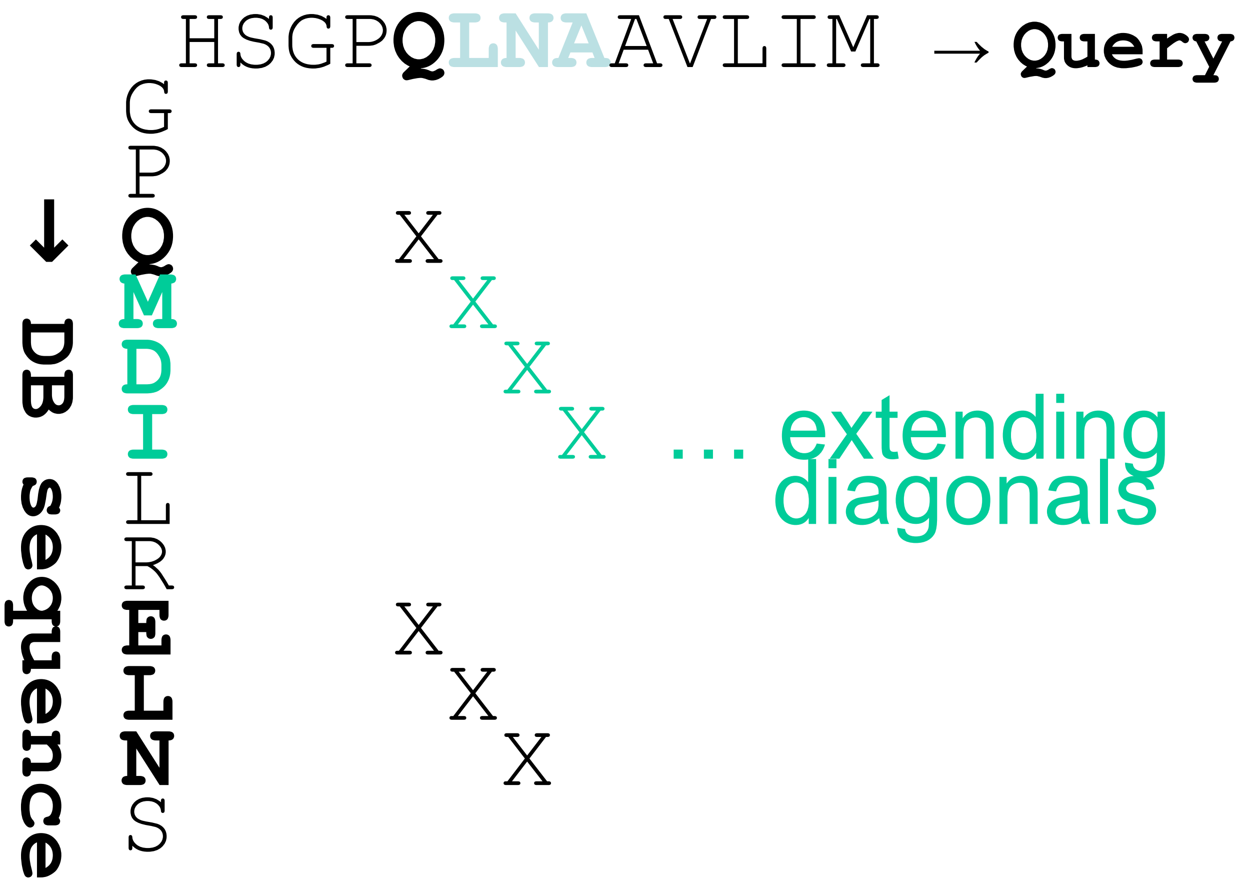
GPQMDILRELNS

DB sequence

Without a clever algorithm, one would (for each 3-mer in the query sequence) have to look at every 3-mer in the database sequence and check whether it is in the list of similar 3-mers for that query word. This would lead to O(N^2) computation steps, where N is the length of the DB sequence, which is too slow..

Query QLN: QLN=15, ELN=12, HLN=10, QMD=9,…

BLAST step 1: Determining Query Words



Query LNA: MDI=…, …

Without a clever algorithm, one would (for each 3-mer in the query sequence) have to look at every 3-mer in the database sequence and check whether it is in the list of similar 3-mers for that query word. This would lead to O(N^2) computation steps, where N is the length of the DB sequence, which is too slow..

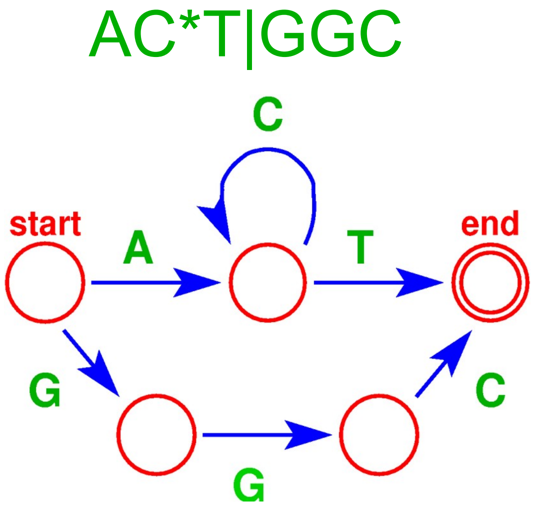
1) Preprocess
2) Scan
3) Extend

Step 2: Scanning the Database with DFA (Deterministic Finite-state Automaton)

- 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

DFA
Finite state machine

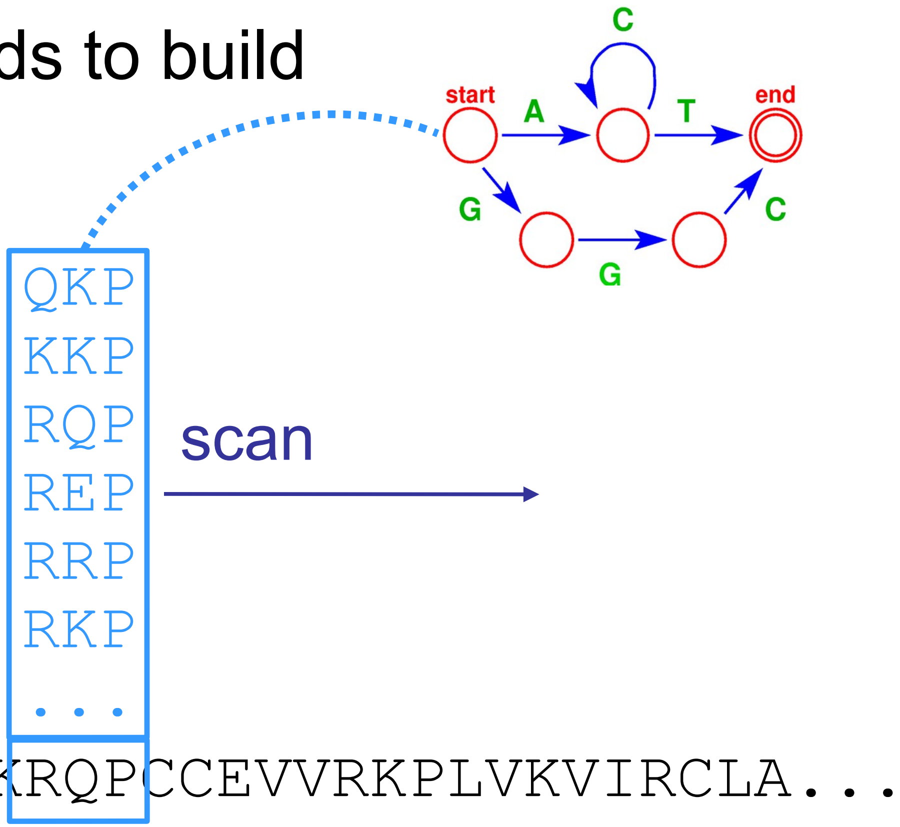
- abstract machine
- constant amount of memory (states)
- used in *computation* and *languages*
- recognizes *regular expressions*
    - `cp dmt*.pdf /home/john`



1) Preprocess
2) Scan
3) Extend

BLAST, Step 2: Find “exact” matches with scanning

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



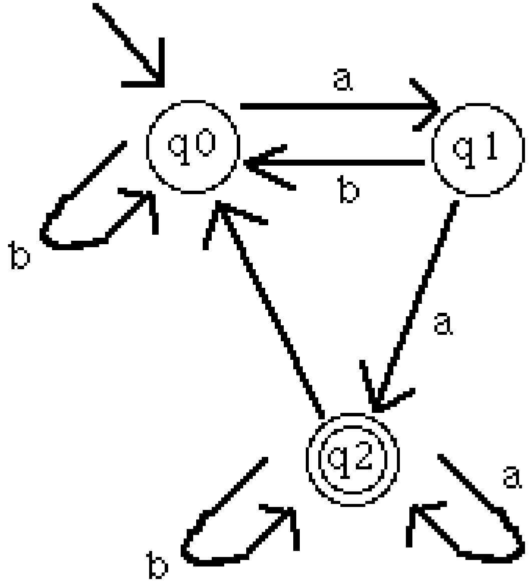
QKP
KKP
RQP
REP
RRP
RKP
. . .

scan

...VLQKPLKKPPLVKRQPCCEVVRKPLVKVIRCLA...

Scanning the Database - DFA

1) Preprocess
2) Scan
3) Extend



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

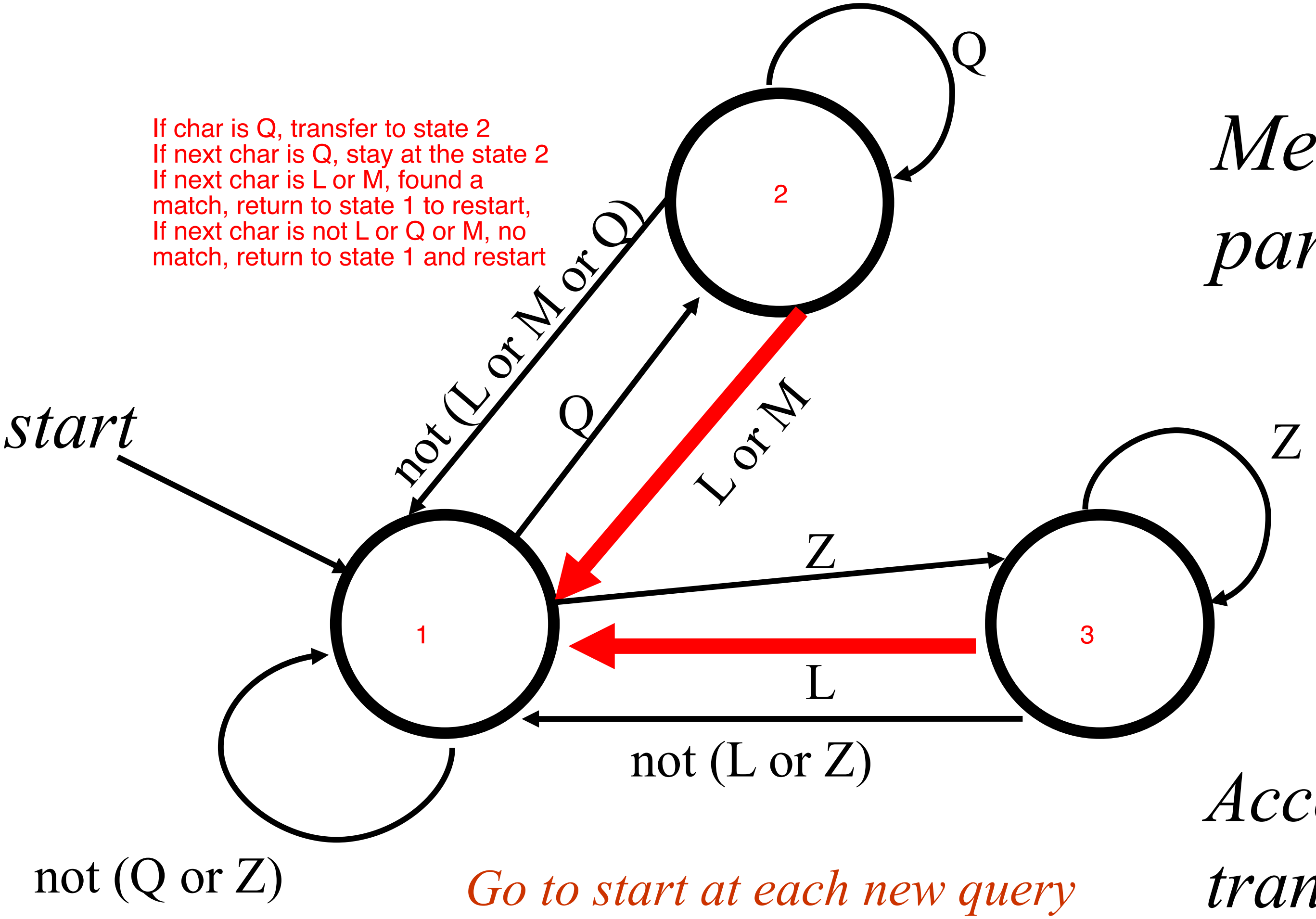
Mealy paradigm: accept/reject transitions

Example (next 2 slides):

- 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

1) Preprocess
2) Scan
3) Extend



If char is Q, transfer to state 2
If next char is Q, stay at the state 2
If next char is L or M, found a match, return to state 1 to restart,
If next char is not L or Q or M, no match, return to state 1 and restart

Mealy paradigm

*Accept on **red** transitions*

Go to start at each new query word

a DFA to recognize the query words

1) Preprocess
2) Scan
3) Extend

- Having preprocessed the query sequence 3-word tables (T-similar words to each of the 3-words in the query sequence) using a Mealy machine and having stored the places in the query sequence where each of the 3-words start (a given 3-word can occur more than once in the sequence), BLAST can now very rapidly fill the alignment search matrix with query-sequence *versus* DB-sequence diagonals (ungapped alignments). This is done for each DB sequence.

