# Sequence Alignment

Marco Botta
Dipartimento di Informatica
Università di Torino
botta@di.unito.it
www.di.unito.it/~botta/didattica/



## Sequence Comparison

Much of bioinformatics involves sequences

- DNA sequences
- RNA sequences
- Protein sequences

We can think of these sequences as strings of letters

- DNA & RNA: alphabet of 4 letters
- Protein: alphabet of 20 letters

3

## Sequence Comparison (cont)

- Finding similarity between sequences is important for many biological questions

For example:

- Find genes/proteins with common origin
  - Allows to predict function & structure
- Locate common subsequences in genes/proteins
  - Identify common “motifs”
- Locate sequences that might overlap
  - Help in sequence assembly

4

## Sequence Alignment

**Input:** two sequences over the same alphabet

**Output:** an **alignment** of the two sequences

Example:

- GCGCATGGATTGAGCGA
- TGCGCCATTGATGACCA

A possible alignment:

```
-GCGC-ATGGATTGAGCGA
TGCGCCATTGAT-GACC-A
```

## Alignments

```
-GCGC-ATGGATTGAGCGA
TGCGCCATTGAT-GACC-A
```

Three elements:

- Perfect matches
- Mismatches
- Insertions & deletions (**indel**)



## Choosing Alignments

There are many possible alignments

For example, compare:

```
-GCGC-ATGGATTGAGCGA
TGCGCCATTGAT-GACC-A
```

to

```
------GCGCATGGATTGAGCGA
TGCGCC----ATTGATGACCA--
```

Which one is better?

7

## Scoring Alignments

**Rough intuition:**

- Similar sequences evolved from a common ancestor
- Evolution changed the sequences from this ancestral sequence by **mutations:**
  - **Replacements**: one letter replaced by another
  - **Deletion:** deletion of a letter
  - **Insertion:** insertion of a letter
- Scoring of sequence similarity should examine how many operations took place

8

## Simple Scoring Rule

Score each position independently:

- Match: +1
- Mismatch: -1
- Indel -2

Score of an alignment is sum of positional scores

## Example

Example:

```
-GCGC-ATGGATTGAGCGA
TGCGCCATTGAT-GACC-A
```

Score: (+1x13) + (-1x2) + (-2x4) = 3

```
------GCGCATGGATTGAGCGA
TGCGCC----ATTGATGACCA--
```

Score: (+1x5) + (-1x6) + (-2x12) = -25



## More General Scores

- The choice of +1,-1, and -2 scores was quite arbitrary
- Depending on the context, some changes are more plausible than others
  - Exchange of an amino-acid by one with similar properties (size, charge, etc.)

  vs.
  - Exchange of an amino-acid by one with opposite properties

## Additive Scoring Rules

- We define a scoring function by specifying a function

$$\sigma : (\mathrm{A} \cup \{-\}) \times (\mathrm{A} \cup \{-\}) \mapsto \Re$$

$$\mathrm{A} = \{a, c, g, t\}$$

  - $\sigma(x,y)$ is the score of replacing $x$ by $y$
  - $\sigma(x,-)$ is the score of deleting $x$
  - $\sigma(-,x)$ is the score of inserting $x$
- The score of an alignment is the sum of position scores

$$\sum_{i=1}^{n} \sigma_i(x, y)$$



## Edit Distance

- The **edit distance** between two sequences is the "cost" of the "cheapest" set of edit operations needed to transform one sequence into the other

$$d(s_1, s_2) = \max_{\text{alignment of } s_1 \& s_2} \text{score(alignment)}$$

- Computing edit distance between two sequences almost equivalent to finding the alignment that minimizes the distance

13

## Computing Edit Distance

- How can we compute the edit distance??
  - If $|s| = n$ and $|t| = m$, there are more than $\binom{m+n}{m}$ alignments
- The additive form of the score allows to perform **dynamic programming** to compute edit distance efficiently

14

- (m+n)(m+n-1)(m+n-2)…1
- m(m-1)(m-2)…(m-n).. n(n-1)(n-2)…1

15

## Recursive Argument

- Suppose we have two sequences: *s[1..n+1]* and *t[1..m+1]*

The best alignment must be in one of three cases:

1. Last position is (*s[n+1],t[m +1]*)
2. Last position is (*s[n +1]*, -)
3. Last position is (-, *t[m +1]*)

$$d(s[1..n+1],t[1..m+1]) = d(s[1..,n],t[1..m]) + \sigma(s[n+1],t[m+1])$$

16

## Recursive Argument

- Suppose we have two sequences: *s[1..n+1]* and *t[1..m+1]*

The best alignment must be in one of three cases:

1. Last position is (*s[n+1],t[m +1]*)
2. Last position is (*s[n +1]*, -)
3. Last position is (-, *t[m +1]*)

$$d(s[1..n+1],t[1..m+1]) = d(s[1..,n],t[1..m+1]) + \sigma(s[n+1],-)$$

17

## Recursive Argument

- Suppose we have two sequences: *s[1..n+1]* and *t[1..m+1]*

The best alignment must be in one of three cases:

