

Practice 1: Search for homologous proteins with BLAST

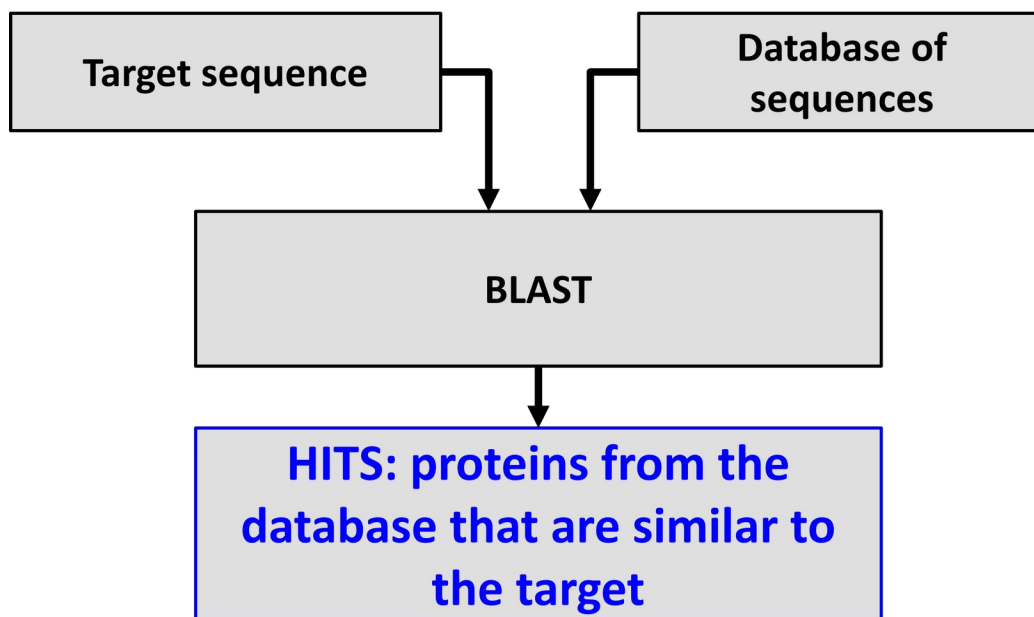
We propose the following problem: We have the sequence of a protein of which we know nothing, neither its function nor structure. This is called protein-problem or target. We want to know if there is a protein with known structure that shares a common ancestor with the target protein (definition of homology). This homologous protein with known structure is called template. We are interested in finding templates because we can use them to model the structure of the target protein. In this exercise we will explore different strategies to find template proteins using BLAST, Hidden Markov Models (HMM) and structure superimposition.

Theoretical concepts:

BLAST (Basic Local Alignment Search Tool): is a program to compare biological sequences and perform local sequence alignments. We will use it to find similar sequences to our target sequence in databases of sequences. BLAST works following these steps:

1. Splits the target sequence into smaller subsequences, also known as words.
2. Matches these words on the database sequence according to a score.
3. The alignments between the words and the database sequences are extended. On each extension step the score of the alignment changes.
4. If the alignment score decreases under a certain threshold the extension stops.
5. All the alignments that show a statistically significant score are returned as hits.

All the protein sequences identified as similar to the target protein are called **hits**.



BLAST provides two statistical measurements for each one of these hits: the score and the expected value (or e-value). High scores will indicate that the two proteins have similar sequences. The e-value is computed as a function of the score and it is the probability of finding a hit with that score just by chance. Besides the score, the computation of the e-value takes into account other parameters such as the size of the database or the length of the alignment.

The e-value can be expressed by the next formula, where n and m are the number of residues of the sequences being compared, N is the size of the database, S is the score, μ is the average of scores and K and λ are other parameters.

$$E(S) \approx \exp\left(-NmnKe^{-(S-\mu)}\right)$$

BLAST computes scores by using substitution matrices. Substitution matrices assign a score to each one of the aligned residues in an alignment. In this tutorial we will use BLAST and PSI-BLAST. PSI-BLAST is the iterative version of BLAST and obtains more specific substitution matrices leading to a better performance finding homologous proteins.

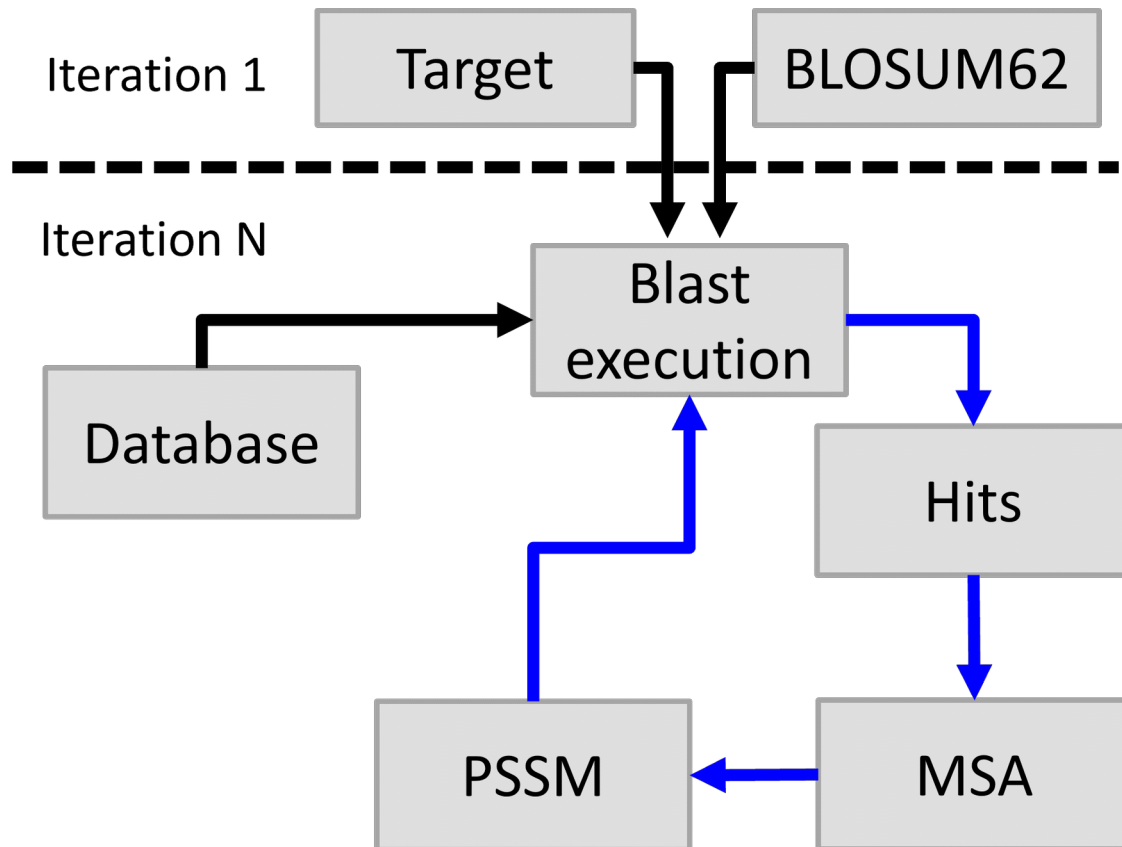
Substitution matrix: Substitution matrices contain scores associated to the frequency of substitution between two amino acids. For example, if you have glutamate (E) aligned with aspartate (D) the score of the aligned pair is $M(E, D)$, where M comes from the substitution matrix.

Substitution matrices are obtained from multiple sequence alignments (MSA). They contain the information regarding residue substitution and conservation of the MSA. According to the information that contain, substitution matrices can be:

- **Non position-specific:** substitution scores are the same for all the positions in the alignment. To obtain non position-specific substitution matrices you compute the substitution frequencies for all positions in the MSA at once.
- **Position-specific:** substitution scores are different for each one of the positions of the alignment. It is like having a different non position-specific substitution matrix for each one of the positions in the alignment. To obtain position-specific substitution matrices you compute the substitution frequencies for each one of the positions in the MSA separately.

One of the most widely used matrices is Blosom62. This has been obtained by aligning blocks of similar sequences, presumably indicating the evolutionary changes of the residues of a protein. Blosom62 is a non position-specific substitution matrix and is the default substitution matrix used by BLAST.

