BCH519 Week 5

Sequence Alignment I

Before we start...

Assignments:
Some short review articles assigned for reading
Written problems due next Tuesday

Today: Lecture on pairwise sequence alignment

Thursday: In-class exercises to be continued for homework. These will make use of your new Python skills and some of the approaches we discuss in class today.

Next week: Pairwise alignment continued; multiple sequence alignment

Supplemental reading if you want more information:

Mount, chapters 3, 4, 6: a detailed treament of pairwise alignments

Korf, Yandell and Bedell, "BLAST." 2003, O'Reilly. In addition to all you'll ever need to know about BLAST, a very good concise treatment of pairwise alignment algorithms and scoring.

Many uses for aligning two or more sequences

- mapping to genome
- assessing homology
- analyzing genome structure
- finding similar/conserved domains

Different types of alingment for different situations

- pairwise alignment
- multiple alignments
- global alignments
- local alignments
- gapped and ungapped alignments
- optimal alignments
- heuristic alignments
- *short-read aligners*

Many alignment tools exist, each with its own strengths and weaknesses

BLAST	PSI-BLAST
FASTA	MAVID
ClustalW	HMMER
LALIGN	MEME
DIALIGN	PIPMAKER
LAGAN	MSA
T-COFFEE	BBA
SIM4	

Pairwise Alignments

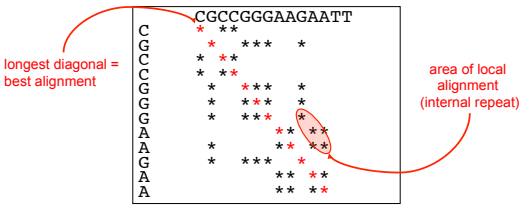
1. Dot matrix analysis
2. Optimal alignments
3. Word-based heuristics

Dot matrix analysis

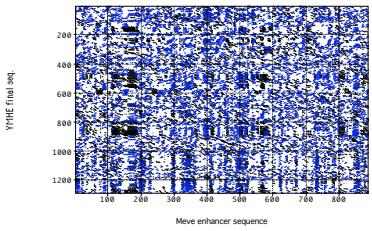
- Good when looking at two moderately-sized sequences
- Provides graphical overview of alignment
- Shows all possible matches of residues/bases
- Many implementations do not provide actual sequences
- Can incorporate a scoring matrix
- Not useful for database searching

Dot matrix analysis

- Create a matrix with one sequence across the top and the other down the left side
- In each row, place a dot in every column with a match
- Alignments are represented by diagonals



Dot matrix analysis



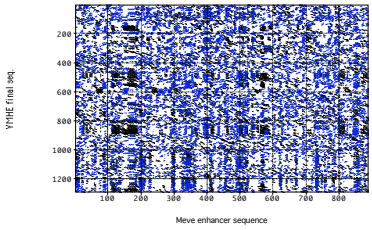
window=10, min. score=20%

Sliding Windows

CGGAATCACCACTGGATG

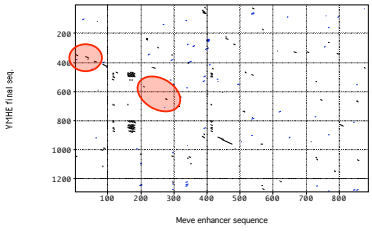
CGGAA ACCAC
GGAAT CCACT
GAATC CACTT
AATCA ACTGG
ATCAC CTGGA
TCACC TGGAT
CACCA GGATG