1. Last position is (*s[n+1]*,*t[m +1]*)
2. Last position is (*s[n +1]*, -)
3. Last position is (-, *t[m +1]*)

$$d(s[1..n+1],t[1..m+1]) = d(s[1..,n+1],t[1..m]) + \sigma(-,t[m+1])$$

18

## Recursive Argument

Define the notation:

$$V[i,j] = d(s[1..i],t[1..j])$$

- Using the recursive argument, we get the following recurrence for *V*:

$$V[i+1,j+1] = \max\begin{pmatrix} V[i,j] + \sigma(s[i+1],t[j+1]) \\ V[i,j+1] + \sigma(s[i+1],-) \\ V[i+1,j] + \sigma(-,t[j+1]) \end{pmatrix}$$

## Recursive Argument

- Of course, we also need to handle the base cases in the recursion:

$$V[0,0] = 0$$
$$V[i+1,0] = V[i,0] + \sigma(s[i+1],-)$$
$$V[0,j+1] = V[0,j] + \sigma(-,t[j+1])$$



## Dynamic Programming Algorithm

| | | A | G | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | | | |
| A 1 | | | | |
| A 2 | | | | |
| A 3 | | | | |
| C 4 | | | | |

We fill the matrix using the recurrence rule

21

## Dynamic Programming Algorithm

| | | A | G | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | | | |
| A 1 | | | | |
| A 2 | | | | |
| A 3 | | | | |
| C 4 | | | | |

Conclusion: d(AAAC,AGC) = -1

22

## Reconstructing the Best Alignment

- To reconstruct the best alignment, we record which case in the recursive rule maximized the score

| | | A | G | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | -2 | -4 | -6 |
| A 1 | -2 | 1 | -1 | -3 |
| A 2 | -4 | -1 | 0 | -2 |
| A 3 | -6 | -3 | -2 | -1 |
| C 4 | -8 | -5 | -4 | -1 |

23

## Reconstructing the Best Alignment

- We now trace back the path that corresponds to the best alignment

```
AAAC
AG-C
```

| | | A | G | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | -2 | -4 | -6 |
| A 1 | -2 | 1 | -1 | -3 |
| A 2 | -4 | -1 | 0 | -2 |
| A 3 | -6 | -3 | -2 | -1 |
| C 4 | -8 | -5 | -4 | -1 |

24

## Reconstructing the Best Alignment

- Sometimes, more than one alignment has the best score

```
AAAC
A-GC
```

| | | A | G | C |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | 0 | -2 | -4 | -6 |
| A 1 | -2 | 1 | -1 | -3 |
| A 2 | -4 | -1 | 0 | -2 |
| A 3 | -6 | -3 | -2 | -1 |
| C 4 | -8 | -5 | -4 | -1 |

## Complexity

Space: $O(mn)$

Time: $O(mn)$

- Filling the matrix $O(mn)$
- Backtrace $O(m+n)$



## Local Alignment

Consider now a different question:

- Can we find similar substring of $s$ and $t$?
- Formally, given $s[1..n]$ and $t[1..m]$ find $i,j,k,$ and $l$ such that $d(s[i..j],t[k..l])$ is maximal

27

## Local Alignment

- As before, we use dynamic programming
- We now want to set $V[i,j]$ to record the best alignment of a **suffix** of $s[1..i]$ and a **suffix** of $t[1..j]$
- How should we change the recurrence rule?

28

## Local Alignment

New option:

- We can start a new match instead of extend previous alignment

$$V[i+1,j+1] = \max \begin{pmatrix} V[i,j] + \sigma(s[i+1],t[j+1]) \\ V[i,j+1] + \sigma(s[i+1],-) \\ V[i+1,j] + \sigma(-,t[j+1]) \\ 0 \end{pmatrix}$$

Alignment of empty suffixes

29

## Local Alignment

- Again, we also need to handle the base cases in the recursion:

$$V[0,0] = 0$$

$$V[i+1,0] = \max(0, V[i,0] + \sigma(s[i+1], -))$$

$$V[0,j+1] = \max(0, V[0,j] + \sigma(-, t[j+1]))$$

30

## Local Alignment Example

$s$ = TAATA
$t$ = ATCTAA

| | | A | T | C | T | A | A |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 0 | | | | | | |
| T | 1 | | | | | | |
| A | 2 | | | | | | |
| A | 3 | | | | | | |
| T | 4 | | | | | | |
| A | 5 | | | | | | |

## Local Alignment Example

*s* = TAATA
*t* = TACTAA

| | 0 | T 1 | A 2 | C 3 | T 4 | A 5 | A 6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **T 1** | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **A 2** | 0 | 0 | 2 | 0 | 0 | 2 | 1 |
| **A 3** | 0 | 0 | 1 | 1 | 0 | 1 | 3 |
| **T 4** | 0 | 0 | 0 | 0 | 2 | 0 | 1 |
| **A 5** | 0 | 0 | 1 | 0 | 0 | 3 | 1 |



## Local Alignment Example

*s* = **TAATA**
*t* = **TACTA**A

| | 0 | T 1 | A 2 | C 3 | T 4 | A 5 | A 6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **T 1** | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **A 2** | 0 | 0 | 2 | 0 | 0 | 2 | 1 |
| **A 3** | 0 | 0 | 1 | 1 | 0 | 1 | 3 |
| **T 4** | 0 | 1 | 0 | 0 | 2 | 0 | 1 |
| **A 5** | 0 | 0 | 2 | 0 | 0 | 3 | 1 |