During this tutorial we will use PSI-BLAST, the iterative version of BLAST. PSI-BLAST starts to search for homologous proteins with a BLOSUM62 matrix. Then, finds the hits, aligns their sequences in a MSA, and from this alignment obtains a position specific substitution matrix (PSSM). Afterwards, PSI-BLAST will search again for homologous sequences but using the PSSM computed previously in stead of the BLOSUM62 substitution matrix. PSI-BLAST works by searching new hits and updating its substitution matrix at each iteration. This enables that, after several iterations, the PSSM is specific for the type of substitutions that happen in the homologous proteins to our target. Therefore, hits are found with higher specificity. The next scheme shows how is PSI-BLAST performing:



Tutorial:

Step 1: Using BLAST

Within "exercise_2" you will find the subdirectory BLAST. Within this there is the sequence problem named "target.fa".

To look for proteins of known structure similar to the target protein, try:

```
blastp -query target.fa -db /mnt/NFS_UPF/soft/databases/blastdat/pdb_seq  
-out target_pdb.out
```

Example of BLAST usage:

> **blastp -query [target_fasta_format]-db [database]-out [output]**

You can see the result of the search in the output file target_pdb.out.

Step 2: Using PSI-BLAST

So far, we have been only using Blosum62. This substitution matrix is generic; therefore we always use the same matrix for all our searches. How could we improve the specificity of our sequence search?

Using a matrix that is only informative for the substitutions in the protein family of our target sequence.

Using a matrix that contains specific substitution scores for each one of the positions of the alignment.

These improvements are achieved using a **Position Specific Substitution Matrix** or PSSM. But, how can we obtain this matrix?

PSSMs are obtained from **Multiple Sequence Alignments** (or MSA). Within the MSA, the probabilities and scores for each possible substitution for all the positions of the MSA are computed. This MSA must contain the target sequence to indicate the reference position of the residues within the MSA.

One way to automatically perform a sequence search with PSSM is to use **PSI-BLAST**. This program is an **iterative** version of BLAST. It looks for sequences similar to the target sequence, it aligns them into a MSA and with this it recalculates a new substitution matrix. The matrix is stored and used to add more similar sequences.

Iteration 0 → Target + Blosum62 → [similar sequences] 0 → MSA 0 + Target → PSSM 0

Iteration 1 → PSSM 0 → [similar sequences] 1 → MSA 1 + Target → PSSM 1

Iteration 2 → PSSM 1 → [similar sequences] 2 → MSA 2 + Target → PSSM 2

.....

Example of PSI-BLAST usage:

> **psiblast -query [target_fasta_format] -db [database] -num_iterations [number of iterations] -out [output]**

Using the previous example, we can make an iterative search into the PDB database. We are going to make 5 iterations:

```
psiblast -query target.fa -db /mnt/NFS_UPF/soft/databases/blastdat/pdb_seq  
-num_iterations 5 -out target_pdb_5.out
```

Results per iteration are indicated by the word “Round”. You can search this word in the output file to compare the results at different iteration rounds.

But this raises a problem: the PDB database contains only sequences with known structure. This implies that is a very **redundant database**. Most of the PDB proteins belong to a small number of species (human and mouse, among others) and in many cases you find the same protein repeated several times. Besides, some protein families have been crystallized several times and have many structures, while many others don’t.

Then, if you obtain a PSSM using this **biased** set of sequences, your sequence search will be biased too. This PSSM will shift your results towards those families with higher number of known structures instead of incorporating the evolutionary information about substitutions, deletions and/or insertions.

To solve this problem we need a **non-redundant database of protein sequences**. SwissProt or Uniprot are examples of non-redundant protein sequence databases. They contain a wide spectrum of protein sequences for a lot of different species. Also, they are much bigger than the PDB. The PSSMs built using these databases will represent the evolutionary information of the target protein family with increased accuracy, and this will improve the results of our sequence search.

So, to perform an unbiased sequence search we will carry out the following strategy:

Run PSI-BLAST on the SwissProt database and store the PSSM generated on the 5th iteration. This can be done with the **-out_pssm** option. Do:

```
psiblast -query target.fa -num_iterations 5
-out_pssm target_sprot5.pssm -out target_sprot_5.out
-db /mnt/NFS_UPF/soft/databases/blastdat/uniprot_sprot.fasta
```

Apply the obtained PSSB to make a sequence search in the PDB database. The obtained results will be proteins with known structure. This can be done with the option **-in_pssm**. Do:

```
psiblast -db /mnt/NFS_UPF/soft/databases/blastdat/pdb_seq -in_pssm
target_sprot5.pssm -out target_pdb_sprot5.out
```

Now we don't make iterations in the PDB database. This is because if we iterate a new PSSM matrix, this would be built with the PDB sequences and the evolutionary significance would be biased again.

Note that in the second execution of PSI-BLAST we are not including the query into the command. This is because the program is including automatically the information of the target sequence in the PSSM used to perform the search.

Other options of "psiblast" are:

- **evaluate**: E-value threshold to see output sequences (default 10)
- **inclusion_ethresh**: E-value threshold to include sequences in the PSSM (default 0.002)

To see all options by doing "psiblast -help"

Step 3: Using PSI-BLAST with alignments of ClustalW

We know that PSSM are obtained from MSA. This MSA can be obtained by PSI-BLAST or by a sequence alignment program, such as **ClustalW**. Using ClustalW enables you to specifically select the sequences whose alignment will build your PSSM. Then, you can introduce the MSA generated by ClustalW into PSI-BLAST and run a sequence search on PDB.

Using ClustalW is extremely simple. Just concatenate the sequences you want to align in fasta format into a file and then type clustalw and the name of the file, like this:

```
> clustalw file.fa
```

Now we will get the fasta sequences for a set of selected PSI-BLAST outputs. For that we will use the program **FetchFasta.pl**. This program takes as input a list file that **YOU HAVE TO GENERATE** by copying and pasting several lines of an output from a PSI-BLAST search on SwissProt (target_sprot_5.out in the tutorial). You can use "vi" or "nano", and it may be saved with a name such as file.list. For example,

the format of the file "file.list" with sequences from the database "target_sprot_5.out " looks similar to this:

```
ORC1_DROME (O16810) ORIGIN RECOGNITION COMPLEX SUBUNIT 1 (DMORC1). 380 e-105
ORC1_SCHPO (P54789) ORIGIN RECOGNITION COMPLEX SUBUNIT 1. 295 2e-79
ORC1_CANAL (O74270) ORIGIN RECOGNITION COMPLEX SUBUNIT 1. 223 1e-57
ORC1_YEAST (P54784) ORIGIN RECOGNITION COMPLEX SUBUNIT 1 (ORIGIN... 189 1e-47
ORC1_KLULA (P54788) ORIGIN RECOGNITION COMPLEX SUBUNIT 1. 179 1e-44
CC18_SCHPO (P41411) CELL DIVISION CONTROL PROTEIN 18. 115 2e-25
CC6_YEAST (P09119) CELL DIVISION CONTROL PROTEIN 6. 87 8e-17
YPZ1_METTF (P29570) HYPOTHETICAL 40.6 KDA PROTEIN (ORF1'). 60 1e-08
YPV1_METTF (P29569) HYPOTHETICAL 40.7 KDA PROTEIN (ORF1). 60 1e-08
SIR3_YEAST (P06701) REGULATORY PROTEIN SIR3 (SILENT INFORMATION ... 47 1e-04
G6PI_OENME (P54243) GLUCOSE-6-PHOSPHATE ISOMERASE, CYTOSOLIC (GP... 31 6.6
```

Then use the following command:

```
perl /mnt/NFS_UPF/soft/perl-lib/FetchFasta.pl -i file.list
-d /mnt/NFS_UPF/soft/databases/blastdat/uniprot_sprot.fasta -o file.fasta
```

Now, we need to obtain the sequence alignment of these sequences plus the target sequence. First we have to concatenate all fasta sequence in one file. **DO NOT FORGET TO INCLUDE THE TARGET SEQUENCE.** The target sequence has to be included in the file because is required to indicate the position of the residues in the PSSM. It is important that the first sequence in the file is the target sequence.