1) Preprocess
2) Scan
3) Extend

BLAST, Step 3: Extending “exact” matches

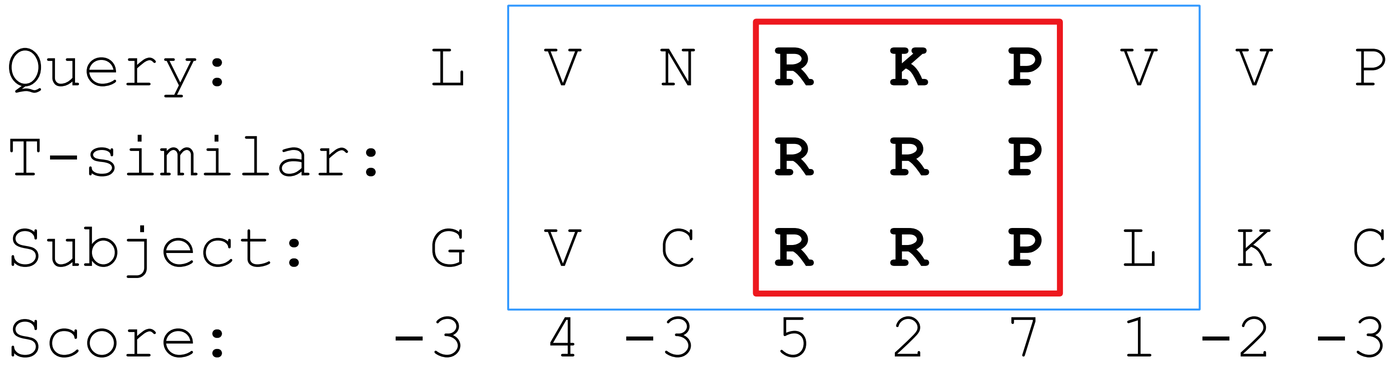
1. Classical BLAST (not in use anymore): two-way non-gapped extension

2. Current BLAST: two-hit method and extention using DP (Gapped-BLAST)

1) Preprocess
2) Scan
3) Extend

BLAST, Step 3: Non-gapped extention (old BLAST)

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



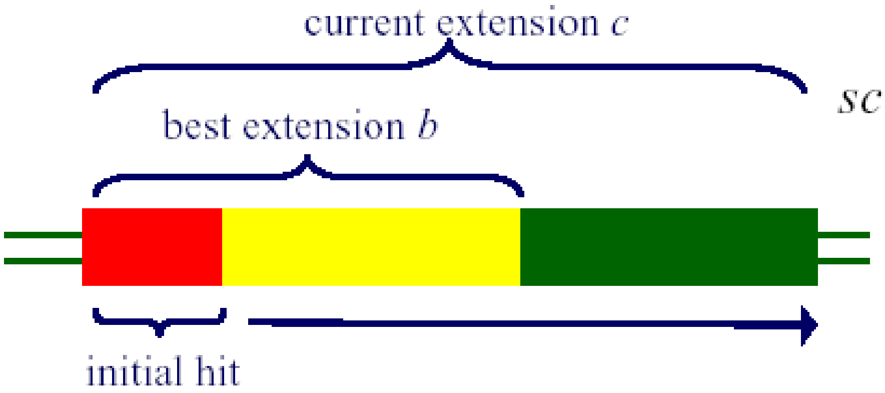
Query:	L	V	N	**R**	**K**	**P**	V	V	P
T-similar:				**R**	**R**	**P**			
Subject:	G	V	C	**R**	**R**	**P**	L	K	C
Score:	-3	4	-3	5	2	7	1	-2	-3

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

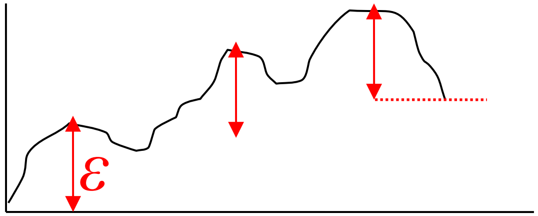
Step 3: Non-gapped extention (old BLAST)

1) Preprocess
2) Scan
3) Extend

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



$$score(c) \geq score(b) - \varepsilon\ ?$$



- return segment pairs scoring at least a threshold score *S*

Step 3:Current BLAST - Extending hits

- two-hit method
- gapped BLAST

Altschul et al., *Nucleic Acids Research* 1997

Two-Hit Method - justification

- old extension step typically accounts for 90% of BLAST's execution time
- key idea: do extension only when there are two non-overlapping hits on the same diagonal within distance A of each other (A = 40 a.a.)
- to maintain sensitivity, lower *T* parameter
  - more single hits found
  - but only small fraction have associated 2nd hit

The Two-Hit Method

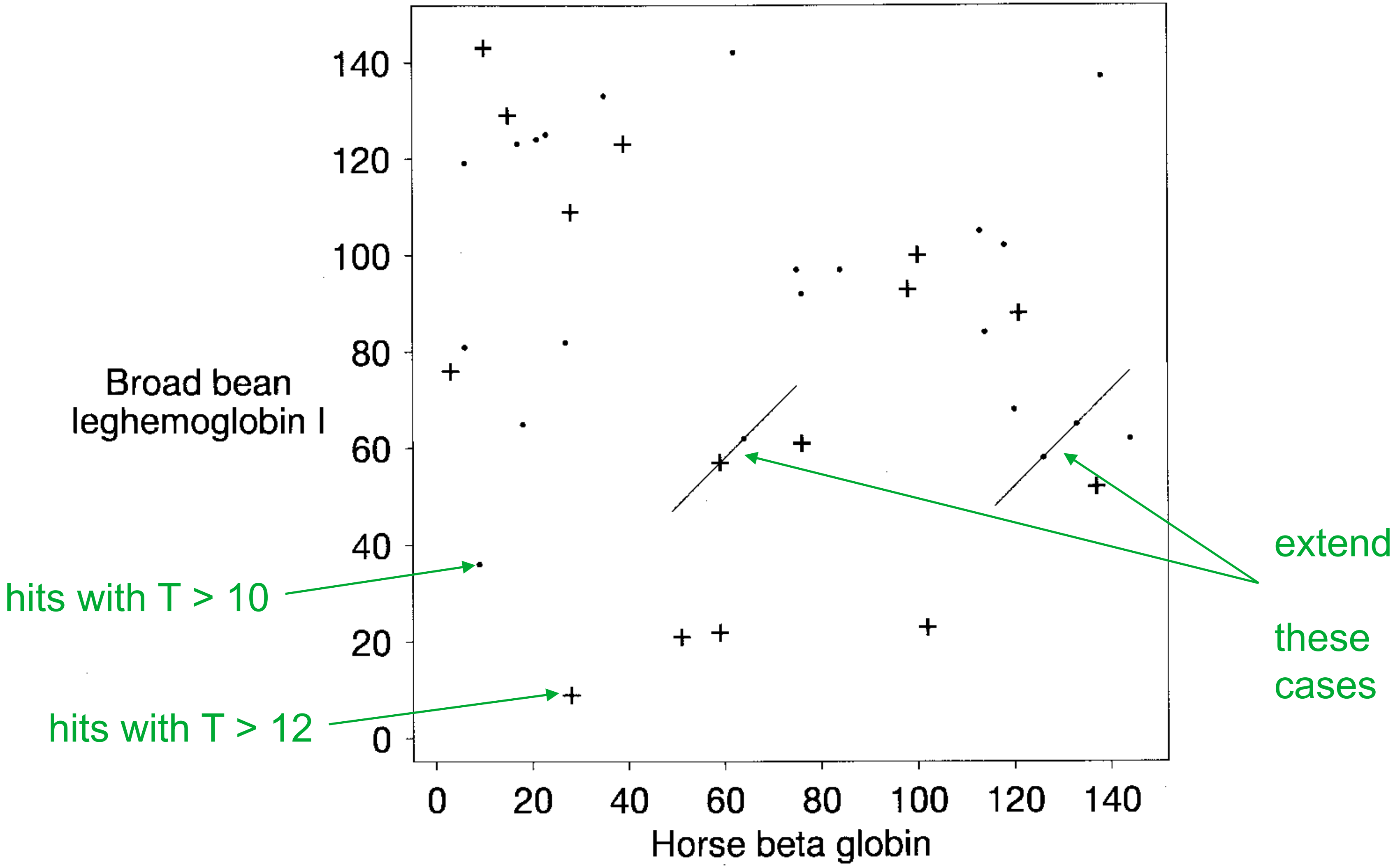
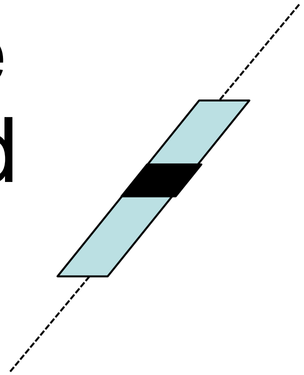


Figure from: Altschul et al. Nucleic Acids Research 25, 1997

Gapped BLAST

- trigger gapped alignment if two-hit extension has a sufficiently high score
- slide diagonal window to 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



Gapped BLAST

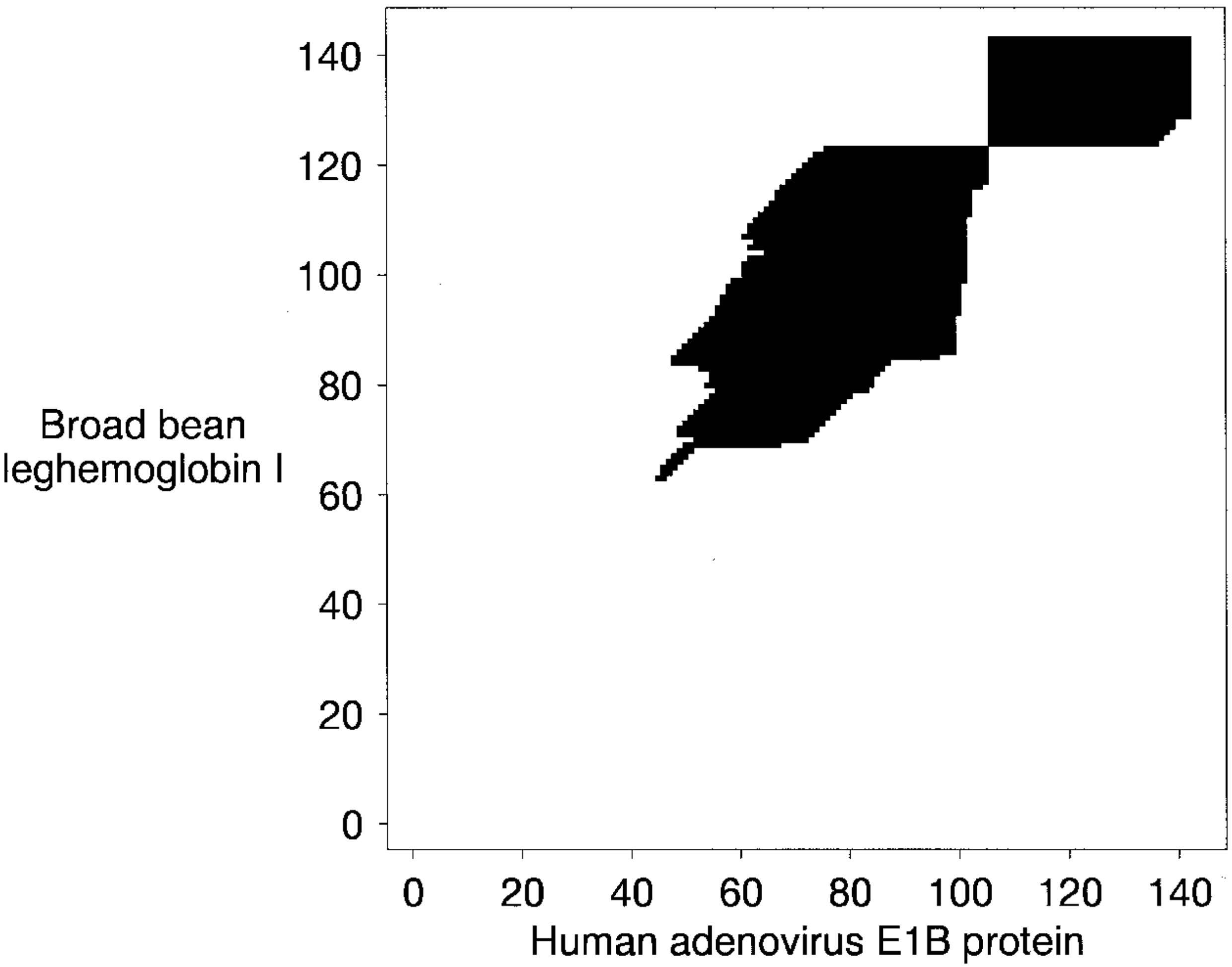
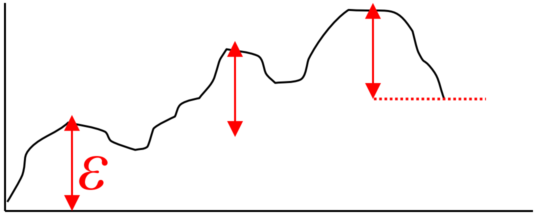