## Local Alignment Example

*s* = **TAA**TA
*t* = TAC**TAA**

| | 0 | T 1 | A 2 | C 3 | T 4 | A 5 | A 6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **T 1** | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| **A 2** | 0 | 0 | 2 | 0 | 0 | 2 | 1 |
| **A 3** | 0 | 0 | 1 | 1 | 0 | 1 | 3 |
| **T 4** | 0 | 1 | 0 | 0 | 2 | 0 | 1 |
| **A 5** | 0 | 0 | 2 | 0 | 0 | 3 | 1 |



## Sequence Alignment

We have seen two variants of sequence alignment:

- Global alignment
- Local alignment

Other variants:

- Finding best overlap (exercise)

All are based on the same basic idea of dynamic programming

35

## Alignment with Gaps

```
I   AAC-AATTAAG-ACTAC-GTTCATGAC
    A-CGA-TTA-GCAC-ACTG-T-C-GA-

II  AACAATTAAGACTACGTTCATGAC---
    AACAATT--------GTTCATGACGCA
```

36

## Gaps

- Both alignments have the same number of matches and spaces but…alignment II seems better.
- Definition: A ***gap*** is any maximal, consecutive run of spaces in a single string.
- The *length of the gap* will be the number of spaces in it.
- Example I has 11 gaps while example II has only 2 gaps.
- Idea: develop alignment scores that take gaps (not spaces) into account.

37

## Biological Motivation

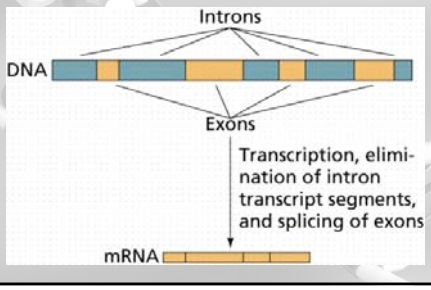
- Number of mutational events
  - A single gap - due to single event that removed a number of residues.
  - Each separate gap - due to distinct independent events.
- Protein structure
  - Protein secondary structure consists of alpha helixes, beta sheets and loops
  - Loops of varying size can lead to very similar structure.

38

## Biological Motivation

## cDNA matching

- **cDNA** - is the sequence after splicing and editing, after the introns have been removed.
- We expect regions of high similarity separated by long gaps.
- These gaps correspond to the introns removed by splicing






## Gap Penalty Models

- Constant Model
  - Gives each gap a constant weight, spaces are free
  - Maximize: $\sum \sigma(S_i', T_i') + W_g \times \#gaps$
  - Time $O(nm)$
  - Works well for cDNA matching
- Affine Model
  - There is a penalty for starting a gap and a penalty for each space extending it.
  - A single gap contributes $W_g + qW_s$
  - Maximize: $\sum s(S_i', T_i') + W_g \times \#gaps + W_s \times \#spaces$
  - Time $O(nm)$
  - Most widely used

41

## Gap Penalty Models

- Convex model
  - Each extra space contributes less penalty
  - Gap function is convex in length
  - Example $W_g + W_s \log q$
  - Time $O(nm \log m)$
  - Better model of biology
- General model
  - The weight of a gap is some arbitrary $w(q)$
  - Time $O(nm^2 + mn^2)$

42

## Example Revised

```
     AAC-AATTAAG-ACTAC-GTTCATGAC
I
     A-CGA-TTA-GCAC-ACTG-T-C-GA-

     AACAATTAAGACTACGTTCATGAC---
II
     AACAATT--------GTTCATGACGCA
```

43

## Indel model

```
AAC-AATTAAG-ACTAC-GTTCATGAC
A-CGA-TTA-GCAC-ACTG-T-C-GA-
```

-6 I

```
AACAATTAAGACTACGTTCATGAC---
AACAATT--------GTTCATGACGCA
```

-6 II

Scoring Parameters:
Match: +1
indel: -2

44

## Constant model

```
AAC-AATTAAG-ACTAC-GTTCATGAC
A-CGA-TTA-GCAC-ACTG-T-C-GA-
```

-6 I

```
AACAATTAAGACTACGTTCATGAC---
AACAATT--------GTTCATGACGCA
```

12 II

Scoring Parameters:
Match: +1
open gap: -2

## Affine model

-17 I

```
AAC-AATTAAG-ACTAC-GTTCATGAC
A-CGA-TTA-GCAC-ACTG-T-C-GA-
```

1 II

```
AACAATTAAGACTACGTTCATGAC---
AACAATT--------GTTCATGACGCA
```

Scoring Parameters:
Match: +1
Open Gap: -2
Extend Gap:-1



## Convex model

-6 I

```
AAC-AATTAAG-ACTAC-GTTCATGAC
A-CGA-TTA-GCAC-ACTG-T-C-GA-
```

~7 II

```
AACAATTAAGACTACGTTCATGAC---
AACAATT--------GTTCATGACGCA
```

Scoring Parameters:
Match: +1
Open Gap: -2
Gap Length: -logn

## Affine Weight Model

- We divide the possible alignments of the prefixes $S_{1..i}$ and $T_{1..j}$ into 3 types:

*A(i,j)*
```
S_______i
T_______j
```

*B(i,j)*
```
S_______i------
T_____________j
```

*C(i,j)*
```
S_____________i
T_______j-----
```



## Affine Weight Model

Recurrence relations

$$A(i,j) = \max \begin{cases} A(i-1, j-1) + s(i,j) \\ B(i-1, j-1) + s(i,j) \\ C(i-1, j-1) + s(i,j) \end{cases}$$

$$B(i,j) = \max \begin{cases} A(i-1, j-1) + W_g + W_s \\ B(i-1, j-1) + W_s \end{cases}$$

$$C(i,j) = \max \begin{cases} A(i-1, j-1) + W_g + W_s \\ C(i-1, j-1) + W_s \end{cases}$$