This can be done by running:

```
cat target.fa > pssm.fasta
```

```
cat file.fasta >> pssm.fasta
```

Then run clustalw:

```
clustalw2 pssm.fasta
```

Then, change the format of the alignment to fasta format:

```
perl /mnt/NFS_UPF/soft/perl-lib/aconvertMod2.pl -in c -out f
<pssm.aln>pssm.fa
```

Then, we introduce this MSA as input to PSI-BLAST using the option **-in_msa** with this file:

```
psiblast -in_msa pssm.fa -out target_pdb_specific.out -db
/mnt/NFS_UPF/soft/databases/blastdat/pdb_seq
```

QUESTIONS FROM THE TUTORIAL

Now we can compare all the results and answer the following questions:

- 1) Why are the e-values different in *target_pdb.out* than in the fifth iteration in *target_pdb_5.out*?
- 2) Why do we need to run psiblast with uniprot_sprot.fasta before searching in pdb_seq?
- 3) When obtaining the file *target_pdb_sprot5.out* why we didn't run 5 iterations as before?
- 4) Search in the SCOP database with the PDB code of the best match of the target sequence. Do all the files *target_pdb_specific.out*, *target_pdb_sprot5.out*, *target_pdb_5.out* and *target_pdb.out* produce the same result?
- 5) Can you use the file *target_sprot5.out* to obtain the name of the fold in SCOP? Why?
- 6) What are the folds of the following sequences?
 - a. *problem1/serc_myctu.fa*
 - b. *problem2/p72_mycmy.fa*
 - c. *problem3/lip_staaau.fa*
 - d. *problem4/orc1_human.fa*

Practice 2: Search for homologous proteins using hidden markov models and the HMMer package

Theoretical concepts:

Hidden Markov Models (HMMs): In the previous practice we found similar sequences to our target protein using BLAST and PSI-BLAST. These programs perform and score alignments using substitution matrices as statistical models. In this practice we will identify similar sequences to our target protein using the programs from the HMMer package. The statistical models that these programs use are Hidden Markov Models (HMM).

Consider a HMM as a model that produces, one by one, amino acids. These amino acids are produced according to the probabilities contained in the HMM. At the end, the probability of a HMM producing a protein sequence is the product of all the individual probabilities of producing its amino acids in the correct order.

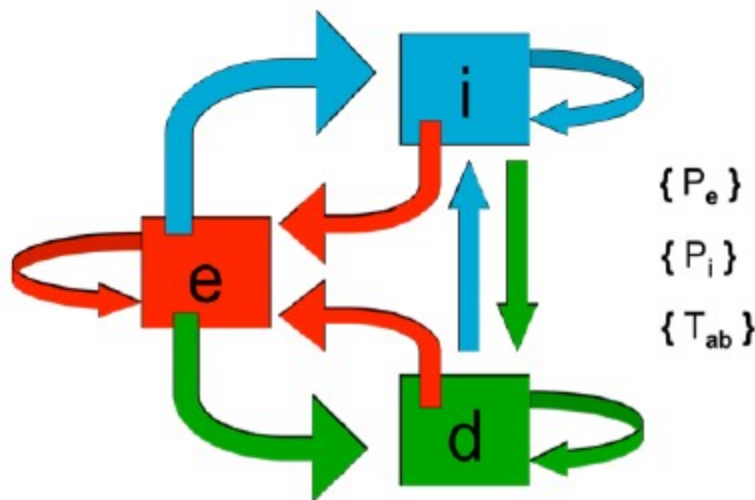
Besides, HMMs have different states. Each state produces amino acids with different frequencies. Then, HMMs change from one state to another while they are producing the protein sequence. In particular, we will work with HMM which contain the following states:

Emission: conserved position of the sequence in reference with the ancestral sequences contained in the HMM.

Insertion: insertion of an amino acid in the sequence in reference with the ancestral sequences contained in the HMM.

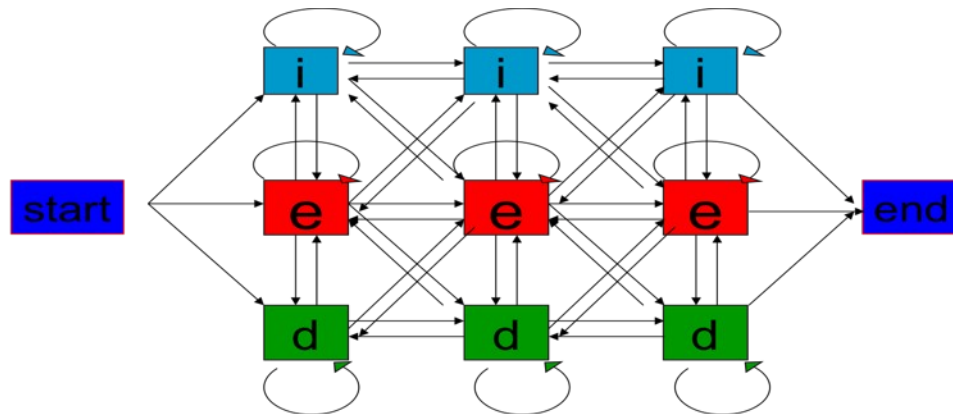
Deletion: elimination of one amino acid in the sequence in reference with the ancestral sequences contained in the HMM.

The change from state to state is determined also by the probabilities contained in the HMM.



As you can see, only two of these states produce amino acids (emission and insertion) while the deletion state doesn't. The deletion state considers the probability of having a gap, in a position of the protein, in comparison with the ancestral proteins contained in the HMM.

Probabilities of amino acid emission, insertion and deletion; as well as the probabilities of changing from one state to another are specific for each position considered. So at the end, the production of a protein sequence can be simplified as a path across different states:



One single HMM can produce the same sequence following different paths. Each one of these paths has an associated probability. This probability is the product of all the probabilities of individual events that take place during the path (inclusion of amino acids and gaps in the sequence and changing from one state to another). Finally, the probability of a HMM producing one sequence is the addition of the probabilities of the different paths that take to that common output.

All the frequencies contained in a HMM are calculated from a Multiple Sequence Alignment (MSA). Therefore, all these frequencies are the representation of the evolutionary relations between the sequences contained in that MSA. We will start this tutorial by building a HMM.

Tutorial:

Step 1: Creating a HMM using hmmbuild

To generate a HMM of a particular family of sequences we need a previous alignment of these sequences. This MSA, named seed, will be turn into a HMM by using the program hmmbuild. Here is an example of HMM usage:

➤ **hmmbuild [model_HMM] [alignment]**

The alignment has to be in STOCKHOLM format, like **globins4.sto**. You will find the required files in the folder HMMER within the directory of exercise_2. We run this as an example:

```
hmmbuild globins4.hmm globins4.sto
```

The file `globins4.sto` contains an alignment with the STOCKHOLM format, from the PFAM database. This alignment contains many sequences of globin domains aligned. Therefore, the resulting HMM will be informative for the evolutionary relationships of the globin domain. HMMs, like the previous one, that are informative for specific protein domains can also be called **profiles**.

Now you can open the file `globins.hmm` to check each column and row. Each position has the logarithm of the probability of emission of a residue. The Aa order is defined in two specific rows of the header (HMM). For each position we have probabilities on two different states (insertion and main), then we have a third row per position with the probabilities of transitions

MODEL PARAMETERS

```

HMMER3/b [3.0 | March 2010]
NAME globins4
LENG 149
ALPH amino
RF no
CS no
MAP yes
DATE Sun Mar 28 09:50:46 2010
NSEQ 4
EFFN 0.964844
CKSUM 2027839109
STATS LOCAL MSV          -9.9014  0.70957
STATS LOCAL VITERBI     -10.7224  0.70957
STATS LOCAL FORWARD     -4.1637  0.70957
HMM
      A      C      D      E      F      G      H      I      ...      W      Y
      m->m  m->i  m->d  i->m  i->i  d->m  d->d
COMPO 2.36553 4.52577 2.96709 2.70473 3.20818 3.02239 3.41069 2.90041 ... 4.55393 3.62921
      2.68640 4.42247 2.77497 2.73145 3.46376 2.40504 3.72516 3.29302 ... 4.58499 3.61525
      0.57544 1.78073 1.31293 1.75577 0.18968 0.00000 *
      1 1.70038 4.17733 3.76164 3.36686 3.72281 3.29583 4.27570 2.40482 ... 5.32720 4.10031 9 - -
      2.68618 4.42225 2.77519 2.73123 3.46354 2.40513 3.72494 3.29354 ... 4.58477 3.61503
      0.03156 3.86736 4.58970 0.61958 0.77255 0.34406 1.23405
...
      149 2.92198 5.11574 3.28049 2.65489 4.47826 3.59727 2.51142 3.88373 ... 5.42147 4.18835 165 - -
      2.68634 4.42241 2.77536 2.73098 3.46370 2.40469 3.72511 3.29370 ... 4.58493 3.61418
      0.22163 1.61553 * 1.50361 0.25145 0.00000 *
//

```

The columns for A C D etc. are the values to score the probabilities of the residues Ala, Cys, Asp etc. in the main state (first row) or insertion state (second row). The third row is for state transition scores (see section 5 of the UserGuide manual HMMER3.0). The COMPO row refers to average scores for all residues. Next two rows refer to the BEGIN state (emissions and transitions, respectively). The END state is defined by LENG, that indicates the last state of the model.