Figure from: Altschul et al. Nucleic Acids Research 25, 1997



“Greedy” cutoff also used here as stop criterion

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 slides)
- **Current BLAST:**
  - Lower *T*: many more hits (more 3-letter words accepted as match)
  - Relatively few hits (diagonal elements) will be on same matrix diagonal within a given distance *A*
  - Perform 2-way local Dynamic Programming (gapped BLAST) only on ‘two-hits’ (preceding bullet)

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

More recent BLAST-related developments

- hashing (indexing) the database
- PSI-BLAST

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

Altschul et al., *Nucleic Acids Research* 1997

Making things even faster- indexing the complete database (or genome sequence)

- SSAHA – Sequence Search and Alignment by Hashing Algorithms (Ning et al., 2001)
- BLAT – BLAST-like Alignment Tool (Kent, 2002)
- PatternHunter (Ma et al., 2002)
- BLASTZ – alignment of genomic sequences (Schwartz et al., 2003)

BLAT – BLAST-Like Alignment Tool

- Analyzing vertebrate genomes requires rapid mRNA/DNA and cross-species protein alignments. **BLAT** (the BLAST-like alignment tool) was developed by Jim Kent from UCSC. It is more accurate and 500 times faster than popular existing tools such as BLAST for mRNA/DNA alignments and 50 times faster for protein alignments at sensitivity settings typically used when comparing vertebrate sequences (e.g. BLAST).
- BLAT's speed stems from an index of all nonoverlapping k-mers in the genome. This index fits inside the RAM of inexpensive computers, and need only be computed once for each genome assembly. BLAT has several major stages. It uses the index to find regions in the genome likely to be homologous to the query sequence. It performs an alignment between homologous regions. It stitches together these aligned regions (often exons) into larger alignments (typically genes). Finally, BLAT revisits small internal exons possibly missed at the first stage and adjusts large gap boundaries that have canonical splice sites where feasible.
- **From Wikipedia, the free encyclopedia**

More on Kent

For reference

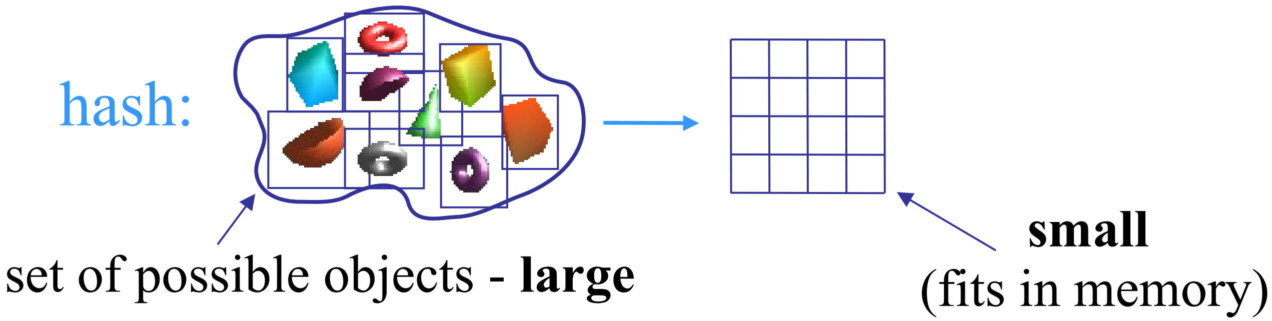
Involvement with the Human Genome Project

While working on his PhD in Biology at the University of California, Santa Cruz, Kent in May 2000, wrote a program, GigAssembler,[5] that allowed the publicly funded Human Genome Project to assemble and publish the human genome sequence. His efforts were motivated out of concern that the data might be made proprietary via patents by Celera Genomics.[6] In their close race with Celera, Kent and the UCSC Professor David Haussler quickly built a modest cluster of 50 commodity Personal Computers running the Linux operating system to run the software. In contrast, Celera was using what was thought then to be one of the most powerful civilian supercomputers in the world. Kent's first assembly on the human genome was released on June 22. Celera finished its assembly 3 days later on June 25, and the dual results were announced at the White House on June 26. On July 7, 2000, the Santa Cruz data was made publicly available on the World Wide Web while the research paper describing this publicly funded genome was published in February 2001 special issue of *Nature*,[7] in parallel with Celera's results in the journal *Science*.[8] In 2002 Tim O'Reilly described Kent's work as "the most significant work of open source development in the past year".

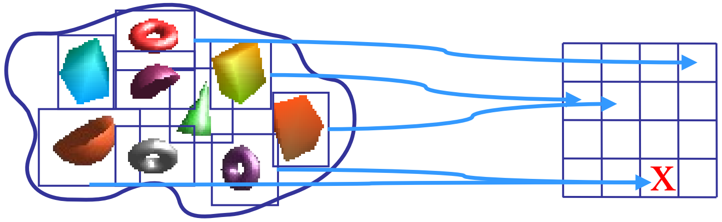
From Wikipedia, the free encyclopedia

Hashing - associative arrays

- Indexing with the object, the
- Hash function:



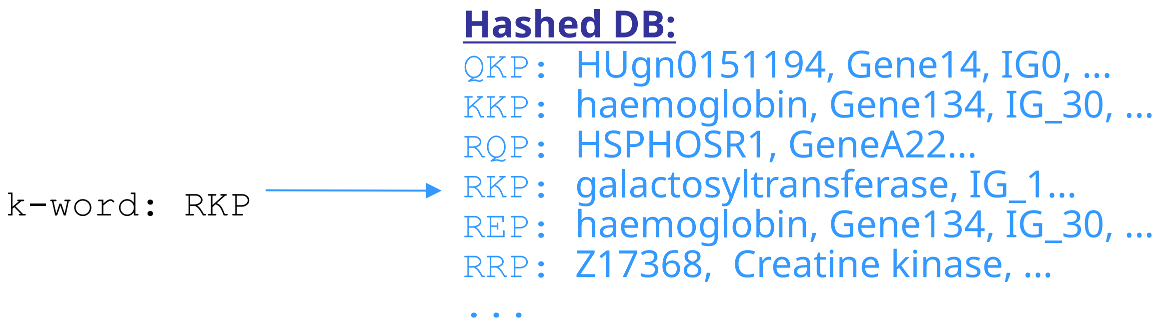
- Objects should be "well spread"



Indexing the database: Find ”exact” matches with hashing

1) Preprocess
2) Scan
3) Extend

- Preprocess the database
  - Hash the database with k-words
  - For each k-word store in which sequences it appears



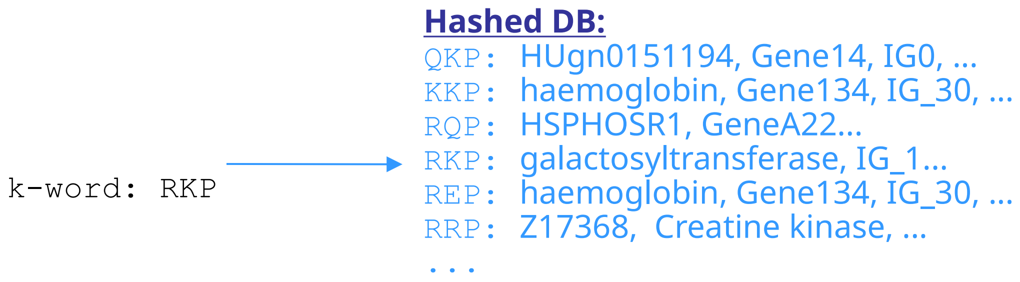
k-word: RKP

Hashed DB:

QKP: HUgn0151194, Gene14, IG0, …
KKP: haemoglobin, Gene134, IG_30, …
RQP: HSPHOSR1, GeneA22…
RKP: galactosyltransferase, IG_1…
REP: haemoglobin, Gene134, IG_30, …
RRP: Z17368, Creatine kinase, …
. . .

Indexing the database: Find “exact” matches with hashing

- The database is preprocessed only once! (independent from the query)
- In a constant time we can get the sequences with a certain *k-word*



k-word: RKP

Hashed DB:

QKP:	HUgn0151194, Gene14, IG0, ...
KKP:	haemoglobin, Gene134, IG_30, ...
RQP:	HSPHOSR1, GeneA22...
RKP:	galactosyltransferase, IG_1...
REP:	haemoglobin, Gene134, IG_30, ...
RRP:	Z17368, Creatine kinase, ...
. . .	

BLAST flavours

- `blastp`: protein query, protein db
- `blastn`: DNA query, DNA db
- `blastx`: DNA query, protein db
  - in all reading frames. Used to find potential translation products of an unknown nucleotide sequence.
- `tblastn`: protein query, DNA db
  - database dynamically translated in all reading frames.
- `tblastx`: DNA query, DNA db
  - all translations of query against all translations of db (compare at protein level)

PSI-BLAST

- Position-Specific Iterated BLAST
- A *profile* (called PSSM by BLAST – *Position Specific Scoring Matrix*) is derived from the result of the first search (using a single query sequence)
- Database is searched against the profile (instead of a sequence) in subsequent rounds
- Up to 3-10 iterations are recommended

PSI-BLAST steps in words

1. Query sequences are first scanned for the presence of so-called *low-complexity regions* (Wooton and Federhen, 1996), *i.e.* regions with a biased composition likely to lead to spurious hits are excluded from alignment.
2. The program then initially operates on a single query sequence by performing a gapped BLAST search
3. Next, 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.
4. The database is rescanned in a subsequent round, now using the PSSM, to find more homologous sequences. Iteration continues until user decides to stop or the search has converged

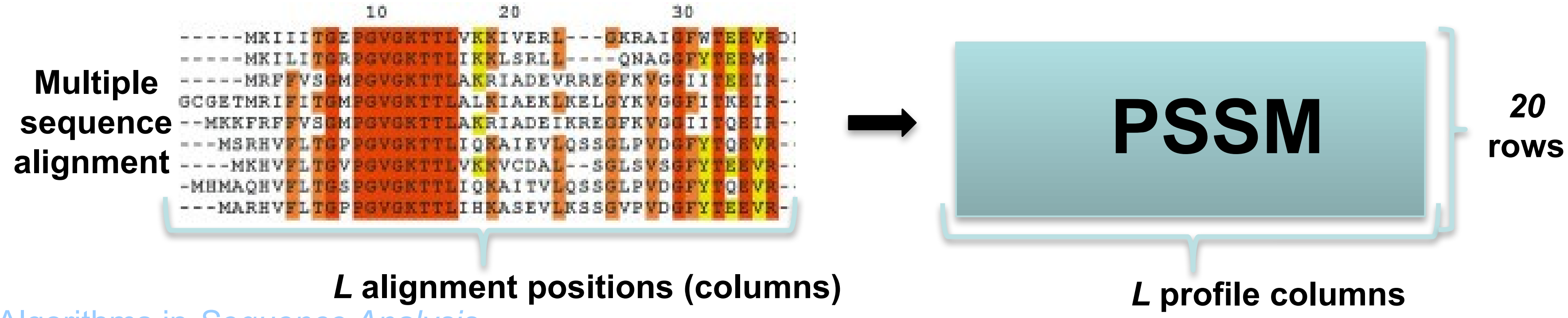
Profile

- a **Profile** is a generalized form of sequence (better search capability)
- probabilities instead of a letter

A	0.5	0.3	0.2	...	0
C	0	0.1	0.0	...	0.5
D	0	0	0.1	...	0.2
.	.	.	.	...	.
.	.	.	.	...	.
.	.	.	.	...	.
W	0	0.3	0.4	...	0.1
Y	0.5	0.3	0.3	...	0.2