## Affine Weight Model

Initial Conditions:

$$A(i,0) = B(i,0) = W_g + iW_s$$

$$A(0,j) = C(i,0) = W_g + jW_s$$

Optimal alignment :

$$V(n,m) = \max\{A(n,m), B(n,m), C(n,m)\}$$

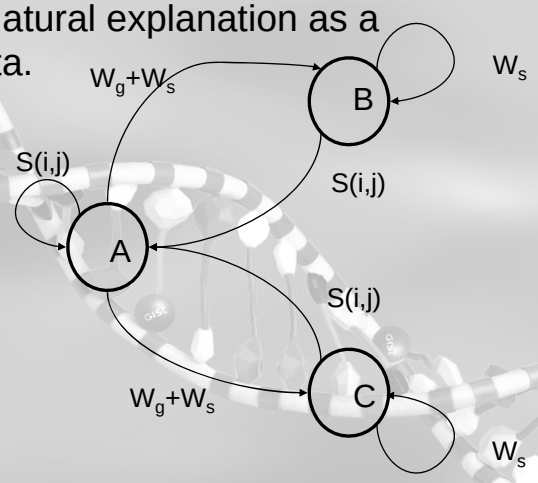
Complexity

- Time: *O(nm)* we compute 3 matrices.
- Space: *O(nm)*



## Affine Weight Model

This model has a natural explanation as a finite state automata.



51

## Alignment in Real Life

- One of the major uses of alignments is to find sequences in a “database”
- Such collections contain massive number of sequences (order of $10^6$)
- Finding homologies in these databases with dynamic programming can take too long

52

## Heuristic Search

- Instead, most searches relay on **heuristic** procedures
- These are not guaranteed to find the best match
- Sometimes, they will completely miss a high-scoring match

We now describe the main ideas used by some of these procedures

- Actual implementations often contain additional tricks and hacks

53

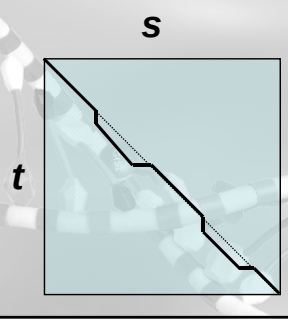
## Basic Intuition

- Almost all heuristic search procedure are based on the observation that real-life matches often contain long strings with gap-less matches
- These heuristic try to find significant gap-less matches and then extend them

54

## Banded DP

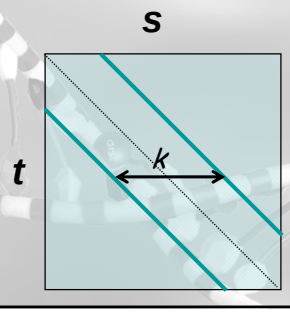
- Suppose that we have two strings *s[1..n]* and *t[1..m]* such that *n≈m*
- If the optimal alignment of s and t has few gaps, then path of the alignment will be close to diagonal



55

## Banded DP

- To find such a path, it suffices to search in a diagonal region of the matrix
- If the diagonal band has width *k*, then the dynamic programming step takes $O(kn)$
- Much faster than $O(n^2)$ of standard DP




56

## Banded DP

**Problem:**

- If we know that *t[i..j]* matches the query *s*, then we can use banded DP to evaluate quality of the match
- However, we do not know this apriori!

How do we select which sequences to align using banded DP?

57

## FASTA Overview

Main idea:

- Find potential diagonals & evaluate them
- Suppose that we have a relatively long gap-less match

  AGCGCCATGGATTGAGCGA

  TGCGACATTGATCGACCTA

- Can we find "clues" that will let us find it quickly?

58

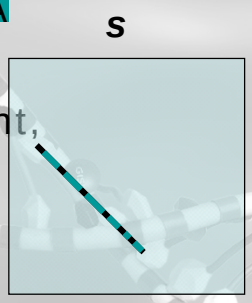
## Signature of a Match

**Assumption**: good matches contain several "patches" of perfect matches

AGCGCCATGGATTGAGCGA

TGCGACATTGATCGACCTA

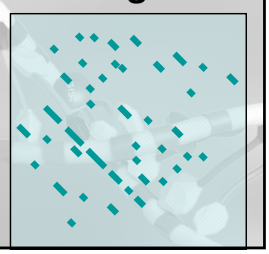
Since this is a gap-less alignment, all perfect match regions should be on one diagonal



59

## FASTA

- Given *s* and *t*, and a parameter *k*
- Find all pairs (i,j) such that s[i..i+k]=t[j..j+k]
- Locate sets of pairs that are on the same diagonal
  - By sorting according to *i-j*
- Compute score for the diagonal that contain all of these pairs




60

## FASTA

Postprocessing steps:

- Find highest scoring diagonal matches
- Combine these to potential gapped matches
- Run banded DP on the region containing these combinations

- Most applications of FASTA use very small *k*
  (2 for proteins, and 4-6 for DNA)