Dot matrix analysis



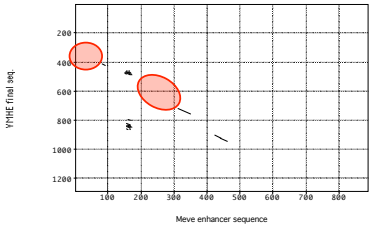
window=10, min. score=20%

Dot matrix analysis



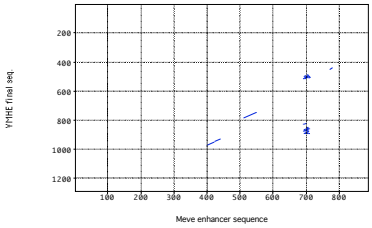
window=10, min. score=60%

Dot matrix analysis



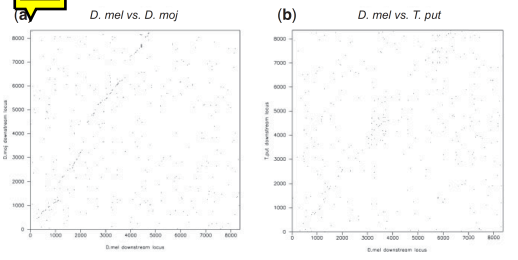
window=30, min. score=60%

Dot matrix analysis



window=30, min. score=60%

Dot matrix analysis



Kazemian, M., Suryamohan, K., Chen, J.-Y., Zhang, Y., Samee, M. A. H., Halfon, M. S., & Sinha, S. (2014). *Genome Biol Evol* **6**:2301.

Pairwise Alignments

1. Dot matrix analysis
2. Optimal alignments
3. Word-based heuristics

What are optimal alignments?

• Guaranteed to find the *mathematically* optimal alignment between two sequences

- can be more than one
- not necessarily the *biologically* optimal alignment

• Depends on a *scoring scheme* to calculate the optimal alignment

Scoring Functions and Alignments

• Scores are often taken from a *scoring matrix*, especially for protein alignments. We'll look at some of these shortly.

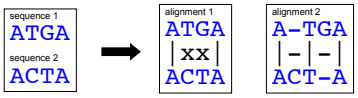
• Here's a simple function ω we can use to calculate a score:

ω(match) = +1
ω(mismatch) = -1
ω(gap) = -2

• The alignment score is just the sum of column scores

• The optimal alignment is the one with the maximum score

Scoring Functions and Alignments



ω(match) = +1, ω(mismatch) = -1, ω(gap) = -2

$$score_{alignment1} = \omega\begin{pmatrix}A\\A\end{pmatrix} + \omega\begin{pmatrix}T\\C\end{pmatrix} + \omega\begin{pmatrix}G\\T\end{pmatrix} + \omega\begin{pmatrix}A\\A\end{pmatrix} = 1 + (-1) + (-1) + 1 = 0$$

$$score_{alignment2} = \omega\begin{pmatrix}A\\A\end{pmatrix} + \omega\begin{pmatrix}-\\C\end{pmatrix} + \omega\begin{pmatrix}T\\T\end{pmatrix} + \omega\begin{pmatrix}G\\-\end{pmatrix} + \omega\begin{pmatrix}A\\A\end{pmatrix} = 1 + (-2) + 1 + (-2) + 1 = -1$$

Note for ω(gap) = -1, alignment 2 score = 1+(-1)+1+(-1) +1=1

Global vs Local Alignments

Global:

```
ACcaCAcA
||  ||| |
ACacCAtA
```

Global score = 2
Best local score = 2

Local:

```
--ACCAcaca
  ||||
acACCAta--
```

Global score = -6
Best local score = 4

Why do we need an efficient way to score and compute alignments?

The dot plots we looked at show all possible alignments between two sequences. But there are approximately

$$\frac{2^{2N}}{\sqrt{2\pi N}}$$

alignments for any two seqeunces of length *N* (10^{179} for *N*=300). So we need something more computationally tractable.

What is Dynamic Programming?

"A **dynamic programming** algorithm solves every subproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the subproblem is encountered"

--Cormen et al. "Introduction to Algorithms," MIT press

Dynamic Programming Algorithm

Four basic parts:

1. recursive definition of optimal score
2. matrix of scores for subproblems
3. bottom-up approach of filling the matrix, solving smallest subproblems first
4. traceback of the matrix to get optimal solution

Needleman-Wunsch Algorithm

The basic dynamic programming algorithm for global pairwise alignment is the Needleman-Wunsch algorithm. A version for local alignments was developed by Smith and Waterman.