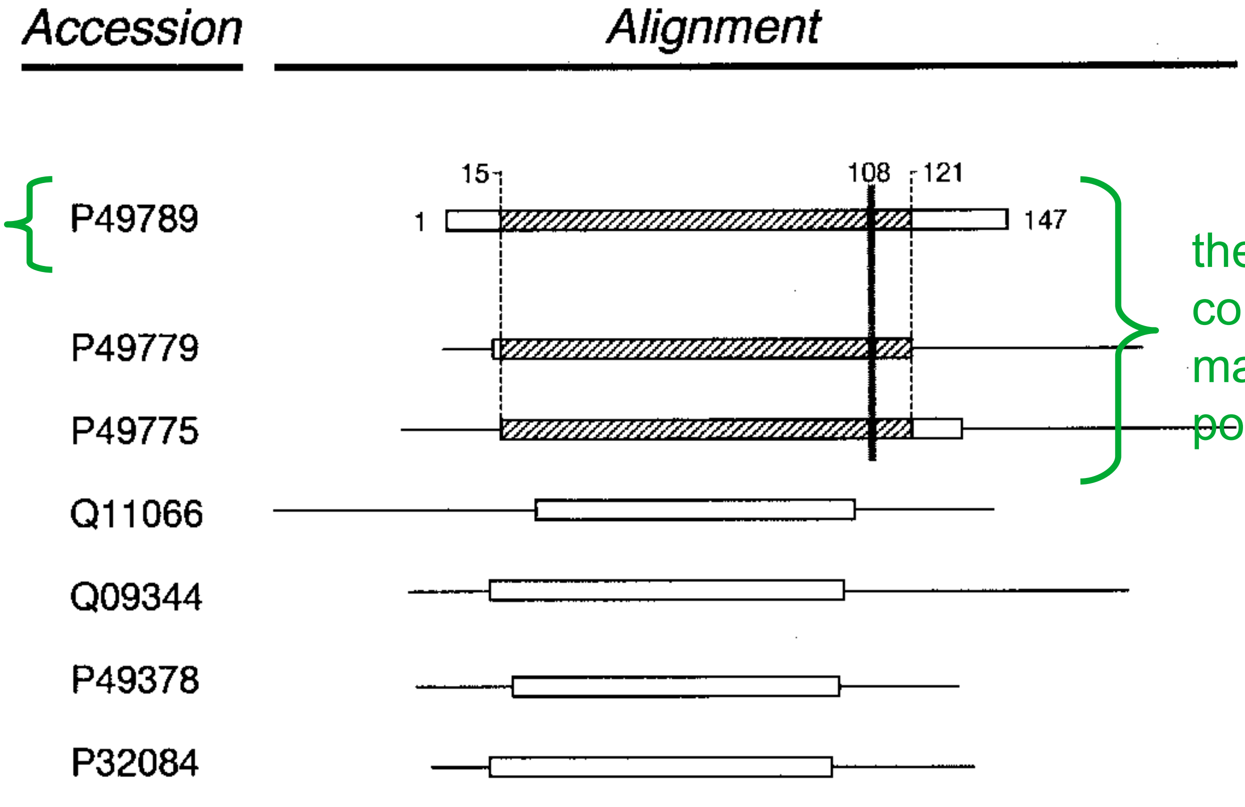
What is a sequence profile matrix?

- A sequence profile is a frequency-based scoring matrix that approximates the likelihood of occurrence of an amino acid at a given (multiple) alignment position
  - The mathematics to convert the a.a. frequencies to probabilities may differ (BLAST uses log conversion)
  - Basically, the scoring matrix has dimension *L* * 20, with L the number of columns (positions) in the multiple alignment (or BLAST alignment)
  - Profiles may have an extra column (21st position) to describe position-specific gap penalties.
  - In BLAST a profile is called **Position-Specific Scoring Matrix** (PSSM) and has only 20 columns.


Multiple sequence alignment
10
20
30
-----MKIIITGEPGVGKTTLVKKIVERL---GKRAIGFWTEEVRDI
-----MKILITGRPGVGKTTLIKKLSRLL----QNAGGFYTEEMR-
-----MRFFVSGMPGVGKTTLAKRIADEVRREGFKVGGIITEEIR-
GCGETMRIFITGMPGVGKTTLALKIAEKLKELGYKVGGFITKEIR-
--MKKFRFFVSGMPGVGKTTLAKRIADEIKREGFKVGGIITQEIR-
---MSRHVFLTGPPGVGKTTLIQKAIEVLQSSGLPVDGFYTQEVR-
----MKHVFLTGVPGVGKTTLVKKVCDAL--SGLSVSGFYTEEVR-
-MHMAQHVFLTGSPGVGKTTLIQKAITVLQSSGLPVDGFYTQEVR-
---MARHVFLTGPPGVGKTTLIHKASEVLKSSGVPVDGFYTEEVR-
L alignment positions (columns)
PSSM
20 rows
L profile columns

PSI BLAST: Constructing the Profile Matrix

query sequence



these sequences contribute to the matrix at position 108

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

PSI-BLAST profile calculation example using frequency normalisation and log conversion

(A)

```
    12345
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

	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

Convert to log to base of 2

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

(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: Determining profile elements more reliably using pseudo-counts

- the value for a given element of the profile matrix is given by:

$$matrix(i,j) = \log\left(\frac{\Pr(a_i \mid \mathrm{col}=j)}{\Pr(a_i)}\right)$$

Overall alignment frequency (preceding slide)

- where the probability of seeing amino acid a_i in column j is estimated as:

Observed frequency

Pseudocount (e.g. database frequency)

$$\Pr(a_i \mid \mathrm{col}=j) = \frac{\alpha f_{ij} + \beta g_{ij}}{\alpha + \beta}$$

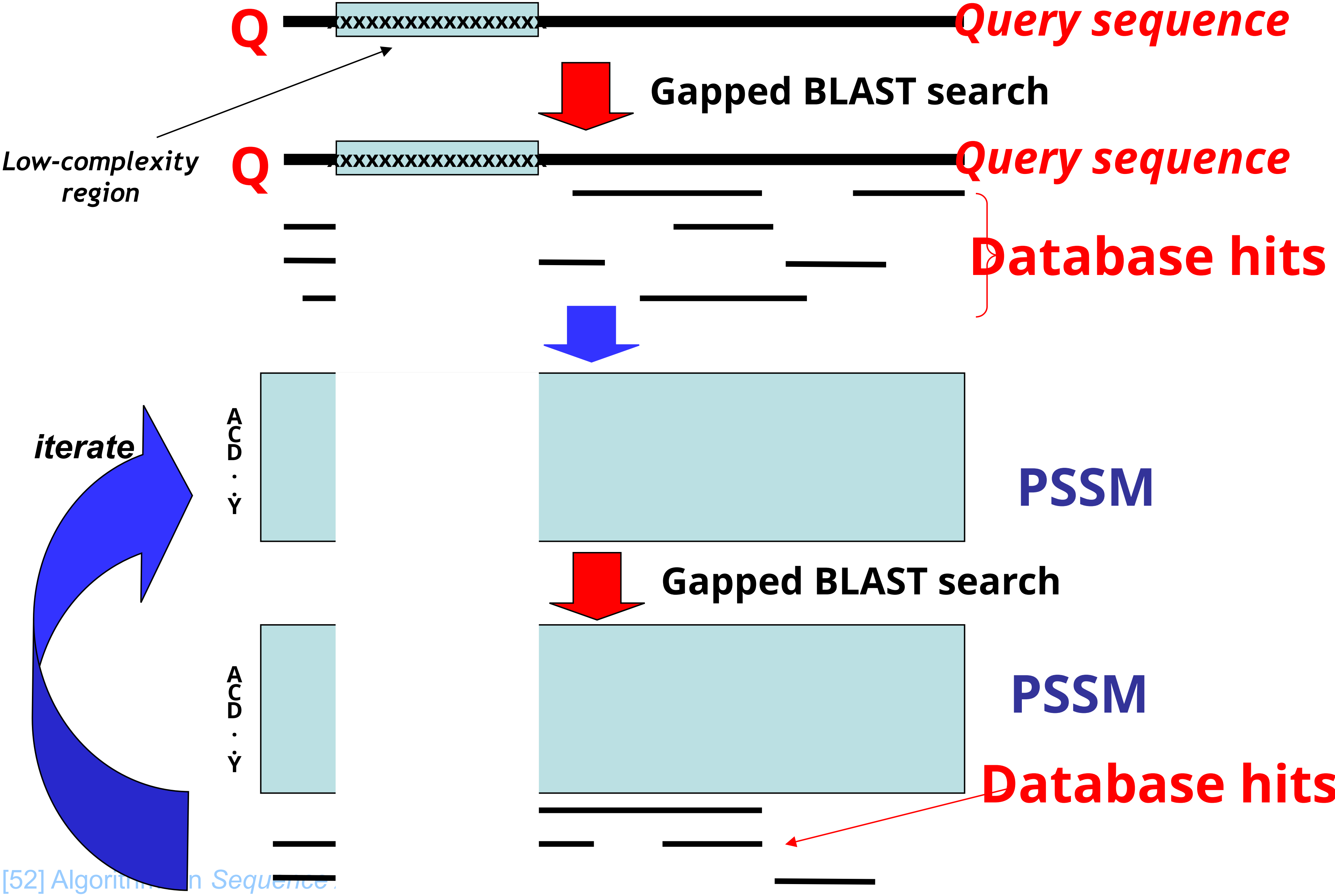
e.g. α = number of sequences in profile, β=1

PSI BLAST: Determining profile elements more reliably using pseudo-counts

Pseudo-counts:

- mix observed a.a. frequencies with prior (e.g. database) frequencies
- useful when multiple alignment contains only few sequences so that there is no statistical sample per column yet
- drawback is pulling all frequencies to prior frequencies, which reduces the differences between sequences
- with greater numbers of sequences in the MSA, the profile becomes less dependent on priors

PSI-BLAST iteration graphic…



Q xxxxxxxxxxxxxxxxx *Query sequence*

Gapped BLAST search

Low-complexity region

Q xxxxxxxxxxxxxxxxx *Query sequence*

Database hits

A C D . . Y

PSSM

iterate

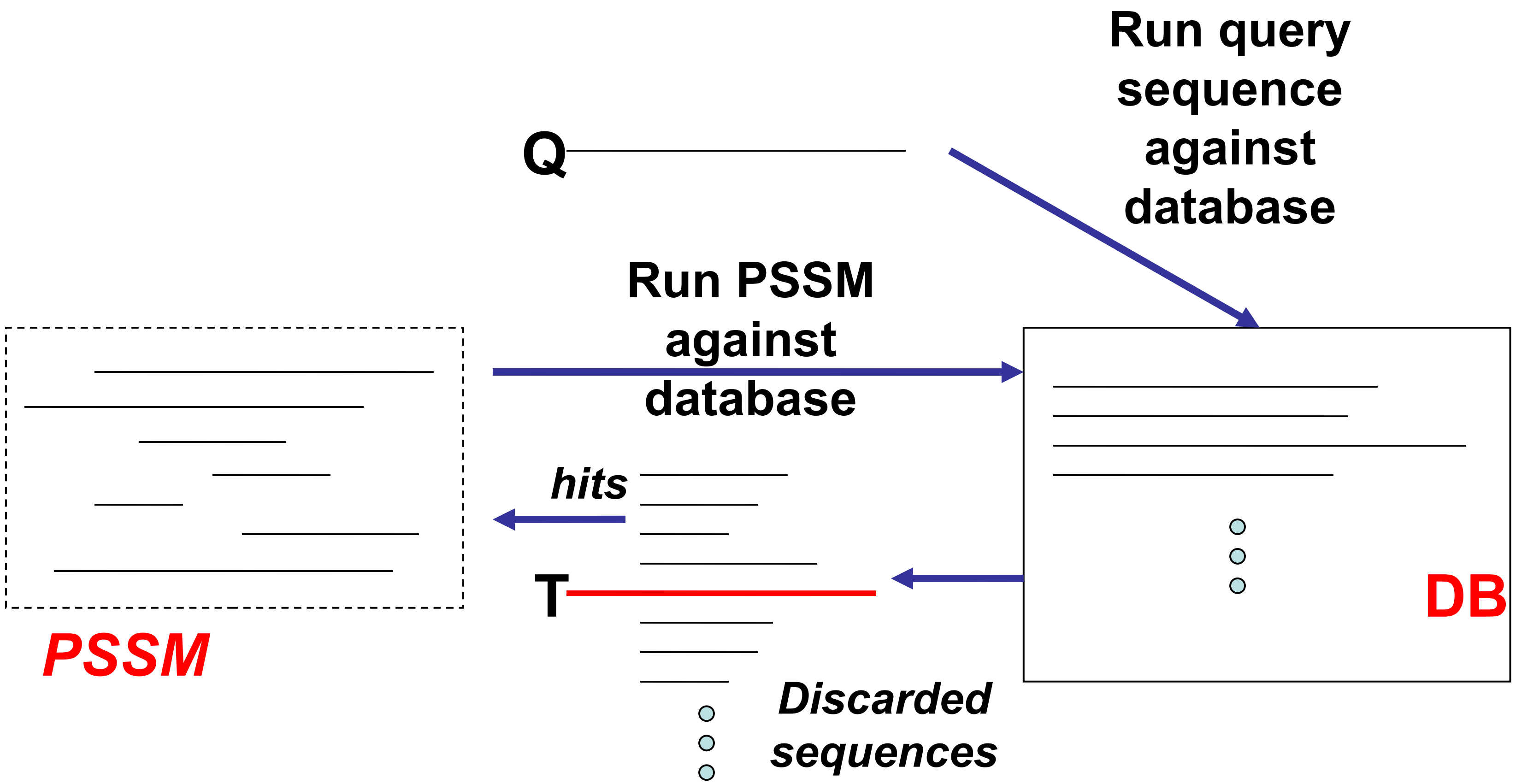
Gapped BLAST search

A C D . . Y

PSSM

Database hits

Another PSI-BLAST iteration graphic…



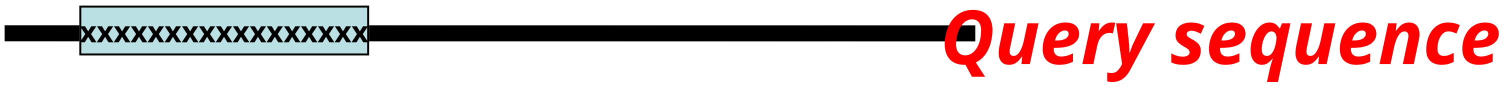
Run query sequence against database

Run PSSM against database

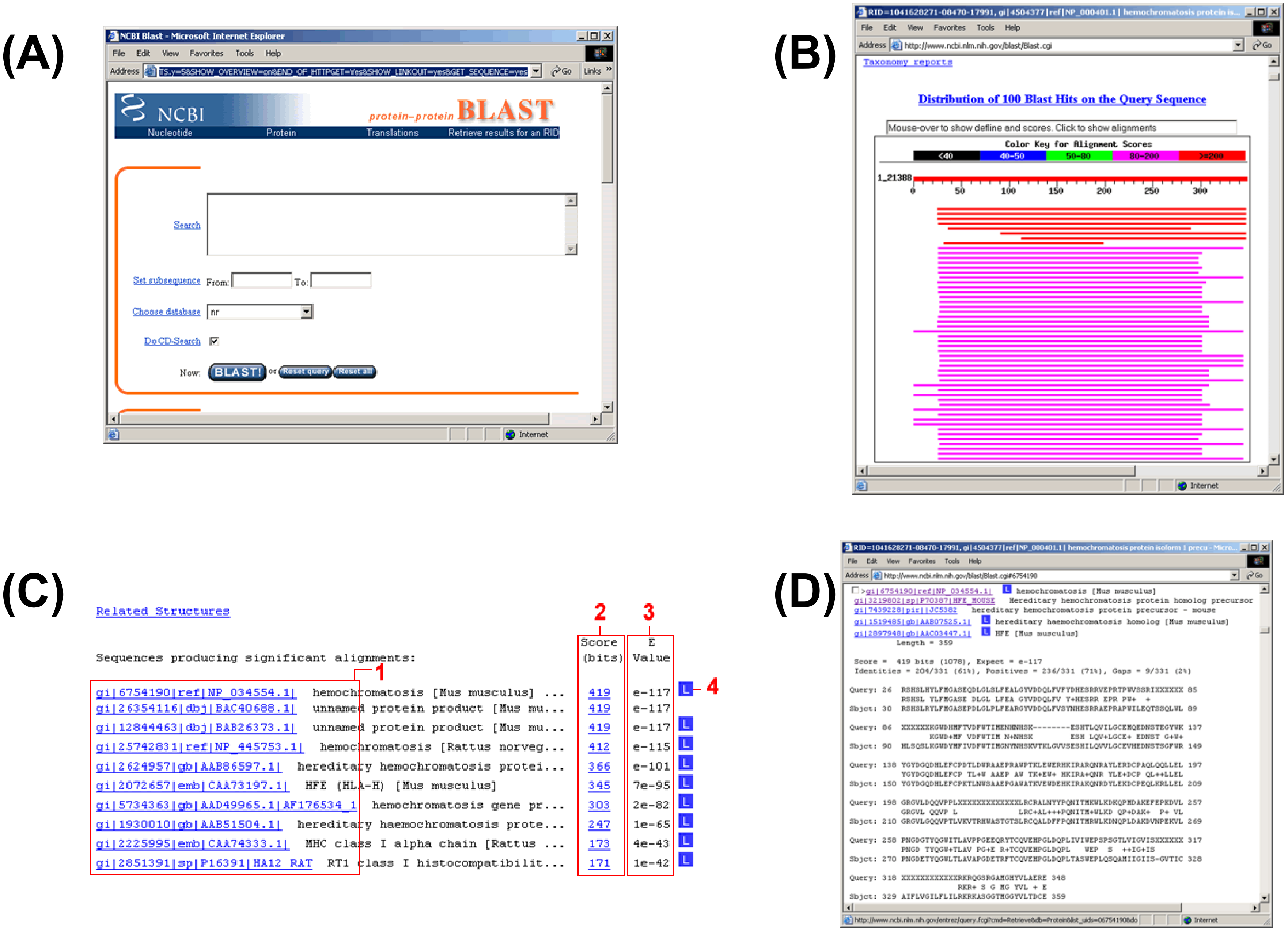
hits

Discarded sequences

Low-complexity sequence filtering



- For example: AAAAA… or AYLAYLAYL… or AYLLYAALY…
- Low-complexity (sub)sequences have a biased composition and contain less information than high-complexity sequences
- Because of the low information content, they often lead to spurious hits without a biological basis (for example, you can’t tell whether a poly-A sequence [AAAAA…] is more similar to a globin, an immunoglobulin or a kinase sequence)
- That is why BLAST filters low-complexity regions in the query sequence out
- See course book *Understanding Bioinformatics* for details



NCBI Blast - Microsoft Internet Explorer

Address: TS.y=5&SHOW_OVERVIEW=on&END_OF_HTTPGET=Yes&SHOW_LINKOUT=yes&GET_SEQUENCE=yes

Search

Set subsequence From: To:

Choose database nr

Do CD-Search ☑

Now: BLAST! or Reset query Reset all

RID=1041628271-08470-17991, gi|4504377|ref|NP_000401.1| hemochromatosis protein is...

Address: http://www.ncbi.nlm.nih.gov/blast/Blast.cgi

Taxonomy reports

Distribution of 100 Blast Hits on the Query Sequence

Mouse-over to show defline and scores. Click to show alignments

Color Key for Alignment Scores

<40	40-50	50-80	80-200	>=200

1_21388

0 50 100 150 200 250 300