As we see, HMMs are composed by collections of position specific probabilities. Then, another way to understand HMM is to consider them as connected Position Specific Substitution Matrices (PSSMs), each one belonging to a different state, plus the probabilities of having deletions and the probabilities of changing from one state to another.

Step 2: Searching for similar sequences using `hmmsearch`

Sequence search with HMM is based on finding sequences that fit with that model. In other words, finding those sequences that would be generated by the HMM with high probability.

This can be done using **`hmmsearch`**. Here is a usage example:

➤ **hmmsearch (options) [model_HMM] [database] > [output]**

In our example, we can search for sequences with known structure fitting with the family of globins:

```
hmmsearch globins4.hmm /mnt/NFS_UPF/soft/databases/blastdat/pdb_seq
> globins_pdb.out
```

The results are sequences sorted by E-value. As in BLAST, this statistic informs of the probability of obtaining a match with that score just by chance.

Besides searching similar sequences in a database, we can also search a domain **within a single sequence** with hmmsearch. By doing this we will identify the regions within the sequence that fit in our HMM. Run the following commands:

```
hmmbuild fn3.hmm fn3.sto
```

```
hmmsearch fn3.hmm 7LESS_DROME.fa > fn3.out
```

The file 7LES_DROME only contains one sequence; therefore only one E-value is retrieved. Nevertheless, the HMM profile finds several domains within this sequence. The output looks like this:

```
>> 7LESS_DROME RecName: Full=Protein sevenless; EC=2.7.10.1;
# score bias c-Evalue i-Evalue hmmfrom hmm to alifrom ali to envfrom env to acc
---
1 ? -1.3 0.0 0.17 0.17 61 74 .. 396 409 .. 395 411 .. 0.85
2 ! 40.7 0.0 1.3e-14 1.3e-14 2 84 .. 439 520 .. 437 521 .. 0.95
3 ! 14.4 0.0 2e-06 2e-06 13 85 .. 836 913 .. 826 914 .. 0.73
4 ! 5.1 0.0 0.0016 0.0016 10 36 .. 1209 1235 .. 1203 1259 .. 0.82
5 ! 24.3 0.0 1.7e-09 1.7e-09 14 80 .. 1313 1380 .. 1304 1386 .. 0.82
6 ? 0.0 0.0 0.063 0.063 58 72 .. 1754 1768 .. 1739 1769 .. 0.89
7 ! 47.2 0.7 1.2e-16 1.2e-16 1 85 [ 1799 1890 .. 1799 1891 .. 0.91
8 ! 17.8 0.0 1.8e-07 1.8e-07 6 74 .. 1904 1966 .. 1901 1976 .. 0.90
9 ! 12.8 0.0 6.6e-06 6.6e-06 1 86 [ 1993 2107 .. 1993 2107 .. 0.89
```

This shows the E-values of the domains: C-Evalue is the significance of matching the domains under the condition that the sequence is a member of the family defined by the HMM profile, I-Evalue is the independent E-value showing the significance of the domain within the whole set of sequences of the searched database (in this case there is only one sequence, so C-Evalue and I-Evalue coincide). The rest of columns indicate the residues for starting and ending the alignment of the profile and the sequence.

Finally, the alignment between sequence and profile is reported:

```
== domain 2 score: 40.7 bits; conditional E-value: 1.3e-14
---CEEEEEEECTTEEEEEEE--S..SS--SEEEEEEEETTCCGCEEEEEETTSEEEEEES--TT-EEEEEEEEETTEE.E CS
fn3 2 saPenlsvsevtstsltsWspkdgggpigtYevyqekgegewqevtvprrttstltgLepgteYefrVqavngagegp 84
saP ++ + ++ l ++W p + +gpi+gY+++++++ + e+ vp+ s+ +++L++gt+Y++ + +n++gegp
7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQIQAGTNYTLALSMINKQGE 520
789999999999*****9998.*****9997 PP
```

The alignment shows the secondary structure of the profile, the model (fn3) and the sequence aligned (7LESS_DROME) with the matches between both, as in PSI-BLAST. The bottom line shows the % of expected accuracy of the match (7 is 65-75%, 8 is 75-85%, 9 is 85-95%, * is 95-100%)

Step 3: Using HMM databases: **hmmcompress** and **hmmsearch**

Consider that we want to perform a search for similar sequences to a target sequence using **hmmsearch**. Then we need to get one HMM that fits with this target sequence, and then use it to perform the search. Many HMM are available in several databases in the internet, but which one should we choose? Having a target sequence, can we have a method to assign the best HMM? Yes, this method is **hmmsearch**.

To learn how it works, we will start by creating a database of profiles. How can this database be generated? The initial MSAs of the families are required. These are named seed-MSAs. As an example, we use the alignments of fn3.sto (transcription factor FN3) and pkinase.sto (one family of kinases) as seeds. Create the HMMs for these alignments running:

```
hmmbuild fn3.hmm fn3.sto
```

```
hmmbuild Pkinase.hmm Pkinase.sto
```

Then, concatenate all the generated HMMs in one file:

```
cat globins4.hmm fn3.hmm Pkinase.hmm > minifam
```

In order to check sequences and profiles very fast, we compress and index the database file using **hmmcompress**. Here is a usage example:

➤ **hmmcompress [database]**

We run then:

```
hmmcompress minifam
```

Now we can search what is the best profile for a given target sequence using the command **hmmsearch**. Here is a usage example:

➤ **hmmsearch (options) [Database_HMM] [sequence] > [output]**

For example we can use the sequence of 7LESS_DROME to search for the best profile in the database previously generated. Run:

```
hmmsearch minifam 7LESS_DROME.fa > 7LESS_DROME_minifam.out
```

When looking at the output we can see that the target sequence may fit with more than one profile and more than once with the same profile. This is because sequences can have more than one domain, and in fact each profile is considered a domain, formed by similar sequences.

Step 4: Performing multiple sequence alignments using **hmmalign**

In addition, the alignment of several sequences can also be done using the HMM profile. This represents a relevant improvement with respect to the alignment of sequences, such as **clustalw**, for two reasons: 1) the control of gaps and substitutions is referred to the initial seed alignment, and this can be guided by the user (i.e. using structural and/or functional knowledge about the family); and 2) it is faster than any agglomerative approach (i.e. **clustalw**), as it aligns all sequences with the profile at the same time. The program is **hmmalign**, here is a usage example:

- **hmmalign [model_HMM] [file_with_sequences] > [output]**

We can show this with the file **globins45.fa**. Run the following commands and test the speed of both approaches, **hmmalign** and **clustalw**:

```
hmmalign globins4.hmm globins45.fa > globins45_hmm.sto
```

```
clustalw globins45.fa
```

Programs of the HMMer package work with alignments in Stockholm format. If you want to transform this format into **clustalw** format and back, you can use the script “**aconvertMod2.pl**”, it allows us to change several alignment formats. The format of the **globins45_hmm.aln** file contains additional rows showing the accuracy of the alignment. This can be transformed into **clustalw** format running the script:

```
perl /mnt/NFS_UPF/soft/perl-lib/aconvertMod2.pl -in h -out c  
<globins45_hmm.sto>globins45_hmm.clu
```

Step 5: Looking for homolog sequences using **phmmer** and **jackhmmer**

A straightforward way to find homolog sequences in a database is to use **phmmer** and **jackhmmer**. These are the equivalents to **BLAST** and **PSI-BLAST** (respectively) in the **HMMER3** package. While **PSI-BLAST** is the iterative version of **BLAST**, **jackhmmer** is the **iterative** version of **phmmer**.