Let's go through an example*, using the following scoring function:

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

and the sequences

ACTCG and **ACAGTAG**

*Krane and Raymer, Fundamental Concepts of Bioinformatics

Needleman-Wunsch Algorithm

Step 1: Initialize the matrix

Values are assigned to the first row/first column. Values are the gap penalty times the distance from the origin:

		A	C	T	C	G
	0	-1	-2	-3	-4	-5
A	-1					
C	-2					
A	-3					
G	-4					
T	-5					
A	-6					
G	-7					

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

Needleman-Wunsch Algorithm

Step 2: Fill in the matrix

Three values are computed for each cell: a match/mismatch (diagonal) score, horizontal gap score, vertical gap score. Each is the sum of a preceeding cell and the respective score for the current cell. The maximum value is placed in the cell.

```
match: A:A =  0+1= 1
hgap:  A:- = -1+(-1)= -2
vgap:  -:A = -1+(-1)= -2
```

		A	C	T	C	G
	0	-1	-2	-3	-4	-5
A	-1	**1**				
C	-2					
A	-3					
G	-4					
T	-5					
A	-6					
G	-7					

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

Needleman-Wunsch Algorithm

Step 2: Fill in the matrix

We continue down the row, filling in scores. As we go, we leave a pointer indicating the source of the score--diagonal, horizontal, or vertical

```
mismatch: A:C = (-1)+(0)= -1
hgap:     A:- = 1+(-1)= 0
vgap:     -:C = -2+(-1)= -3
```

		A	C	T	C	G
	0	-1	-2	-3	-4	-5
A	-1	**1**	**0**			
C	-2					
A	-3					
G	-4					
T	-5					
A	-6					
G	-7					

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

Needleman-Wunsch Algorithm

Step 2: Fill in the matrix

Continue with this process until the entire matrix is filled in. Note that at any one step, all we need to calculate is a single cell score. The value in the bottom right cell is the optimal score of the alignment, as you will see.

```
match: G:G = 1+1= 2
hgap:  G:- = 0+(-1)= -1
vgap:  -:G = 1+(-1)= 0
```

		A	C	T	C	G
	0	-1	-2	-3	-4	-5
A	-1	**1**	0	-1	-2	-3
C	-2	0	2	1	0	-1
A	-3	-1	1	2	1	0
G	-4	-2	0	1	2	2
T	-5	-3	-1	1	1	2
A	-6	-4	-2	0	1	1
G	-7	-5	-3	-1	0	**2**

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

note: most pointers omitted for clarity

Needleman-Wunsch Algorithm

Step 3: Traceback

Start in the bottom right cell and follow the arrows back to the origin. The sequence is reconstructed from right to left following the original rules: diagonal for a match/mismatch, vertical for a gap in the upper sequence, horizontal for a gap in the lefthand sequence.

```
AC--TCG
||  | |
ACAGTAG
```

score=1+1+(-1)+(-1)+1+0+1=2

		A	C	T	C	G
	0	-1	-2	-3	-4	-5
A	-1	**1**	0	-1	-2	-3
C	-2	0	**2**	1	0	-1
A	-3	-1	1	2	1	0
G	-4	-2	0	1	2	2
T	-5	-3	-1	1	1	2
A	-6	-4	-2	0	1	1
G	-7	-5	-3	-1	0	**2**

ω(match) = +1
ω(mismatch) = 0
ω(gap) = -1

note: most pointers omitted for clarity

Smith-Waterman Algorithm

The Smith-Waterman algorithm for local alignment is similar, except that the edges of the matrix are initialized to zero and the maximum score is never less than zero. Pointers are only placed if the score is >0, and traceback starts from the highest-scoring cell and ends at a score of zero.

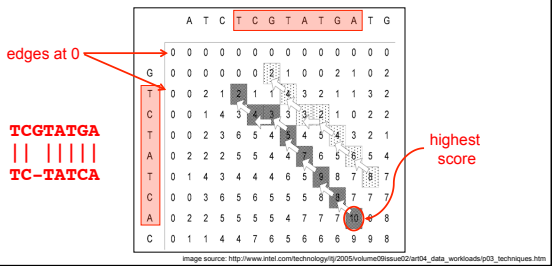


image source: http://www.intel.com/technology/itj/2005/volume09issue02/art04_data_workloads/p03_techniques.htm

Computational Considerations

Even these dynamic programming algorithms remain computationally and memory intensive and increase as the product of the sequence lengths. Aligning two 100 kb sequences requires about 80 GB of RAM! There are variations of these algorithms that can reduce this, but they are still not suitable for aligning long sequences or for many rapid alignments, as in a database search.