```
Related Structures

                                                                     Score    E
Sequences producing significant alignments:                          (bits) Value

gi|6754190|ref|NP_034554.1|  hemochromatosis [Mus musculus] ...       419   e-117  L
gi|26354116|dbj|BAC40688.1|  unnamed protein product [Mus mu...       419   e-117
gi|12844463|dbj|BAB26373.1|  unnamed protein product [Mus mu...       419   e-117  L
gi|25742831|ref|NP_445753.1|  hemochromatosis [Rattus norveg...       412   e-115  L
gi|2624957|gb|AAB86597.1|  hereditary hemochromatosis protei...       366   e-101  L
gi|2072657|emb|CAA73197.1|  HFE (HLA-H) [Mus musculus]                345   7e-95  L
gi|5734363|gb|AAD49965.1|AF176534_1  hemochromatosis gene pr...       303   2e-82  L
gi|1930010|gb|AAB51504.1|  hereditary haemochromatosis prote...       247   1e-65  L
gi|2225995|emb|CAA74333.1|  MHC class I alpha chain [Rattus ...       173   4e-43  L
gi|2851391|sp|P16391|HA12_RAT  RT1 class I histocompatibilit...       171   1e-42  L
```

RID=1041628271-08470-17991, gi|4504377|ref|NP_000401.1| hemochromatosis protein isoform 1 precu - Micro...

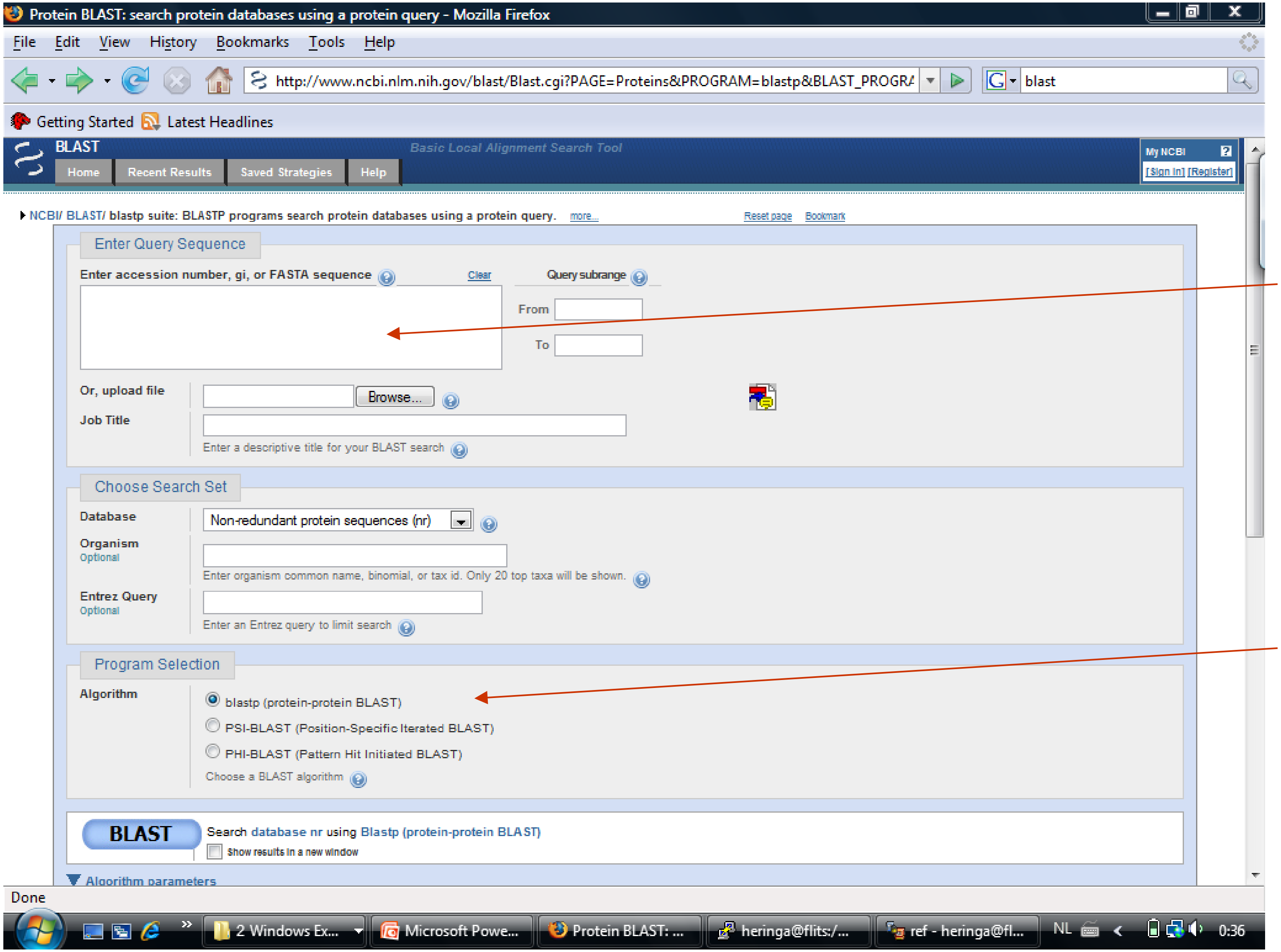
Address: http://www.ncbi.nlm.nih.gov/blast/Blast.cgi#6754190

```
□ >gi|6754190|ref|NP_034554.1| L hemochromatosis [Mus musculus]
 gi|3219802|sp|P70387|HFE_MOUSE  Hereditary hemochromatosis protein homolog precursor
 gi|7439228|pir||JC5382  hereditary hemochromatosis protein precursor - mouse
 gi|1519485|gb|AAB07525.1| L hereditary haemochromatosis homolog [Mus musculus]
 gi|2897948|gb|AAC03447.1| L HFE [Mus musculus]
          Length = 359

 Score =  419 bits (1078), Expect = e-117
 Identities = 204/331 (61%), Positives = 236/331 (71%), Gaps = 9/331 (2%)

Query: 26  RSHSLHYLFMGASEQDLGLSLFEALGYVDDQLFVFYDHESRRVEPRTPWVSSRIXXXXXX 85
            RSHSL YLFMGASE DLGL LFEA GYVDDQLFV Y+HESRR EPR PW+  +
Sbjct: 30  RSHSLRYLFMGASEPDLGLPLFEARGYVDDQLFVSYNHESRRAEPRAPWILEQTSSQLWL 89

Query: 86  XXXXXXKGWDHMFTVDFWTIMENHNHSK--------ESHTLQVILGCEMQEDNSTEGYWK 137
                  KGWD+MF VDFWTIM N+NHSK        ESH LQV+LGCE+ EDNST G+W+
Sbjct: 90  HLSQSLKGWDYMFIVDFWTIMGNYNHSKVTKLGVVSESHILQVVLGCEVHEDNSTSGFWR 149