## FASTA Output

```
SCORES   Init1: 1201  Initn: 1844  Opt: 1915
Smith-Waterman score: 1915;   59.3% identity in 496 aa overlap

                                 10        20        30        40
A41264                     MADKKKITASLIYAVSVAAIGSLQFGYNTGVINAPEKIIQAFYNRTL
                             ::::|::|: ||  |::|||||||| ||||||:|:|:  ||:|
A49158       MPSGFQQIGSEDGEPPQQRVTGTLVLAVFSAVLGSLQFGYNIGVINAPQKVIEQSYNETW
                     10        20        30        40        50        60

              50            60        70        80        90       100
A41264       SQRSG----ETISPELLTSLWSLSVAIFSVGGMIGSFSVSLFVNRFGRRNSMLLVNVLAF
               |:|     :| |  ||:||:|||||||||||||:|| :::: : :||: :||: ||||
A49158       LGRQGPEGPSSIPPGTLTTLWALSVAIFSVGGMISSFLIGIISQWLGRKRAMLVNNVLAV
                     70        80        90       100       110       120

. . . . . .
```



## BLAST Overview

- BLAST uses similar intuition
- It relies on **high scoring** matches rather than exact matches
- It is designed to find alignments of a target string $s$ against large databases

## High-Scoring Pair

- Given parameters: length *k*, and threshold *T*
- Two strings *s* and *t* of length *k* are a **high scoring pair** (HSP) if $d(s,t) > T$
- Given a query *s[1..n]*, BLAST construct **all** words w, such that *w* is an HSP with a *k*-substring of *s*
  - Note that not all substrings of *s* are HSPs!
- These words serve as **seeds** for finding longer matches

## High Scoring Pair

**Query Sequence** M V G A S T P R Q G A I L V R W S

P R Q — Word

P R E
P K Q
P K E
H R Q
— Neighborhood Words

H T Q
A A A
— Below Threshold

65

## Finding Potential Matches

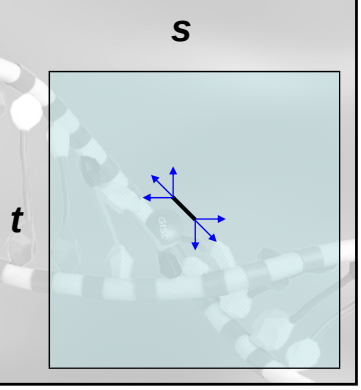
We can locate seed words in a large database in a single pass

- Construct a FSA that recognizes seed words
- Using hashing techniques to locate matching words

66

## Extending Potential Matches

- Once a seed is found, BLAST attempts to find a local alignment that extends the seed
- Seeds on the same diagonal are combined (as in FASTA)

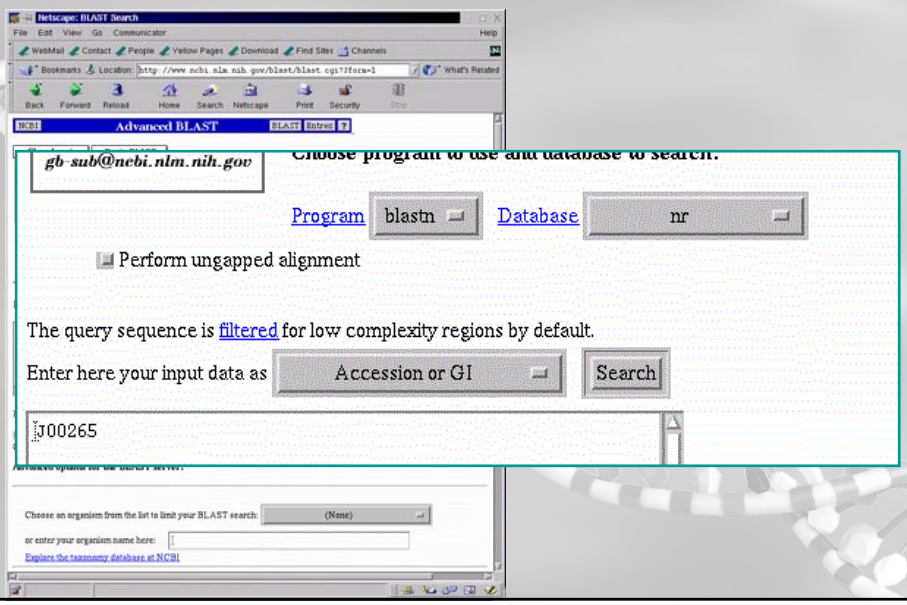


## BLAST programs

- BLASTN - Nucleotide query searching a nucleotide database.
- BLASTP - Protein query searching a protein database.
- BLASTX - Translated nucleotide query sequence (6 frames) searching a protein database.
- TBLASTN - Protein query searching a translated nucleotide (6 frames) database.
- TBLASTX - Translated nucleotide query (6 frames) searching a translated nucleotide (6 frames) database