Needleman-Wunsch Algorithm

Sometimes there can be more than one way to perform the traceback. This means that there are alternative, equally optimal alignments. But mathematically equivalent alignments are not necessarily biologically equivalent:

```
ACA------TCGGAT      AC------ATCGGAT
|||      |||||       ||      ||||||
ACAGCATTATCGGAC      ACAGCATTATCGGAC
```

For instance, these two alignments have equivalent scores. But suppose we know that the sequence ATCGGA is an important transcription factor binding site? This would lead us to favor the righthand alignment as being more biologically meaningful. Note that this could have consequences for our sequence analysis—for instance, a search for the string ATCGGA (you know how to do this now, using Python!) would fail to find the site in the alignment on the left, even though it clearly exists in the (ungapped) actual top sequence, and is perfectly conserved between the two sequences.

```
ACATCGGAT
```

More on Scoring: Affine Gaps

Most alignment algorithms use *affine gaps*. With affine gap penalties, there is a different score depending on whether you are opening or extending a gap. Why?

Let's look at some simple gapped alignments:

alignment 1
```
AC--TCG
||  |||
ACAGTCG
```

alignment 2
```
AC-------TCG
||       |||
ACAGCATTATCG
```

alignment 3
```
AC----T---CG
||    |   ||
ACAGCATTATCG
```

The first two alignments represent a similar biological phenomenon: a single *indel* (insertion/deletion) event. The third has two such events. Here's how they score using a fixed gap penalty of -5 and a match score of +2:

alignment 1: 2+2-5-5+2+2+2= 0

alignment 2: 2+2-5-5-5-5-5-5+2+2+2= -25

alignment 3: 2+2-5-5-5-5+2-5-5-5+2+2= -25

More on Scoring: Affine Gaps

alignment 1
```
AC--TCG
||  |||
ACAGTCG
```

alignment 2
```
AC-------TCG
||       |||
ACAGCATTATCG
```

alignment 3
```
AC----T---CG
||    |   ||
ACAGCATTATCG
```

Alignment 1 scores 0, but alignment 2 scores -25. Meanwhile, alignment 3 scores the same as alignment 2. Does this make sense?

Aren't 1 and 2 really equivalently good alignments? Does it matter how *large* a gap is?

What about 2 and 3? *A single indel event is more likely than multiple events*

alignment 1: 2+2-5-5+2+2+2= 0

alignment 2: 2+2-5-5-5-5-5-5+2+2+2= -25

alignment 3: 2+2-5-5-5-5+2-5-5-5+2+2= -25

More on Scoring: Affine Gaps

Let's see what happens if instead we employ an affine gap score in which opening a gap scores -7, but extending a gap scores -1:

alignment 1
```
AC--TCG
||  |||
ACAGTCG
```

alignment 2
```
AC-------TCG
||       |||
ACAGCATTATCG
```

alignment 3
```
AC----T---CG
||    |   ||
ACAGCATTATCG
```

alignment 1: 2+2-7-1+2+2+2= 2

alignment 2: 2+2-7-1-1-1-1-1-1+2+2+2= -3

alignment 3: 2+2-7-1-1-1+2-7-1-1+2+2= -9

Note that the first two scores are now much closer together than they were when using the fixed gap penalty, and that alignment 3 now scores worse than alignment 2.

This type of scoring scheme favors a few long gaps over many short gaps, which seems more biologically plausible.

Scoring matrices

- Also referred to as *substitution matrices*
- First developed by Dayhoff in 1970s for amino acid similarity
- The Dayhoff, or PAM (point accepted mutation) matrices are based on multiple alignments of related proteins, and calculate how frequently a given amino acid is substituted for another. For instance, phenylalanine is often substituted for tyrosine, a similar aromatic amino acid. The PAM matrices are based on a limited data set and make certain evolutionary assumptions (see next slide).
- The PAM1 matrix used proteins of 85% or better identity
- The PAM1 values can essentially be interpreted as the probability that 1 amino acid in 100 will undergo substitution

Scoring matrices—PAM

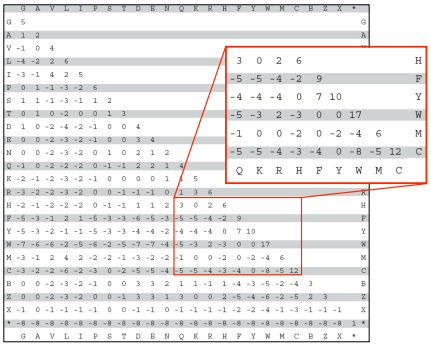
• The PAM100 matrix was made by multiplying the PAM1 matrix by itself 100 times, 250 times for the PAM250, etc. The idea was to approximate the increased substitution frequencies over the course of evolution.