Query: 138 YGYDGQDHLEFCPDTLDWRAAEPRAWPTKLEWERHKIRARQNRAYLERDCPAQLQQLLEL 197
            YGYDGQDHLEFCP TL+W AAEP AW TK+EW+ HKIRA+QNR YLE+DCP QL++LLEL
Sbjct: 150 YGYDGQDHLEFCPKTLNWSAAEPGAWATKVEWDEHKIRAKQNRDYLEKDCPEQLKRLLEL 209

Query: 198 GRGVLDQQVPPLXXXXXXXXXXXXXLRCRALNYYPQNITMKWLKDKQPMDAKEFEPKDVL 257
            GRGVL QQVP L             LRC+AL+++PQNITM+WLKD QP+DAK+  P+ VL
Sbjct: 210 GRGVLGQQVPTLVKVTRHWASTGTSLRCQALDFFPQNITMRWLKDNQPLDAKDVNPEKVL 269

Query: 258 PNGDGTYQGWITLAVPPGEEQRYTCQVEHPGLDQPLIVIWEPSPSGTLVIGVISXXXXXX 317
            PNGD TYQGW+TLAV PG+E R+TCQVEHPGLDQPL   WEP  S  ++IG+IS
Sbjct: 270 PNGDETYQGWLTLAVAPGDETRFTCQVEHPGLDQPLTASWEPLQSQAMIIGIIS-GVTIC 328

Query: 318 XXXXXXXXXXXXRKRQGSRGAMGHYVLAERE 348
                        RKR+ S G MG YVL + E
Sbjct: 329 AIFLVGILFLILRKRKASGGTMGGYVLTDCE 359
```

http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=Protein&list_uids=06754190&do

Figure 6

PSI-BLAST entry page



http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?PAGE=Proteins&PROGRAM=blastp&BLAST_PROGRA

NCBI/ BLAST/ blastp suite: BLASTP programs search protein databases using a protein query.

Enter Query Sequence

Enter accession number, gi, or FASTA sequence

Query subrange

From

To

Or, upload file

Job Title

Enter a descriptive title for your BLAST search

Choose Search Set

Database: Non-redundant protein sequences (nr)

Organism
Optional

Enter organism common name, binomial, or tax id. Only 20 top taxa will be shown.

Entrez Query
Optional

Enter an Entrez query to limit search

Program Selection

Algorithm

- ● blastp (protein-protein BLAST)
- ○ PSI-BLAST (Position-Specific Iterated BLAST)
- ○ PHI-BLAST (Pattern Hit Initiated BLAST)

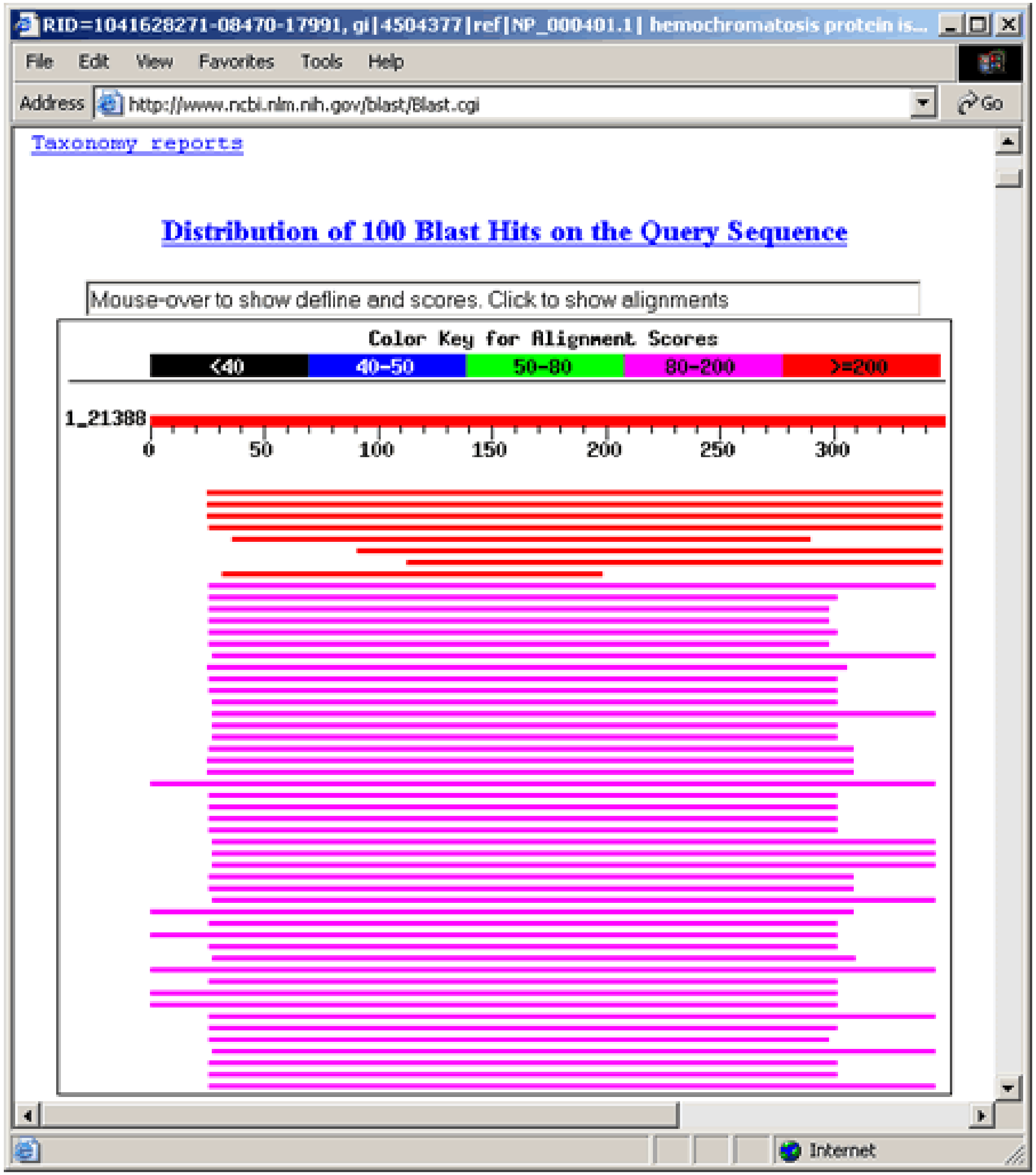
Choose a BLAST algorithm

Search database nr using Blastp (protein-protein BLAST)

□ Show results in a new window

Paste your query sequence

Choose the BLAST program you want



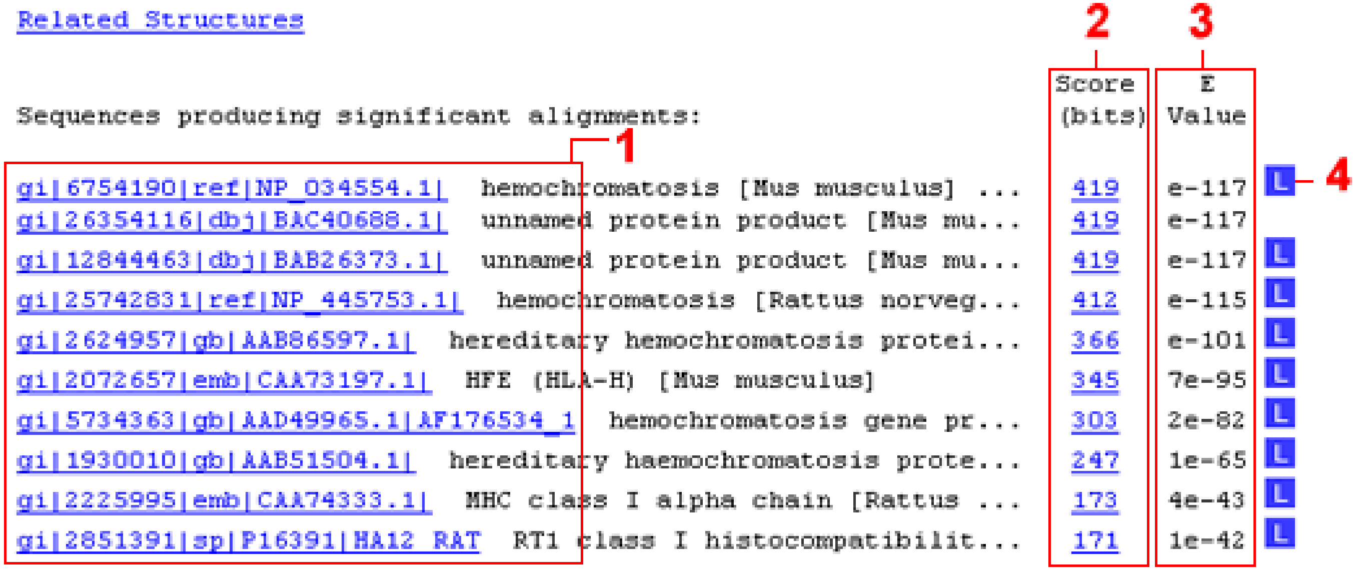
RID=1041628271-08470-17991, gi|4504377|ref|NP_000401.1| hemochromatosis protein is...

Address http://www.ncbi.nlm.nih.gov/blast/Blast.cgi

Taxonomy reports

Distribution of 100 Blast Hits on the Query Sequence

Mouse-over to show defline and scores. Click to show alignments



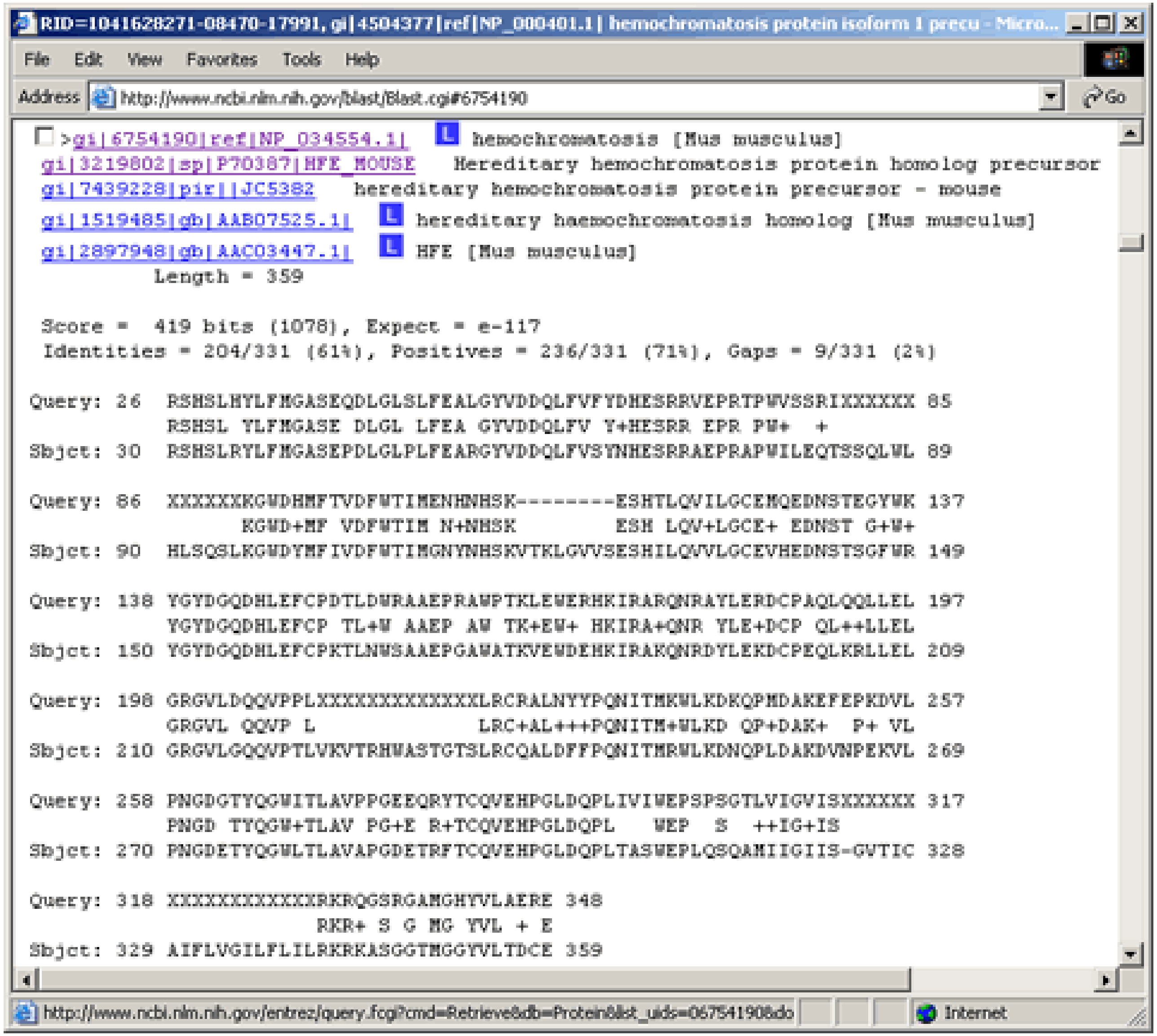
```
Related Structures