The commands to run these programs are:

- **phmmer [target_sequence] [database_of_sequences] > [output]**
- **jackhmmer [target_sequence] [database_of_sequences] > [output]**

The output has the same format as for `hmmsearch`, and the input are sequences with FASTA format. The difference between `phmmer` and `jackhmmer` is that `jackhmmer` perform several iterations (up to a maximum of five by default, which can be changed using the `-N` flag) and it generates internally a HMMER profile at each iteration, while `phmmer` is like `jackhmmer` at iteration 1. Iterations are shown in the output entitled by "Round" (search this word in the file to compare the results at different iterations). The internal HMMER profile is used for the search (like with `hmmsearch`) and the profile is generated with the first sequences matched with the target and taken from the database. We can show this with the file `globins45.fa` as database, and a globin sequence (i.e. `hbb_human`) as target sequence.

```
jackhmmer hbb_human.fa globins45.fa > globins_jackhmmer.out
```

```
phmmer hbb_human.fa globins45.fa > globins_phmmer.out
```

The problem of using `jackhmmer` is that if the database of sequences where the search is performed is small or redundant, the profiles are biased (i.e. searching in "pdb_seq"). We already tackled this problem in "TUTORIAL Step 2" of practice 2.1. Do you remember how do we solve this problem?

Step 6: Using HMMs from the PFAM database

PFAM is a database that contains the profiles of several known families of sequences (see BOX, at the end of the tutorial). A good strategy would be to find the best HMMER profile in PFAM for our target sequence. We will select the best profile of your choice and fetch it from the database. Then, we will run a search of sequences in `pdb_seq` using this profile. These options are done with the commands **hmmscan** (to scan in PFAM) and **hmmfetch** (to fetch a profile from PFAM).

Step 6.1) Assign the best profile(s) to the target sequence (i.e. `hbb_human`) using **hmmscan**:

```
hmmscan /mnt/NFS_UPF/soft/databases/pfam-3/Pfam-A.hmm hbb_human.fa  
> hb_human_db.out
```

Then, we have to inspect the output of `hmmscan` to find which HMM from PFAM fits the best with our target sequence. Once we have find it, we must copy the name that this HMM has in PFAM. We will use this name to fetch the HMM from PFAM using the `hmmfetch` program.

```
# hmmscan :: search sequence(s) against a profile database
# HMMER 3.0 (March 2010); http://hmmerr.org/
# Copyright (C) 2010 Howard Hughes Medical Institute.
# Freely distributed under the GNU General Public License (GPLv3).
#
# query sequence file:          hbb_human
# target HMM database:         /cursos/BE/databases/pfam/Pfam-A.hmm
#
Query:      HBB_HUMAN [L=146]
Description: Human beta hemoglobin.
Scores for complete sequence (score includes all domains):
--- full sequence ---   --- best 1 domain ---   -#dom-
E-value  score  bias    E-value  score  bias    exp  N  Model      Description
-----
6.1e-30  103.6   0.0    4.5e-29  100.8   0.0    1.9  2  Globin      Globin
----- inclusion threshold -----
0.12   11.9   0.7     0.35   10.4   0.0    2.1  2  BCA_ABC_TP_C Branched-chain amino acid ATP-binding cassette
```

Next, we need to know the HMM of the profile(s) assigned to our target sequence. They can be extracted from the PFAM database using the program **hmmfetch**, here is a usage example:

➤ **hmmfetch [database_HMM] [name_HMM] > [file_HMM]**

Therefore, in our example, assuming we have found a domain_target:

Step 6.2) extract the profile(s) from PFAM that correspond to the domains of the target sequence which are found in the column indicated as “model” (see example in step 3 of this tutorial). Let’s assume the name of the model we have found for hbb_human is “domain_hbb”, then we execute the command:

```
hmmfetch /mnt/NFS_UPF/soft/databases/pfam-3/Pfam-A.hmm "domain_hbb"
> domain_hbb.hmm
```

Step 6.3) Search for sequences with known structure that contain the same domain as our target using **hmmsearch**:

```
hmmsearch domain_hbb.hmm /mnt/NFS_UPF/soft/databases/blastdat/pdb_seq
> hbb_pdb_by_HMM.out
```

Step 7: Some format changes

Additionally, we have seen how to use aconvertMod2.pl to transform a STOCKHOLM format in CLUSTALW. We also need other programs and scripts to transform some of these formats:

- 1) Transforming a CLUSTALW format of a MSA (Multiple Sequence Alignment) in a STOCKHOLM format. To do this transformation first we need to transform into a FASTA format the alignment using aconvertMod2.pl:

```
perl /mnt/NFS_UPF/soft/perl-lib/aconvertMod2.pl -in c -out f
<alignment.aln>alignment.fa
```


Then we transform the FASTA alignment to STOCKHOLM:

```
perl /mnt/NFS_UPF/soft/perl-lib/fasta2sto.pl alignment.fa > alignment.sto
```

We can also transform a STOCKHOLM format MSA file in a FASTA MSA file:

```
perl /mnt/NFS_UPF/soft/perl-lib/sto2fasta.pl -g alignment.sto > alignment.fa
```

- 2) Transform HMMER3.0 format files in HMMER2.0 format files (old) and ASCII to binary and viceversa.

```
hmmconvert [options] [name_HMM] > [name2_HMM]
```

- 3) Or generate a set of sequences derived from a profile

```
hmmemit [options] -N #number [name_HMM] -o [output]
```

Where the file output contains the generated sequences in FASTA format and #number is the number of sequences to be generated

QUESTIONS FROM THE TUTORIAL

- 1) Compare the results of phmmer, jackhmmer with the results of hmmsearch using "domain_hbb.hmm" (see hbb_pdb_by_HMM.out) when searching homologs in pdb_seq for hbb_human.
- 2) If a protein sequence has more than one domain in PFAM, do you think the result of using hmmsearch and jackhmmer will be the same? Why? Test the example with 7LES_DROME in SwissProt.
- 3) In practice 2.1 we used PSI-BLAST to fish sequences in the database uniprot_sprot.fasta and generate a PSSM profile which was used for searching homologs in PDB. Check the manual of HMMER3.0 and create your own protocol in which you use the program jackhmmer in a similar approach: use SwissProt database to generate the HMM profile and perform the search in pdb_seq.
- 4) Use hmmscan to search the best model(s) for 7LES_DROME in PFAM and search the homologs in PDB with this/these model(s). Compare the results of this search with the results of your protocol search in question 3. What are the differences? Why?
- 5) Use your protocol described in question 3 to search homologs of 7LES_DROME in PDB and compare with the results of the protocol described in practice 2.1 when using PSI-BLAST.
- 6) Use the sequence target.fa from practice 2.1. Apply phmmer, jackhmmer and the protocols of questions 3 and 4 to find homologs in PDB. What's the fold of this sequence? Compare the result with the homologs found in practice 2.1
- 7) Use hmalign and FetchFasta.pl to align the sequence of target.fa and its homologs of PDB
- 8) If you have to align the sequence 7LES_DROME and its homologs of PDB what's the best model to use? Produce the alignment with the models from question 4 and your protocol in question 3 to show your answer.
- 9) What are the folds of the following sequences?
 - a. *problem1/serc_myctu.fa*
 - b. *problem2/p72_mycmy.fa*
 - c. *problem3/lip_staau.fa*
 - d. *problem4/orc1_human.fa*

Exercise 3: Protein superimposition

We propose the following problem: we have two protein structures and we want to know how different they are structurally. In this situation we are not interested in sequence similarity, we only want to know if the spatial conformation of the two proteins is similar. Our hypothesis is that homologous proteins share a similar structure, although they may have different sequences. Remember that protein structure is more conserved across evolution than sequence. The way to assess structural similarity between proteins is to perform protein superimpositions.