## BLAST Search

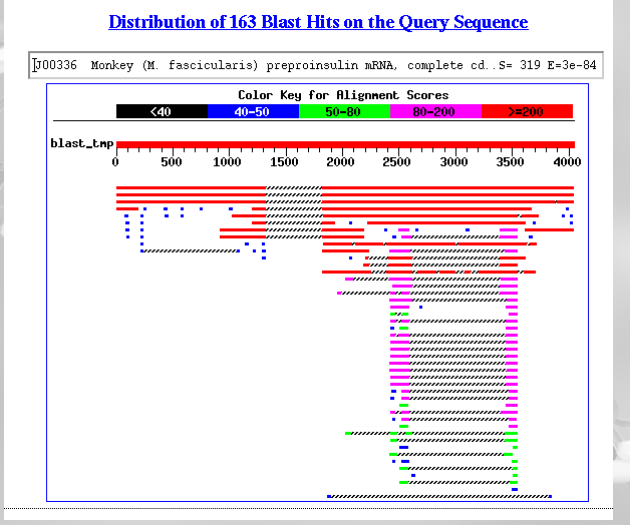


# BLAST Output

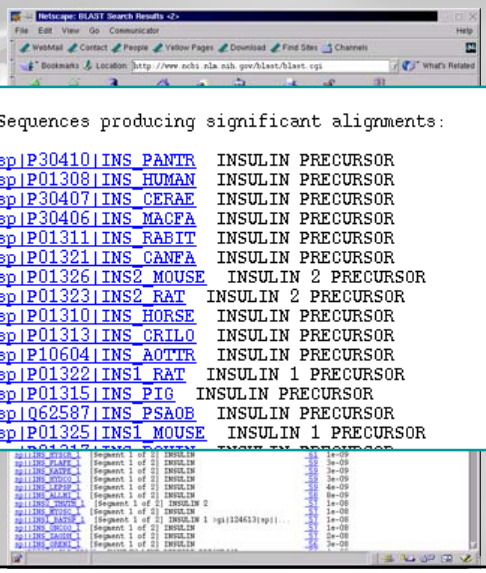
- List of hits
  - Database accession codes, name, description.
  - Score in bits (Usually >30 bits is significant )
  - Expectation value E()
- For each hit
  - A header including hit name, description, length
  - Each hit may contain several HSPs
  - Score and expectation value
    - how many identical residues
    - how many residues contributing positively to the score
- The local alignment itself



# BLAST Output



# BLAST Output



```
                                                          Score     E
Sequences producing significant alignments:               (bits)  Value

sp|P30410|INS_PANTR  INSULIN PRECURSOR                     184   6e-47
sp|P01308|INS_HUMAN  INSULIN PRECURSOR                     184   6e-47
sp|P30407|INS_CERAE  INSULIN PRECURSOR                     182   4e-46
sp|P30406|INS_MACFA  INSULIN PRECURSOR                     182   4e-46
sp|P01311|INS_RABIT  INSULIN PRECURSOR                     169   2e-42
sp|P01321|INS_CANFA  INSULIN PRECURSOR                     166   1e-41
sp|P01326|INS2_MOUSE  INSULIN 2 PRECURSOR                  157   7e-39
sp|P01323|INS2_RAT  INSULIN 2 PRECURSOR                    157   7e-39
sp|P01310|INS_HORSE  INSULIN PRECURSOR                     157   7e-39
sp|P01313|INS_CRILO  INSULIN PRECURSOR                     157   1e-38
sp|P10604|INS_AOTTR  INSULIN PRECURSOR                     156   2e-38
sp|P01322|INS1_RAT  INSULIN 1 PRECURSOR                    154   8e-38
sp|P01315|INS_PIG  INSULIN PRECURSOR                       152   2e-37
sp|Q62587|INS_PSAOB  INSULIN PRECURSOR                     151   7e-37
sp|P01325|INS1_MOUSE  INSULIN 1 PRECURSOR                  146   2e-35
```



# BLAST Output

```
sp|P01323|INS2_RAT INSULIN 2 PRECURSOR
          Length = 110

 Score =  157 bits (394), Expect = 7e-39
 Identities = 73/86 (84%), Positives = 77/86 (88%)
 Frame = +2

Query: 83  FVNQHLCGSHLVEALYLVCGERGFFYTPKTRREAEDLQVGQVELGGGPGAGSLQPLALEG 262
           FV QHLCGSHLVEALYLVCGERGFFYTP +RRE ED QV Q+ELGGGPGAG LQ LALE
Sbjct: 25  FVKQHLCGSHLVEALYLVCGERGFFYTPMSRREVEDPQVAQLELGGGPGAGDLQTLALEV 84

Query: 263 SLQKRGIVEQCCTSICSLYQLENYCN 340
           + QKRGIV+QCCTSICSLYQLENYCN
Sbjct: 85  ARQKRGIVDQCCTSICSLYQLENYCN 110
```

73

## What do we use

- Originally Blast did not allow gaps.
  - Now people use gapped-Blast
  - Gapped blast joins different diagonals.
- For proteins Blast is superior
- For nucleotides Fasta is better.

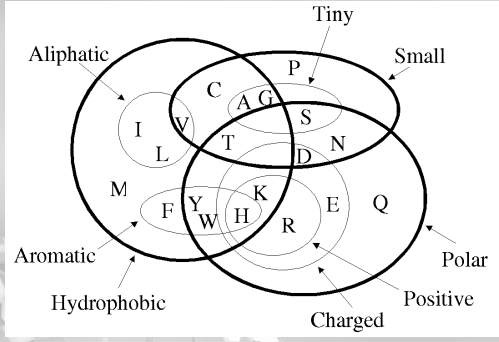
86

## Protein Alignments

- As we saw, there are many possible alignments, often with the same score
- We are interested in biologically meaningful alignments
- The resulting alignment depends on the score assigned to pairs of amminoacids and on the indels and gap penalty functions

## Amminoacid Similarity

- Amminoacids can be classified according to their chemical and physical properties.
- When comparing them, one must take into account these properties




88

## Scoring Matrices

- Scoring Matrices assign a numerical score to any possible pairs of amminoacids, accounting for their chemical and physical properties
- Amminoacid Substitution Matrices *or* Symbol Comparison Tables
- There are tables for protein comparison and for nucleic acid comparison
- A huge number of such matrices are available, each based on different substitution models

## PAM Matrices

- PAM (Point Accepted Mutations) matrices were developed at the end of the '70s by analysing the mutations of amminoacid sequences of proteins superfamilies tightly related.
- They noticed that these mutations were not at all random.
- Some substitutions occured more often than others, probably because they do not alter the function and/or the structure of a protein.