                                                              Score    E
Sequences producing significant alignments:                  (bits)  Value

gi|6754190|ref|NP_034554.1|  hemochromatosis [Mus musculus] ...   419   e-117  L
gi|26354116|dbj|BAC40688.1|  unnamed protein product [Mus mu...   419   e-117
gi|12844463|dbj|BAB26373.1|  unnamed protein product [Mus mu...   419   e-117  L
gi|25742831|ref|NP_445753.1|  hemochromatosis [Rattus norveg...   412   e-115  L
gi|2624957|gb|AAB86597.1|  hereditary hemochromatosis protei...   366   e-101  L
gi|2072657|emb|CAA73197.1|  HFE (HLA-H) [Mus musculus]            345   7e-95  L
gi|5734363|gb|AAD49965.1|AF176534_1  hemochromatosis gene pr...   303   2e-82  L
gi|1930010|gb|AAB51504.1|  hereditary haemochromatosis prote...   247   1e-65  L
gi|2225995|emb|CAA74333.1|  MHC class I alpha chain [Rattus ...   173   4e-43  L
gi|2851391|sp|P16391|HA12_RAT  RT1 class I histocompatibilit...   171   1e-42  L
```

1 - This portion of each description links to the sequence record for a particular hit.

2 - Score or bit score is a value calculated from the number of gaps and substitutions associated with each aligned sequence. The higher the score, the more significant the alignment. Each score links to the corresponding pairwise alignment between query sequence and hit sequence (also referred to as subject sequence).

3 - E Value (Expect Value) describes the likelihood that a sequence with a similar score will occur in the database by chance. The smaller the E Value, the more significant the alignment. For example, the first alignment has a very low E value of e^{-117} meaning that a sequence with a similar score is very unlikely to occur simply by chance.

4 - These links provide the user with direct access from BLAST results to related entries in other databases. ‘L’ links to LocusLink records and ‘S’ links to structure records in NCBI's Molecular Modeling DataBase.



RID=1041628271-08470-17991, gi|4504377|ref|NP_000401.1| hemochromatosis protein isoform 1 precu - Micro...
File Edit View Favorites Tools Help
Address http://www.ncbi.nlm.nih.gov/blast/Blast.cgi#6754190
Go
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=Protein&list_uids=06754190&do
Internet

```
□ >gi|6754190|ref|NP_034554.1|  L  hemochromatosis [Mus musculus]
 gi|3219802|sp|P70387|HFE_MOUSE   Hereditary hemochromatosis protein homolog precursor
 gi|7439228|pir||JC5382   hereditary hemochromatosis protein precursor - mouse
 gi|1519485|gb|AAB07525.1|  L  hereditary haemochromatosis homolog [Mus musculus]
 gi|2897948|gb|AAC03447.1|  L  HFE [Mus musculus]
          Length = 359

 Score =  419 bits (1078), Expect = e-117
 Identities = 204/331 (61%), Positives = 236/331 (71%), Gaps = 9/331 (2%)

Query: 26  RSHSLHYLFMGASEQDLGLSLFEALGYVDDQLFVFYDHESRRVEPRTPWVSSRIXXXXXX 85
            RSHSL YLFMGASE DLGL LFEA GYVDDQLFV Y+HESRR EPR PW+   +
Sbjct: 30  RSHSLRYLFMGASEPDLGLPLFEARGYVDDQLFVSYNHESRRAEPRAPWILEQTSSQLWL 89

Query: 86  XXXXXXKGWDHMFTVDFWTIMENHNHSK--------ESHTLQVILGCEMQEDNSTEGYWK 137
                  KGWD+MF VDFWTIM N+NHSK        ESH LQV+LGCE+ EDNST G+W+
Sbjct: 90  HLSQSLKGWDYMFIVDFWTIMGNYNHSKVTKLGVVSESHILQVVLGCEVHEDNSTSGFWR 149

Query: 138 YGYDGQDHLEFCPDTLDWRAAEPRAWPTKLEWERHKIRARQNRAYLERDCPAQLQQLLEL 197
            YGYDGQDHLEFCP TL+W AAEP AW TK+EW+ HKIRA+QNR YLE+DCP QL++LLEL
Sbjct: 150 YGYDGQDHLEFCPKTLNWSAAEPGAWATKVEWDEHKIRAKQNRDYLEKDCPEQLKRLLEL 209

Query: 198 GRGVLDQQVPPLXXXXXXXXXXXXXLRCRALNYYPQNITMKWLKDKQPMDAKEFEPKDVL 257
            GRGVL QQVP L             LRC+AL+++PQNITM+WLKD QP+DAK+  P+ VL
Sbjct: 210 GRGVLGQQVPTLVKVTRHWASTGTSLRCQALDFFPQNITMRWLKDNQPLDAKDVNPEKVL 269

Query: 258 PNGDGTYQGWITLAVPPGEEQRYTCQVEHPGLDQPLIVIWEPSPSGTLVIGVISXXXXXX 317
            PNGD TYQGW+TLAV PG+E R+TCQVEHPGLDQPL   WEP  S  ++IG+IS
Sbjct: 270 PNGDETYQGWLTLAVAPGDETRFTCQVEHPGLDQPLTASWEPLQSQAMIIGIIS-GVTIC 328

Query: 318 XXXXXXXXXXXXRKRQGSRGAMGHYVLAERE 348
                        RKR+ S G MG YVL + E
Sbjct: 329 AIFLVGILFLILRKRKASGGTMGGYVLTDCE 359
```

‘X’ residues denote low-complexity sequence fragments that are ignored

Recap

- Databases grow exponentially, sequence databases even more so than others (e.g. structural or functional DBs)
- Homology searching: predicting function by sequence database searching using the homology principle
- BLAST
  - Original BLAST (not used anymore)
  - Two-hit method and gapped BLAST
  - Extensions of Blast (indexing the sequence database)
- PSI-BLAST
  - Calculating the PSSM
  - Pseudo-counts
  - Low-complexity subsequences
  - Iteration
- FASTA
  - Base-20 hashing

BELOW SLIDES ARE FOR REFERENCE

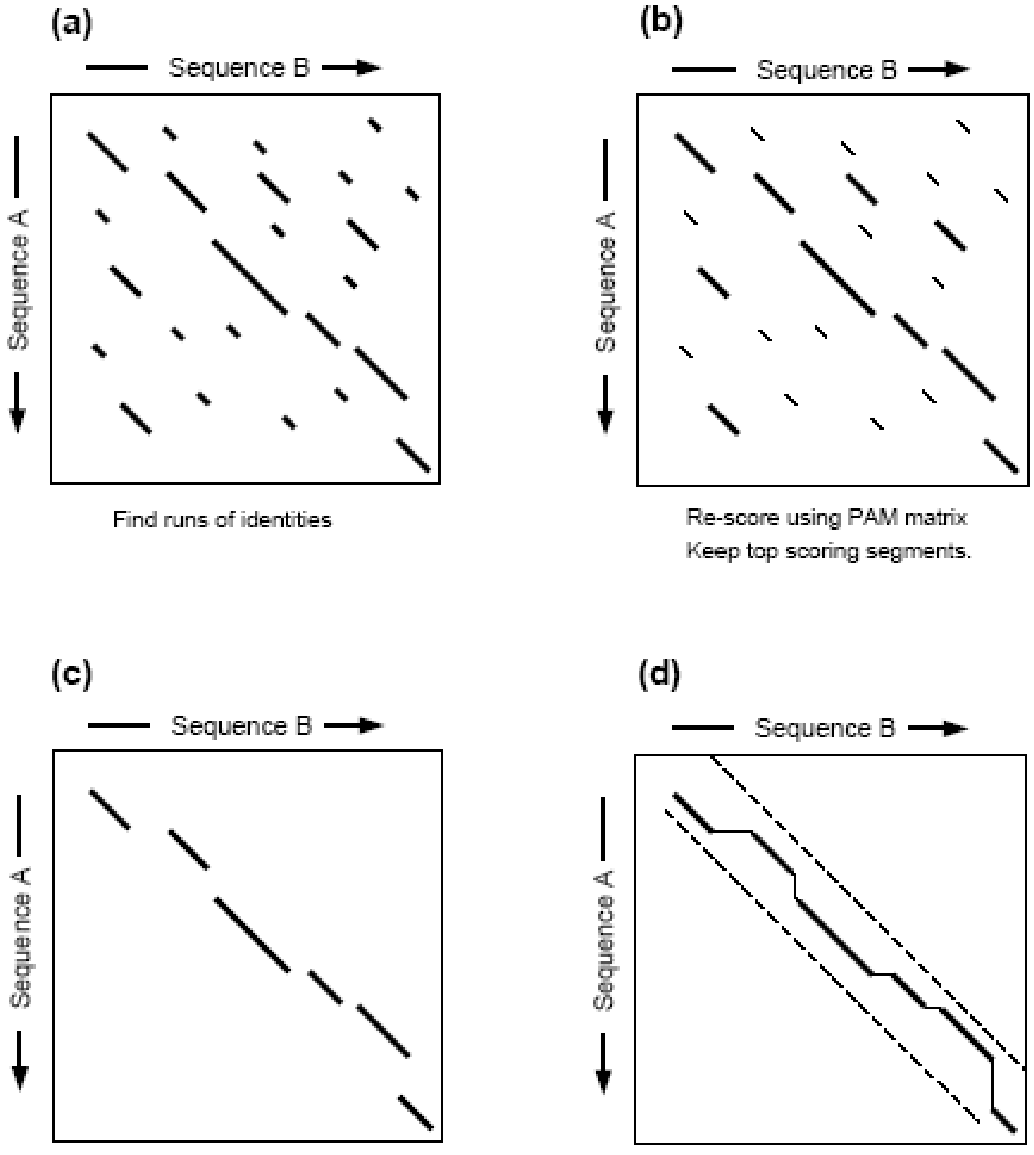
NOT COVERED IN LECTURE

NOT EXAM MATERIAL

A piece of history: FASTA

- First homology searching method (FASTP – Lipman & Pearson, 1985)
- Until gapped-BLAST and PSI-BLAST appeared (1997), FASTA has in fact been the better method
    - Unlike the impression created by the BLAST folks (from 1990)
- Compares a given query sequence with a library of sequences and calculates for each pair the highest scoring local alignment
- Similar to BLAST, speed is obtained by delaying application of the dynamic programming technique to the moment where the most similar segments are already identified by faster and less sensitive techniques
- FASTA routine operates in four steps:

FASTA Algorithm



Step 1

Find runs of identities

Step 2

Re-score using PAM matrix
Keep top scoring segments.

Step 3

Apply "joining threshold"
to eliminate segments that
are unlikely to be part of the alignment
that includes highest scoring segment.

Step 4

Use dynamic programming
to optimise the alignment in a
narrow band that encompasses
the top scoring segments.

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 target sequence the 10 regions with highest density of ungapped 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 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 step is slow – $O(n^2)$

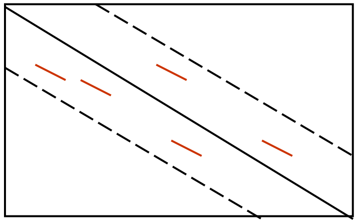
FASTA output example

```
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 how to make step 1 fast

(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 (continued)

- Preprocessing: a lookup table is constructed for all words in the query sequence, which is then used to compare all encountered words in every database sequence
- **General example of hashing (old school)**: the telephone book to find persons' phone numbers (names are ordered)
  - Using a phone book, you do not need to search through all names until you find the person you want
  - In computer speak: find a hash function *f* such that *f*(*name*) can be directly assigned to an address in the computer where the telephone number is stored
- Hashing of sequence information is typically based on ***k*-mers** (that are often alpha-numerically ordered)

Hashing - examples

- *T9 Predictive Text* in first-generation mobile phones
  - “hello”:

    4, 4, 3, 3, 5, 5, 5,
    (*pause*) 5, 5, 5, 6, 6, 6
  - “hello” in T9:

    4, 3, 5, 5, 6
  - Collisions: 4, 6:

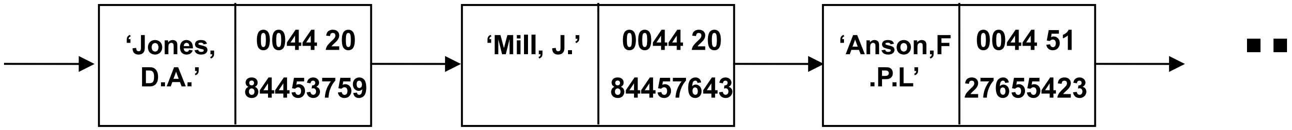
    “in”, “go”

Hashing - examples (cont..)

- *Other easier hash function: let a=1, b=2, c=3, etc.*
    - “hello” now gets hash address 8+5+12+12+15 = 52
    - “olleh” will get same address (collision)
    - Each word encountered gets a hash address immediately and can be indexed.
    - How good is this hash function?

HASHING (cont.)