• So the more divergent the sequences you want to compare, the higher the number PAM matrix you should use.

• One drawback to this method is that any estimate errors in PAM1 get multiplied in the higher-number matrices.

• In practice, PAM250 has stood up well for moderately diverged proteins (e.g., 20% similarity). PAM120 is a common choice for closely related sequences.

PAM250



adapted from Pertsemlidid and Fondon 2001

Scoring matrices—BLOSUM

• In the 1990s, Henikoff and Henikoff developed the BLOSUM (BLOcks SUbstitution Matrix) matrices. These take advantage of the much greater amount of available sequence data compared to when the PAM matrices were constructed.

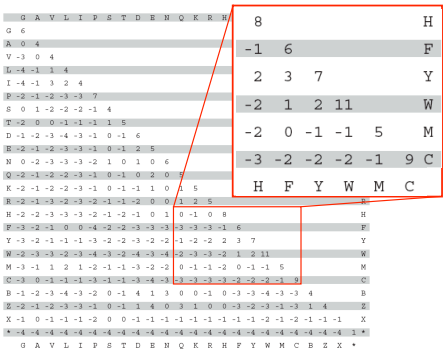
• The BLOSUM matrices take ungapped “blocks” of sequence from multiple alignments of protein families, and then cluster the blocks based on percent identity before calculating the matrix scores.

• BLOSUM62 uses sequences all at least 62% identical, BLOSUM80 all 80% identical, etc.

• Opposite to PAM, the higher the BLOSUM number, the more appropriate for closely related sequences

• The BLOSUM matrices appear to give better performance in many instances, such as when using BLAST, possibly on account of their greater degree of empirical data.

BLOSUM62



adapted from Pertsemlidid and Fondon 2001

Pairwise Alignments

1. Dot matrix analysis
2. Optimal alignments
3. Word-based heuristics

Word-based (*k*-tuple based) alignments

- It’s not computationally tractable to align long sequences or to search large sequence databases using the Needleman-Wunsch or Smith-Waterman algorithms
- Two popular heuristic algorithms, BLAST and FASTA, are not guaranteed to find optimal alignments (but usually do, or at least produce very close approximations)
- On the other hand, they are very fast
- Both rely on the basic assumption that alignable sequences will contain at least some short regions—“words”—of identity or high similarity
- We’ll look at BLAST, the more popular of the tools

BLAST: Basic Local Alignment Search Tool

- BLAST has become an almost indispensible and ubiquitous part of the biologist's toolkit
- Nevertheless, many biologists don't really use BLAST correctly, and certainly fail to take advantage of its power (see Box 1, "The good, the bad and the ugly" in the paper for today's homework)
- There are several flavors of BLAST, including BLASTN (for nucleic acids), BLASTP (proteins), BLASTX (search protein database using nucleotide sequence), etc.
- We'll look a little bit at the algorithm, at the theory, and at ways to use BLAST better
- We'll also look at running BLAST from the command line, formatting our own BLAST databases, and parsing BLAST outputs

The BLAST algorithm I: seeding

BLAST works by looking for all of the common "words" between the two sequences being compared. The default word size for protein alignments is 3. In other words, if your sequence is

```
PQGLLTFVRQLEIY
```

the first word is PQG. There are 8000 possible matches (20^3) to a 3-letter word.

We can score each match to a word. For instance, using BLOSUM62, a match of PQG to itself is the sum of a P-P, a Q-Q, and a G-G match, or 7+5+6=18. A match of PQG to PEG scores 15, and to PQA, 12.

BLAST uses a neighborhood threshold score *T* to decide how many matches, or "hits," to keep. For instance, at *T*=13, we would consider PEG a hit, but would discard PQA.