## Unit and PAM Matrices

- We use PAM units to measure the distance among amminoacid sequences.
- Two sequences S1 and S2 are 1 PAM unit apart if S1 can be transformed into S2 with 1 single mutation every 100 amminoacids, on average.
- In general, an amminoacid could mutate many times, eventually returning to its original value; therefore, two sequences that are 1 PAM apart, may be different less than 1%.

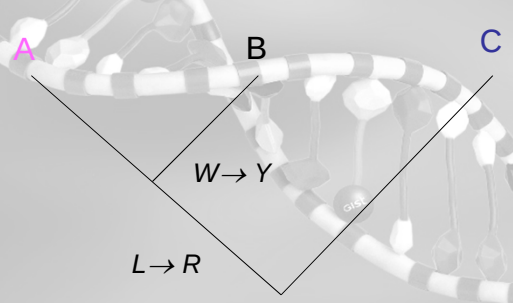
91

## PAM Matrices

- According to this model, amminoacidic substitutions observed in a given period of time, can be extrapolated for longer periods
- In the computation, a mutation in a given site is considered independent of previous mutational events in the same site

92

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Specie A | A | W | T | V | A | S | A | V | R | T | S | I |
| Specie B | A | Y | T | V | A | A | A | V | R | T | S | I |
| Specie C | A | W | T | V | A | A | A | V | L | T | S | I |



93

## PAM Matrices Computation: An Example

- $p_i = a_i/a_{tot}$ *frequency of amminoacid i*
- $f_{ij} = n(a_i \rightarrow a_j)$ *number of mutations* $a_i \rightarrow a_j$
- $f_i = \sum_j f_{ij}$ *number of mutations of* $a_i$
- $f = \sum_i f_i$ *total number of mutations*

94

## PAM Matrices Computation: An Example

- If $m_i = f_i/100 \cdot f\, p_i$ is the probability of mutation of $a_i$, then

$$M_{ii} = 1 - m_i$$

is the probability of conservation of $a_i$

- The probability of a mutation $a_i \rightarrow a_j$ is

$$M_{ij} = (f_{ij}/f_i)\, m_i$$

## PAM Matrices Computation: An Example

- Matrix $M_{ij}$ so computed is a transition matrix
- In general, to compute the probabilities for k evolutionary steps:

$$M_{ij}^{k}$$

## PAM Matrices

- There are different types of PAM Matrices, each one is used to compare 2 squences that are a given number of PAM units apart.
- For instance, PAM250 can be used to compare sequences that are 250 PAM units apart.
- An entry (i,j) of the PAM250 matrix contains the score of the pair of amminoacids (Ai,Aj); such score is proportional to the expected frequency of the substitution of Ai with Aj in two sequences that are 250 PAM units apart.

Table 1 – The log odds matrix for 250 PAMs (multiplied by 10)

| | A | C | D | E | F | G | H | I | K | L | M | N | P | Q | R | S | T | V | W | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 2 | -2 | 0 | 0 | -4 | 1 | -1 | -1 | -1 | -2 | -1 | 0 | 1 | 0 | -2 | 1 | 1 | 0 | -6 | -3 |
| C | | 12 | -5 | -5 | -4 | -3 | -3 | -2 | -5 | -6 | -5 | -4 | -3 | -5 | -4 | 0 | -2 | -2 | -8 | 0 |
| D | | | 4 | 3 | -6 | 1 | 1 | -2 | 0 | -4 | -3 | 2 | -1 | 2 | -1 | 0 | 0 | -2 | -7 | -4 |
| E | | | | 4 | -5 | 0 | 1 | -2 | 0 | -3 | -2 | 1 | -1 | 2 | -1 | 0 | 0 | -2 | -7 | -4 |
| F | | | | | 9 | -5 | -2 | 1 | -5 | 2 | 0 | -4 | -5 | -5 | -4 | -3 | -3 | -1 | 0 | 7 |
| G | | | | | | 5 | -2 | -3 | -2 | -4 | -3 | 0 | -1 | -1 | -3 | 1 | 0 | -1 | -7 | -5 |
| H | | | | | | | 6 | -2 | 0 | -2 | -2 | 2 | 0 | 3 | 2 | -1 | -1 | -2 | -3 | 0 |
| I | | | | | | | | 5 | -2 | 2 | 2 | -2 | -2 | -2 | -2 | -1 | 0 | 4 | -5 | -1 |
| K | | | | | | | | | 5 | -3 | 0 | 1 | -1 | 1 | 3 | 0 | 0 | -2 | -3 | -4 |
| L | | | | | | | | | | 6 | 4 | -3 | -3 | -2 | -3 | -3 | -2 | 2 | -2 | -1 |
| M | | | | | | | | | | | 6 | -2 | -2 | -1 | 0 | -2 | -1 | 2 | -4 | -2 |
| N | | | | | | | | | | | | 2 | -1 | 1 | 0 | 1 | 0 | -2 | -4 | -2 |
| P | | | | | | | | | | | | | 6 | 0 | 0 | 1 | 0 | -1 | -6 | -5 |
| Q | | | | | | | | | | | | | | 4 | 1 | -1 | -1 | -2 | -5 | -4 |
| R | | | | | | | | | | | | | | | 6 | 0 | -1 | -2 | 2 | -4 |
| S | | | | | | | | | | | | | | | | 2 | 1 | -1 | -2 | -3 |
| T | | | | | | | | | | | | | | | | | 3 | 0 | -5 | -3 |
| V | | | | | | | | | | | | | | | | | | 4 | -6 | -2 |
| W | | | | | | | | | | | | | | | | | | | 17 | 0 |
| Y | | | | | | | | | | | | | | | | | | | | 10 |