By Alberto Meseguer

Supervised by Baldo Oliva and Altair C. Hernandez

Last update: January 25th, 2024

Theoretical concepts:

Protein superimposition: Protein superimposition is the procedure by which two protein structures are placed in space minimizing the distance between the backbone atoms of both structures. This involves that one of the two structures is rotated and translated in space, so that fits as good as possible the coordinates of the other structure. Once two proteins are superimposed we can quantify how different the two structures are. The measurement we use to quantify how different two structures are is the Root-Mean-Square deviation.

RMSD (Root-Mean-Square deviation): It is a measurement of the average distance between two sets of atoms. When working with proteins, these are the atoms of superimposed proteins. This means that one of the two proteins has been rotated and translated to minimize the distance between the two proteins. In particular, we work with the atoms from the protein backbone. The RMSD is defined by the next formula:

$$\text{RMSD}(\mathbf{v}, \mathbf{w}) = \sqrt{\frac{1}{n} \sum_{i=1}^n ((v_{ix} - w_{ix})^2 + (v_{iy} - w_{iy})^2 + (v_{iz} - w_{iz})^2)}$$

Where V and W are the proteins that will be compared, n is the number of atom pairs that will be compared, (v_{ix}, v_{iy}, v_{iz}) are the coordinates of one atom of V and (w_{ix}, w_{iy}, w_{iz}) are the coordinates from one atom of W superposed with V using a linear application of rotation and translation. To compute the RMSD between proteins V and W we will use the coordinates of the backbone atoms of residues that are as close as possible in space after the superposition.

Structural alignment: in sequence alignments, equivalent residues are those that fill the same position in the alignment according to a sequence similarity criteria. In structural alignments, equivalent residues are those that are the closest in space. We can represent structural alignments by using the formats that we already know (clustal, fasta, stockholm).

Structural alignments contain information regarding the substitutions that take place among residues that are placed in specific locations of the protein. Therefore, they also contain evolutionary information. Since protein structure is more conserved than sequence

across evolution, structural alignments can be a reliable source of evolutionary data regarding distant homologous proteins.

We can transform structural alignments into position-specific substitution matrices (PSSM) or hidden markov models (HMM), just as we saw during the exercises 2.1 and 2.2, respectively. By doing so, we will obtain statistical models that contain the evolutionary relations between residues that play the same role in protein structure, for a set of homologous proteins.

Tutorial:

There are many programs that perform protein superimposition (XAM, STAMP, TMalign,...). Some of them, like chimera or pymol, are programs for protein visualization as well. In this practical we are going to use pymol. Pymol is a program for protein structure visualization that can superimpose structures. Besides this, pymol has much more available tools, which makes it a very versatile program.

If you want to download and/or install pymol, you can do it following the next link: <https://pymol.org/2/>

Pymol is used by a big community. This makes possible that we have an extense documentation and lots of tutorials for pymol. To solve any doubts you may have working with pymol, check the pymolwiki: https://pymolwiki.org/index.php/Main_Page. In this page you have access to the documentation of pymol as well as many other resources such as tutorials. Also, there are many pymol tutorials in youtube, in particular this list was very useful to me: https://youtube.com/playlist?list=PLUMhYZpMLtal_Z7to3by2ATHP-cl4ma5X

In this tutorial we are going to work with the structures of globins. Globins are proteins that are responsible for oxygen transport. To do so, they carry within them an hemo group, that allows oxygen binding. Globin monomers can associate creating homotetramers (protein complexes created by 4 identical subunits). We can take a look to these proteins by using pymol, just open pymol and open one of the structures you wave in the directory in this practical:

file > open > select files

Or you can just use the fetch command and download a structure from the PDB to your pymol session:

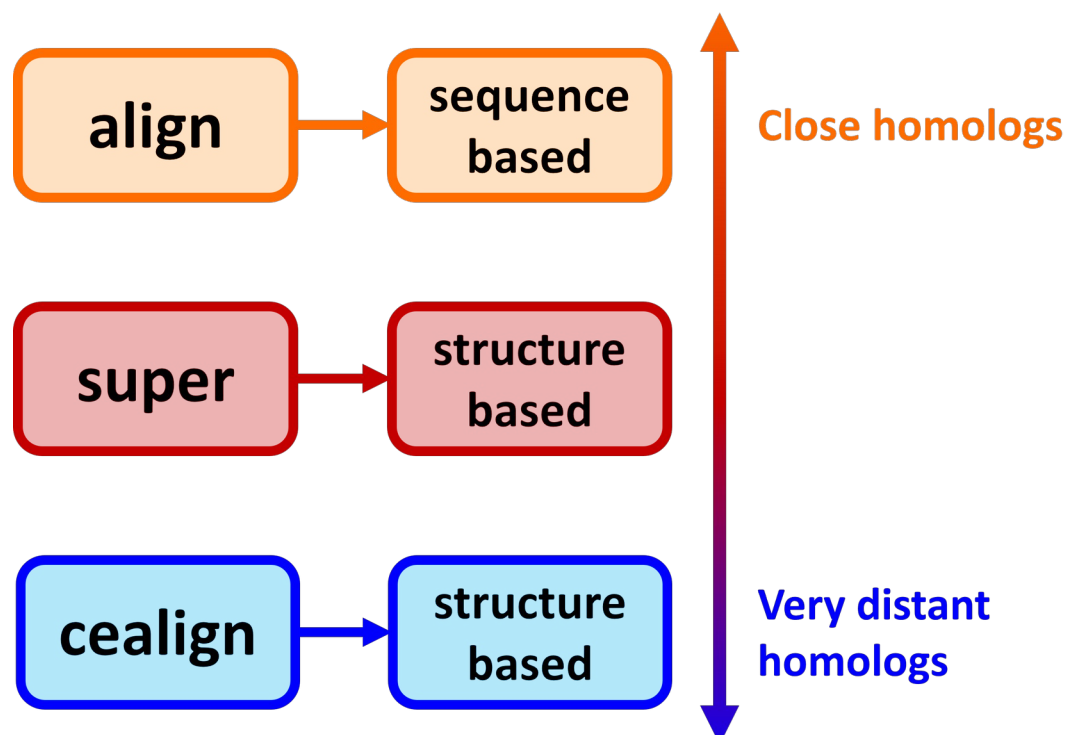
fetch lecd

Step 1: Superimpose 2 structures

Pymol has 3 different commands that can be used to perform protein superimposition: align, super and cealign. Each of these commands performs superimposition using a different algorithm and they perform better or worse depending on the similarity of the proteins you want to superimpose.

- **Align:** It performs a sequence alignment between the two proteins. Then, it matches the two structures by putting close in space the pairs of amino acids that were aligned based on sequence information. This method works well when the proteins that we are working with are close homologs, because align requires the two proteins to have similar sequences.
- **Super:** It superimposes two structures without using sequence information. It just tries to minimize the RMSD between the two proteins. This method works well with proteins that are distant homologs. Distant homologs have different sequences but similar structures, and since super only uses structural information, it just needs structural similarity to perform well.
- **Cealign:** It superimposes two structures without using sequence information. It just tries to minimize the RMSD between the two proteins, but using a different algorithm than super. Cealign works well with proteins that are very distant homologs. In such cases, sequence and structure can be quite different between the proteins we are working with.

It is clear to see that each method is intended for proteins within an homology range. So, although cealign can superimpose very distant proteins, for close homologs the command performing the best will be align. You can see the properties of these different commands in the following image:



In this tutorial we are going to work with the super command, since the proteins that we will superimpose have similar structures, but quite different sequences. The syntax to use these three commands is the same, so if you know how to use one of them you can use the three of them. Here you have the links to the documentation for each of these commands:

Align: <https://pymolwiki.org/index.php/Align>

Super: <https://pymolwiki.org/index.php/Super>

Cealign: <https://pymolwiki.org/index.php/Cealign>

Start by loading in your pymol session the pdb structures 4mbn and 1ecd (you can load them from the directory of the practical or fetch them from the PDB)

fetch 4mbn

fetch 1ecd

Remove all the water molecules, since they could interfere with the superimposition:

remove resn hoh

Use the super command to superimpose the two structures:

super 4mbn, 1ecd, object=aln2

See that the syntax involves putting first the pdb object that will move, then the pdb object that will remain fixed, and finally the name that we want to give to the superimposition object.