This reduces the number of possible matches from 8000 down to around 50.

The BLAST algorithm I: seeding

A high value for *T* limits the number of hits and therefore reduces the *search space* for the algorithm, making it run faster. However, it also reduces the *sensitivity*, meaning that there is a higher chance of missing a particular alignment.

Another important parameter is the word size *W*. Smaller word sizes will produce more hits. **The relationship between *T, W,* and the scoring matrix is of critical importance** for controlling the speed and sensitivity of BLAST.

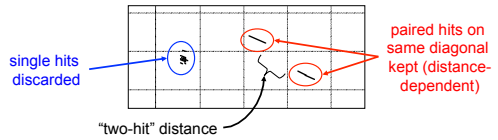
(For nucleotide BLASTN, words are always exact matches; *T* is not used and only *W* is varied.)

The BLAST algorithm I: seeding

BLAST then scans each database sequence for an exact match to one of the accepted neighborhood words. Each found match is considered a "seed."

This goes very fast, because BLAST has already pre-processed the sequence database and indexed it for all words.

In recent versions of BLAST, something called the "two-hit" algorithm is used. This requires that there be two word hits along the same diagonal within a specfied distance of one another. This helps to reduce spurious matches and increases search speed.



The BLAST algorithm II: extension

The seeds are then extended in each direction by local alignment using the Smith-Waterman algorithm. Extension proceeds up to a predetermined distance past the maximum score for the current alignment.

Although the original BLAST algorithm used only ungapped extensions, more recent versions can create gapped alignments using affine gap penalties as described earlier.

BLAST algorithm III: scoring and significance

The alignments are then scored and evaluated for statistical significance. The extended seeds that score above a statistical threshold are kept and referred to as "*high-scoring segment pairs*," or *HSPs*.

Raw *scores* depend on the similarity matrix and the gap penalties:

$$S = \left(\sum M_{ij}\right) - cO - dG$$

summed scores for each pair in the alignment

# of gaps x gap initiation penalty

gap length x gap extension penalty

BLAST algorithm III: scoring and significance

Normalized scores ("bit scores") allow for comparison of alignments generated using different scoring matrices and gap penalties.

$$S' = (\lambda S - \ln K) / \ln 2$$

λ and *K* reflect the matrices and penalties used for the alignment. λ is empirically estimated from aligning random sequences.

BLAST algorithm III: scoring and significance

The"E" value gives the expectation that a given score is based on chance alone and depends on the size of the database being searched. The *smaller* the E-value, the greater the significance of the match.

$$E = (n \times m) / \left(2^{S'}\right)$$

size of database (residues)

length of query sequence

Bit score

In some cases, the *E*-value you see in the BLAST report is based on the score of not just one but a group of HSPs. This is indicated in the BLAST report as Expect(n) where n equals the number of HSPs scored. The calculation for this is somewhat more complicated; we won't go into it here.

Karlin-Altschul equation

The statistics of BLAST are based on the *Karlin-Altschul* equation, which relates the various parameters:

$$E = kmne^{-\lambda S}$$

E: random expectation *k:* constant
m: size of query *n:* size of database
λS*:* normalized score

The BLAST algorithm: masking

Some sequence words are quite common in biology—e.g., poly-Q repeats, or simple nucleotide repeats such at CACACACA. Aligning these *low-complexity sequences* can lead to a lot of wasted time finding short but unimportant matches. BLAST contains options for *masking* such sequences.

Low-complexity filtering replaces these sequences with "N" or "X", which have negative scores for nucleic acid and protein alignments, respectively.

Sometimes this filtering can be too stringent and lead to lost alignments due to the break-up of HSPs into very short segments that don't score above threshold. A way to prevent this is to use "*soft masking*." Soft-masking disallows seeding in low-complexity regions, but allows extension of seeded alignments through them.

A few other BLAST notes:

Obviously, BLAST is a complex algorithm and this is a simplified treatment. Also, there are some differences in how things are done in the different BLAST programs, e.g., BLASTN vs. BLASTP.

Note that we have been discussing NCBI-BLAST. There is another version of BLAST, Washington University's WU-BLAST, that has a number of differences, although the essentials are similar.