This takes too long……



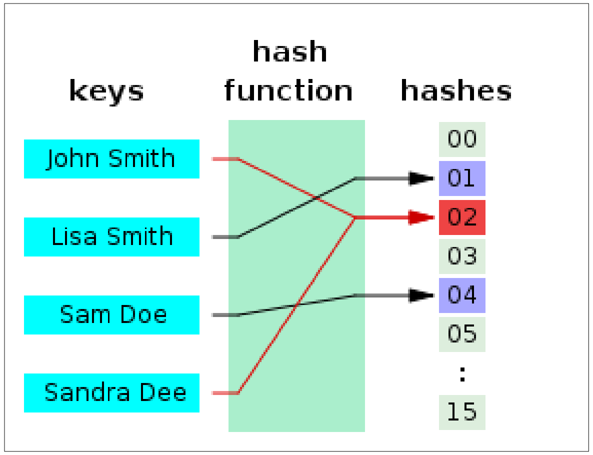
‘Jones, D.A.’ 0044 20 84453759

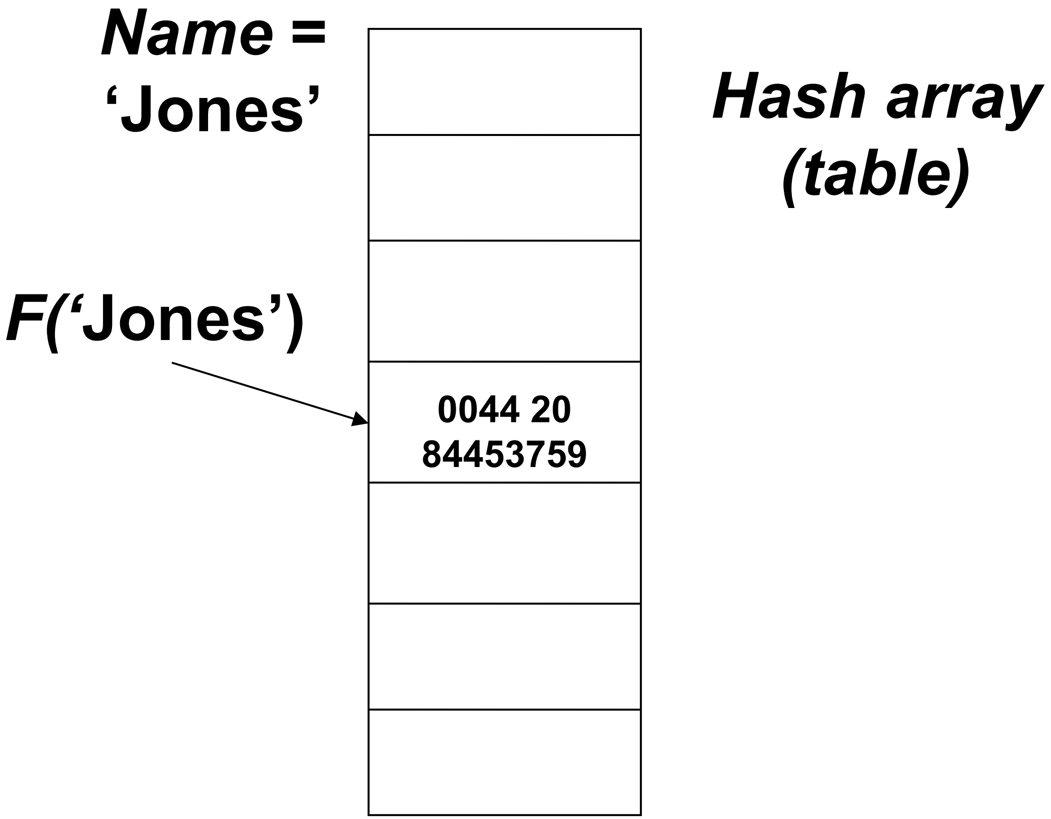
‘Mill, J.’ 0044 20 84457643

‘Anson,F.P.L’ 0044 51 27655423

..

HASHING (cont.)





Name = **‘Jones’**

F(‘Jones’)

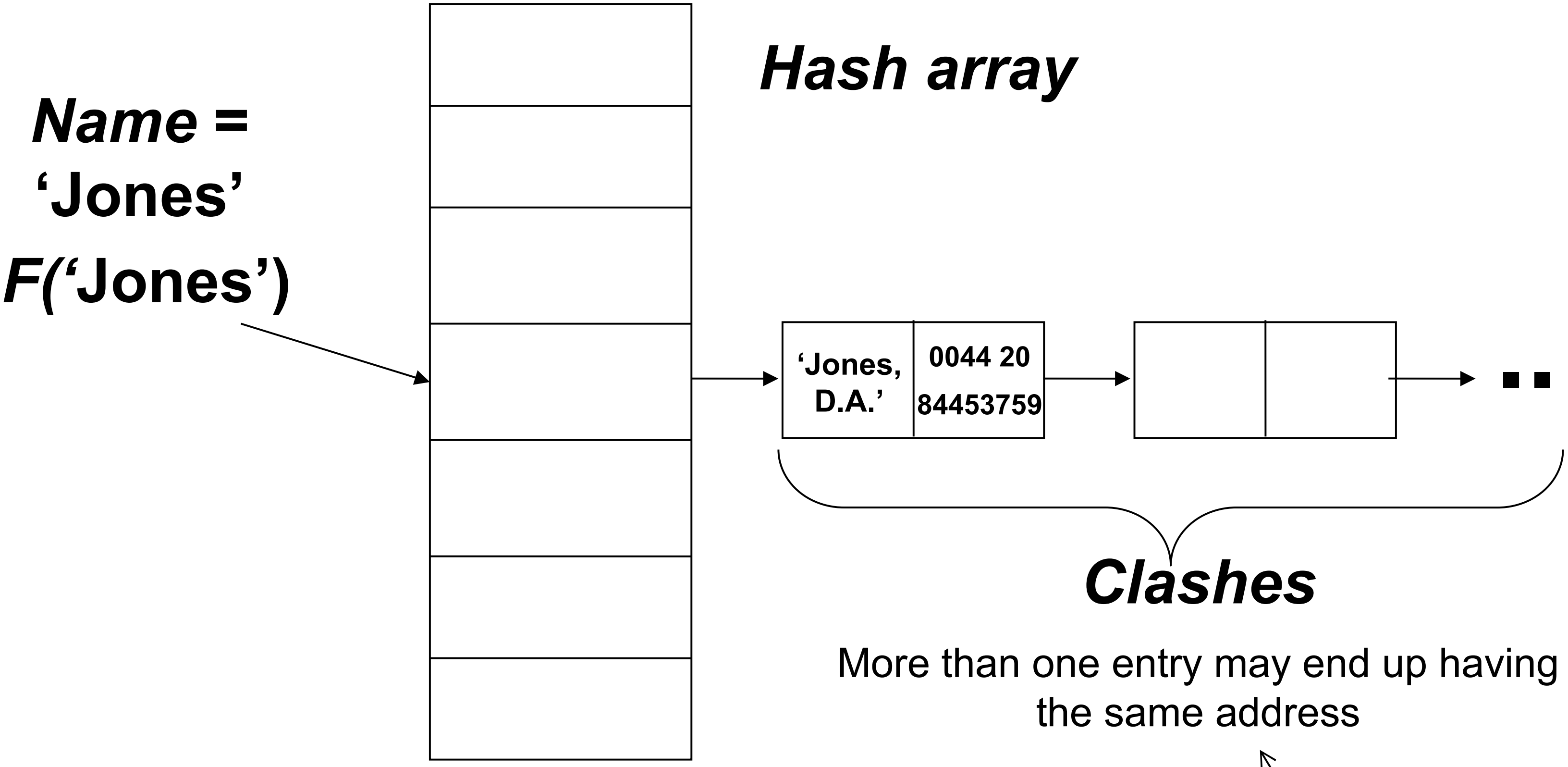
0044 20 84453759

Hash array (table)

For sequences:

- name is *subword* in database sequence
- telephone number is sequence position(s) of subword

HASHING (cont.)



More than one entry may end up having the same address

Secondary hash function will solve searching though clash list

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

Query: ERLFERLACER ....

DB: MERIFERLACACTR ....

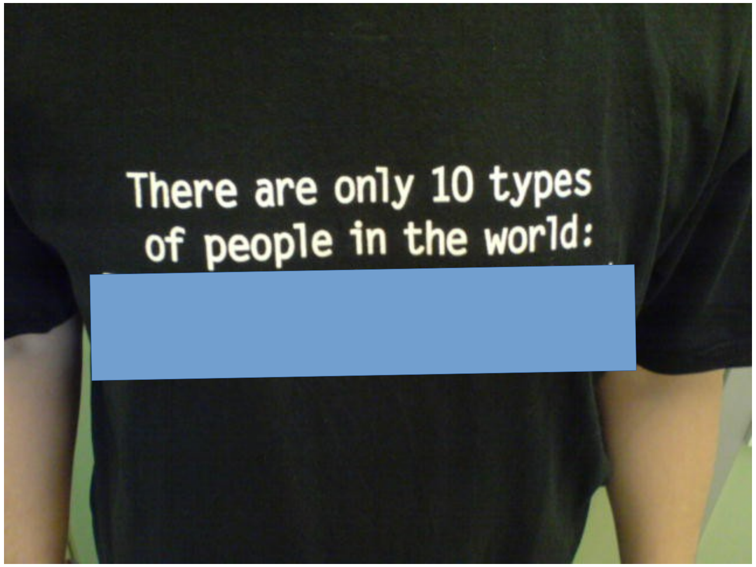
Query hash table:

Word	*Position*
ER	1, 5, 10
RL	2, 6
LF	3
FE	4
LA	7
AC	8
CE	9
….	…

You only need to go through the DB sequence once: for each word encountered (ER, RL, LF, ..), 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 right away (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)).

Number systems



Base-1: binary
0-1
Base-8: octal
0-7
Base-10: decimal 0-9
Base-16: hexadecimal 0-F

Base 20: vigesimal → FASTA

(Decimal) 503 = $5*10^2 + 0*10^1 + 3*10^0$

(Base-20) 213 = $2*20^2 + 1*20^1 + 3*20^0$ = 823 (decimal)

The FASTA implementation

Query:

ERLFERLACER
12345678911
01

One-let codes: ACDEFGHIKLMNPQRSTVWY
0 5 10 15

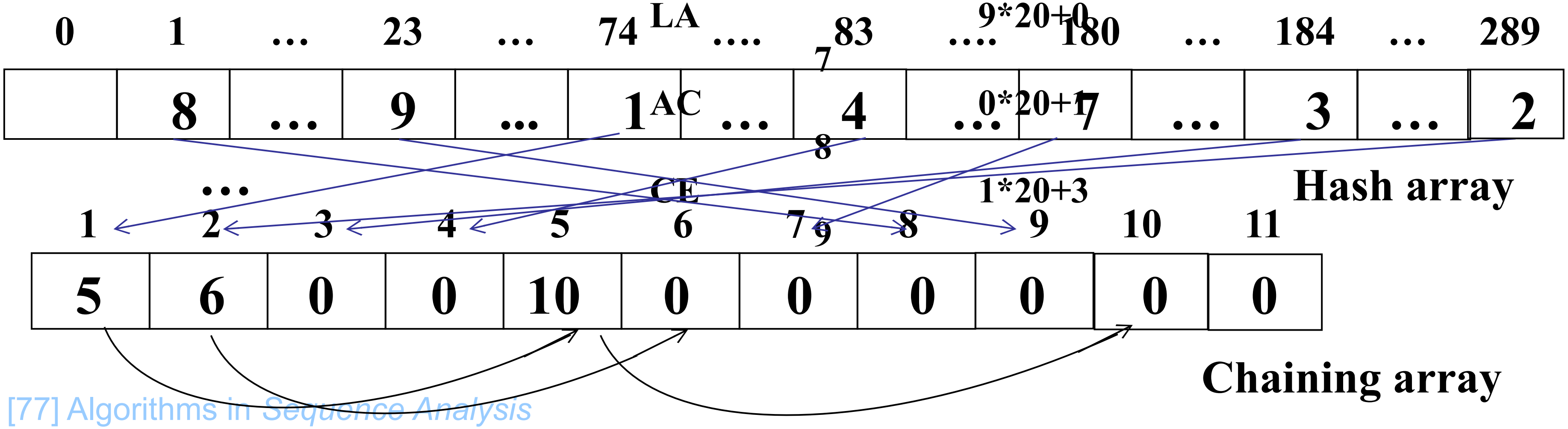
Dipeptides are hashed using vigesimal (base-20) numbering scheme

Using the FASTA hash/chaining array setup, one can run along the dipeptides of a database sequence (linear search) and know instantaneously where the identical dipeptides in the query sequence are positioned.

Query hash table:

Word	*Position*	*Hash*
ER	1, 5, 10	3*20+14
RL	2, 6	14*20+9
LF	3	9*20+4
FE	4	4*20+3
LA	7	9*20+0
AC	8	0*20+1
CE	9	1*20+3

E=3, R=14
h(‘ER’)=3×20+14=74



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, the database search.