# BLOSUM Matrices (Henikoff & Henikoff, 1992)

- Blocks Amino Acid Substitution Matrices = BLOSUM
- Based on the amminoacid substitutions observed in ~2000 conserved blocks of sequences.
- These blocks are extracted from 500 protein families
- Segments belonging to each block are clustered according to their similarity. Every cluster is considered as a single sequence and the number of mutations in each column is computed

## Example of BLOSUM Matrix computation

99

. . . A . . .
. . . A . . .
. . . A . . .
. . . A . . .
. . . S . . .
. . . A . . .
. . . A . . .
. . . A . . .
. . . A . . .
. . . A . . .

- 9 A and 1 S
- 36 A→A ($f_{AA}$) and 9 A→S ($f_{AS}$)
- 210 possible pairs of amminoacids
- The frequency of A→A is $q_{AA} = f_{AA}/(f_{AA} + f_{AS}) = 0.8$
- The frequency of A→S is $q_{AS} = f_{AS}/(f_{AA} + f_{AS}) = 0.2$

## Example of BLOSUM Matrix computation

100

- The expected frequency that A is involved in a mutation is $p_A = (q_{AA} + q_{AS}/2) = 0.9$
- The expected frequency that S is involved in a mutation is $p_S = (q_{AS}/2) = 0.1$
- The expected frequency of a pair AA is $e_{AA} = p_A^2 = 0.81$
- The expected frequency of a pair AS is $e_{AS} = 2\, p_A\, p_S = 0.18$

## Example of BLOSUM Matrix computation

101

- The score for AA in the matrix is $q_{AA}/e_{AA} = 0,99$ and for AS is $q_{AS}/e_{AS} = 1.11$
- The values are converted in bits:

$$s_{AA} = \log_2(q_{AA}/e_{AA}) = -0.04$$

$$s_{AS} = \log_2(q_{AS}/e_{AS}) = 0,30$$

## BLOSUM Matrix computation

102

- From every cluster a matrix is computed
- For instance, a cluster that contains sequences with >60% identities results in a matrix called BLOSUM60
- The most frequently used matrix is BLOSUM62

Table 2 – The log odds matrix for BLOSUM 62

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



## The Hazard of Large Databases

- Define $p_\varepsilon = P(d(s,t) > \varepsilon \mid U)$
- This is the probability that two unrelated sequences will match with score $> \varepsilon$ by chance
- Assuming that they are independent of each other, and all are unrelated to $s$, we have

$$P(\max_t d(s,t) > \varepsilon) = 1 - (1 - p_\varepsilon)^N \approx 1 - e^{-Np_\varepsilon}$$

105

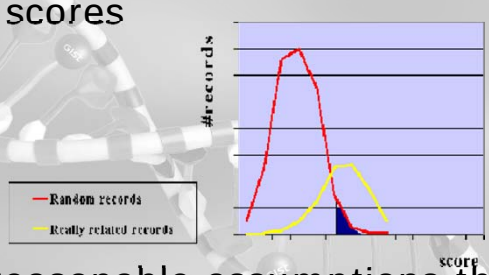
## Local Matching

- Question: Which local alignment query is expected to give a higher score:
  - To a short sequence
  - To a long sequence?
- A local match can begin at any of the *nm* entries in the **DP** matrix.
- The score is the optimal of all these starting points.
- If all starting points were independent we would need to calculate the probability of attaining such a score in *nm* trails.

106

## Score Significance-Fasta

- How meaningful is a score?
- Calculate distribution of scores and related scores




- Under reasonable assumptions the scores for un-gapped alignment behave according to the Extreme Value Distribution

107

## Extreme Value Distribution (BLAST)

- We ask the following questions: Given a database of size m and a sequence of size n
- What is the expected number of hits with score at least S? This number is called an E-score

$$E(S) = Kmne^{-\lambda S}$$

- Notice this is a Poisson distribution.
  - K corrects for the dependencies
  - λ depends on the scoring matrix
  - Doubling length of sequence doubles expectation
  - Doubling score causes E to decrease exponentially

108

## Blast P-value

- Recall Poisson distribution:
  - Probability of finding no hits with a score >= S

$$e^{-E}$$

  - Therefore probability of finding at least one hit with score >= S is

$$1 - e^{-E}$$

  - This is called the P-value.