After executing this command you will have the RMSD value for this superimposition in your pymol console:

```
File Edit Build Movie Display Setting Scene Mouse Wizard Plugin Help
ExecutiveAlign: 775 atoms aligned.
ExecutiveRMS: 23 atoms rejected during cycle 1 (RMSD=2.77).
ExecutiveRMS: 12 atoms rejected during cycle 2 (RMSD=2.58).
ExecutiveRMS: 5 atoms rejected during cycle 3 (RMSD=2.51).
ExecutiveRMS: 1 atoms rejected during cycle 4 (RMSD=2.49).
ExecutiveRMS: 1 atoms rejected during cycle 5 (RMSD=2.48).
Executive: RMSD = 2.476 (733 to 733 atoms)
Executive: object "aln2" created.
```

Besides, we are giving a name to the superimposition object that we are creating in this execution. Giving a name to this superimposition has the advantage that it enables us to

represent it as an alignment. This alignment is based in structural similarity, remember that we are using the super command. Here the criteria that we use to determine what amino acids go in the same column is to minimize the RMSD between the two structures. Therefore, this alignment that we get is based on structural similarity and not sequence similarity (like the ones we saw on previous practicals).

You can visualize this alignment if you click on the S button on the bottom right corner of your pymol screen:



This will display the alignment in the section of the screen where we can visualize protein sequences:

```

PyMOL>
/4mbn/4MBN//1      6   11  16  21  26  31  36  41  46  51  56  61  66  71  76  81  86  91  96  101 106 111 116 121 126 130 136 141 146 151 216 224
V LSEGEWQLVLHVMKVEADVAGHGQDILIRLFKSHPETLEKFDKFKHLKTEAREMKASEDLKKGVTYLTALGAILKKKGHEAELKPLAASHATKHKIPIKYLEFISEATIHVLHSHRHPGDFGADAGGA HEMNKALELFRKDIKAKYKELGYQG S04 S04
/1ecd/1ECD//1      1    6   11  16  21  26  31  36  41  46  51  56  61  66  71  76  81  86  91  96  101 106 111 116 121 126 131 136
HEM LS---ADQISTVQASFDKVK-GDPVGILYAVFKADPSIMAKFTQFAGK-DLESIKGTAPFETHANRIVGFFSKIIGELPNIEADVNTFVASHK-PRGVTHDQLNFRAGFVSVMKAHTDFAGAERAWGA ---TLDTFFGMIFSKM-----
  
```

See that you can select groups of amino acids in this sequence/alignment display, which is very useful. Finally, we can store this alignment as a separate file in clustalw format using the save command:

save aln2.aln, aln2

See that the syntax of the command is save, then the name of the output file, and finally the name of the object that contains the alignment (aln2 in this case). This command will save the alignment in your pymol directory.

Step 2: Superimpose as many structures as you want

In this next step we are going to superimpose 4 globin proteins. These 4 proteins are in the P3_directory or you can get them from the PDB using the fetch command. They are 1ecd.pdb, 1lh1.pdb, 2lhb.pdb and 4mbn.pdb.

Remember to remove the water molecules:

remove resn hoh

To superimpose many structures together you need to set one structure as reference (1ecd) and superimpose the rest of proteins on top. To do this you will have to use the

super command several times. See that, since we want all the 4 proteins to be included into the same superimposition, we will force pymol to put all the superimpositions into the same object:

super 4mbn, 1ecd, object=aln4

super 2lhb, 1ecd, object=aln4

super 1lh1, 1ecd, object=aln4

After doing these commands you should see that the 4 proteins are properly superimposed. Now take a look at the alignment you are getting from this superimposition, you should see something like this:

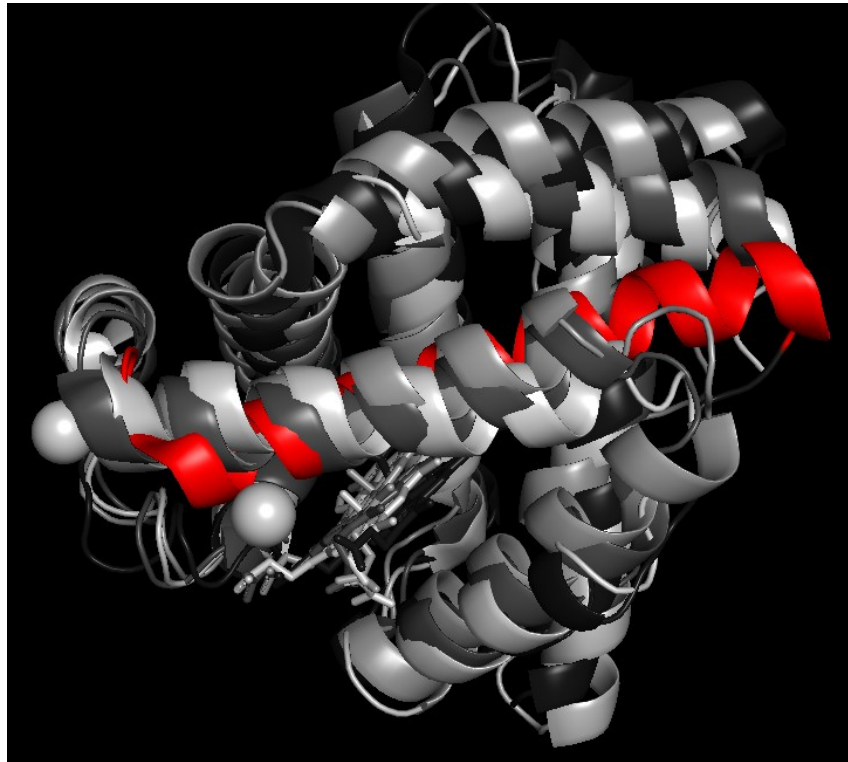
```
PyMOL>
/4mbn/4MBN/1 V L S 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/2lhb/2LHB/1 V 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/1lh1/1LH1/1 P I V DTGSVAPLS-A-REKTKIRSAWAPYST-YETSGVDILVKFFSTPAQGEFFPKF-KGLT- 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/1ecd/1ECD/1 G HEM ACE AL-----T-ESQARLVKSSUEEFNA-N-IPKHTHRFFILVLEIAPAKDLFSFLKGTSEVPNNPELQAHAGKVFKLYEARIQLEVTG- 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116
HEM --- --- ---LS-A-DQISTVQHSFDKV-K-G--DPVGILYAVFKADPSIMAKTQF--AGK-----DLESIK-G--TAPFETHANRIVGFFSKIIG-EL-PHIEADVNTFVASHK-PRGVTHDQLNFRAGFVSVMKANT--D-F-AGAEARAG
```

Do you think that this alignment is good? Do you think that this alignment corresponds with the superimposition that you are seeing in the screen? Can you identify a region of this alignment that is not aligned with the rest of the sequences? Take five minutes to think about these questions.

In this alignment you can see that there is a big region that is not aligned with the rest of the sequences:

```
PyMOL>
/4mbn/4MBN/1 V L S 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/2lhb/2LHB/1 V 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/1lh1/1LH1/1 P I V DTGSVAPLS-A-REKTKIRSAWAPYST-YETSGVDILVKFFSTPAQGEFFPKF-KGLT- 61 66 71 76 81 86 91 96 101 106 111 116 121 126
/1ecd/1ECD/1 G HEM ACE AL-----T-ESQARLVKSSUEEFNA-N-IPKHTHRFFILVLEIAPAKDLFSFLKGTSEVPNNPELQAHAGKVFKLYEARIQLEVTG- 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116
HEM --- --- ---LS-A-DQISTVQHSFDKV-K-G--DPVGILYAVFKADPSIMAKTQF--AGK-----DLESIK-G--TAPFETHANRIVGFFSKIIG-EL-PHIEADVNTFVASHK-PRGVTHDQLNFRAGFVSVMKANT--D-F-AGAEARAG
```

If you look for this region in the superimposition you will see that is properly superimposed, but is slightly different to the other superimposed structures. Here you can see this problematic regions highlighted in red:



As you can see, this helix is not fitting with the other super well. When pymol builds the alignment based on structural information is putting a distance cutoff between the amino acids that can be paired. Since this helix is different to the others, the amino acids of the helix don't pass the distance cutoff and they become unaligned. This is an artifact that we can correct using properly the command super.

Now, we are going to force that the distance cutoff for two amino acids to be aligned is 5.0 Amstrongs, instead of 2.0 Amstrongs which is the default. To do this, restart your pymol session, load again all the proteins and use the following commands:

super 4mbn, 1ecd, object=aln4_corrected, cutoff=5.0

super 2lhb, 1ecd, object=aln4_corrected, cutoff=5.0

super 1lh1, 1ecd, object=aln4_corrected, cutoff=5.0

After doing this you should get an alignment that looks way better:

```
PyMOL>
/4mbn/4MBN//1
V L S EGE-----M-QLVLHVMKVEADY-AG--HGQDILRLFKSHPETLEKFDRF-----KHLK-TEAEHK-A--SEDLKKGVTYLTALGAILKKK-GHHEBELKPLAOSHATKHKIPKYLEFTSEATIHVLHSPH---PGDFGADADGA HEMMNAKLELFRKD---IARKYKELGYC
/2lhb/2LHB//1
P I V DTGSAVPLSA-AEKTIRSAWAPVYSTYETSGVDILVKFFTSTPADEFFPKF---KGLT-TADELKKSDVRWAERIINAVDDAVASMDDEKMSMKLNLGSKHAK-SFOVDPE--YFKVL-AAVIADTVH---AGDAGFEKL-M---SMICILLRSAY HEM
/1lh1/1LH1//1
I 2 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126 131 136 141 146 151
HEM ACE AL-----TESQAALVKSSWEFNA-NIPKHTRRFFILVLEIAPAKDLFSFKGTSEVPCHNPQLQA-H--AGKVEKLYEARIOLEVTVGV---VTRATLKNLGSVHV-SKGVADARFPVVKERILKTIKEVGAKUSEELNSAWTI---AYDELAIVIKK---EMDDAA
/1ecd/1ECD//1
HEM ---L-1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126 131 136
-----LSA-DQISTVQASFDKV--KG--DPVGILYAVFKADPSIMAKFTQF-----AGK--DLESIK-G--TAPFETHANRIVGFFSKIIGEL-PNIEADVNTFVASHK-PRGVTHDQLNFRAGFVSVMKAHT---DFAGAEAWGA---TLDTFFGMIFS---KM
```

This alignment looks way better. However, we still get gaps in it. Can you find where are these gaps happening? Can you find any pattern? Does it make sense to you?

Interestingly, most of the gaps are happening in loops. Loops usually are the most variable regions in a protein family, both in terms of structure and also sequence. Structure variability in loops is usually high because they are the most flexible parts of a protein.

Finally, you can store this alignment in clustalw format in your pymol directory by using the save command:

```
save aln4_corrected.aln, aln4_corrected
```

Step 3: Create a structure based HMM

The MSA we just created is different from any other MSA we created before. This new MSA is based on structural similarity. Also, since the proteins that we aligned are distant homologs (different sequence, similar structure). Therefore, our MSA contains structural information for distant homologs of the globin protein family. We can use this MSA to create a HMM, and then this HMM will contain structural information for distant homologs of the globin family.

To create a HMM out of this MSA the first thing you have to do is to upload the MSA file to the citrix. We recommend you to do that by using google drive.

Next, you have to transform the format of the alignment, from clustalw to stockholm, which is the only format that programs from the HMMer package can take as input:

```
perl /shared/PERL/aconvertMod2.pl -in c -out f  
<aln4_corrected.aln>aln4_corrected.fa
```

```
perl /shared/PERL/fasta2sto.pl aln4_corrected.fa > aln4_corrected.sto
```

Finally, you can execute hmmbuild to create a HMM using as input your MSA in stockholm format.

```
hmmbuild hmm_structural.hmm aln4_corrected.sto
```

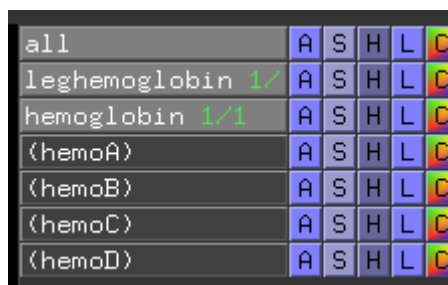
Step 4: Use superimposition to model protein-protein interactions

Homologous tend to share their protein-protein interactions as well as the conformation in which these interactions take place. This principle is at the basis of modeling protein-protein interactions using superimposition and the template of the interaction we want to model.

For this step we will work with leghemoglobin, a vegetal globin, and the tetrameric form of human hemoglobin. Leghemoglobin and hemoglobin are homologous proteins, therefore we expect that leghemoglobin can also create tetramers. The problem is that we don't know how this leghemoglobin looks like. To solve this we can use superimposition.

We know that the conformation of tetrameric leghemoglobin should be similar to the tetrameric conformation of hemoglobin. Therefore, we can superimpose the leghemoglobin monomer on top of each of the hemoglobin monomers. This will create four different sets of coordinates for the leghemoglobin monomer. If we save as a new pdb each one of these new set of coordinates we will be able to reconstruct the tetrameric structure of leghemoglobin.

Start by opening a new session of pymol and loading the pdb structures for hemoglobin leghemoglobin that you have in the P3_directory. Then, create new pymol objects for each of the chains in the hemoglobin tetramer. We will call them hemo_A, hemo_B, hemo_C and hemo_D. Using the sequence display and the chain identifiers is a good way to select and create objects for each one of the chains.



all	A	S	H	L	C
leg hemoglobin 1/	A	S	H	L	C
hemoglobin 1/1	A	S	H	L	C
(hemoA)	A	S	H	L	C
(hemoB)	A	S	H	L	C
(hemoC)	A	S	H	L	C
(hemoD)	A	S	H	L	C

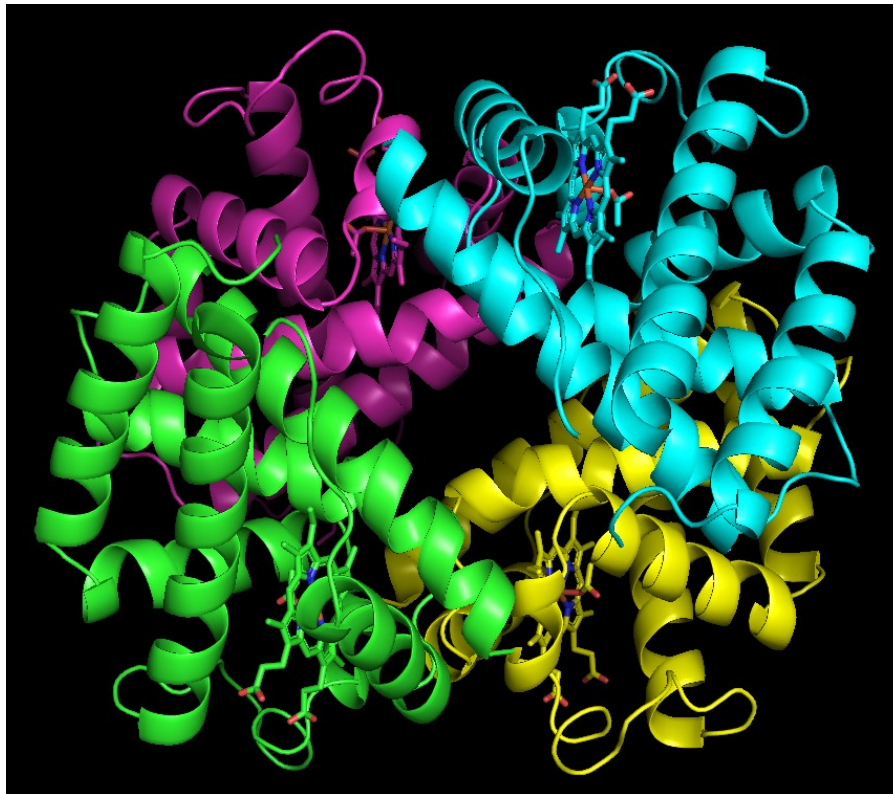
Now we will superimpose the leghemoglobin monomer on the chain A of the tetrameric hemoglobin:

super leg hemoglobin, hemoA

Once done this, save the leghemoglobin monomer in this new set of coordinates as a new pdb file:

save leg hemoglobin_A.pdb, leg hemoglobin

Now, repeat this same procedure for hemoglobin chains B, C and D. Finally, if you open all the leghemoglobin pdb files you have generated you will be able to see the tetrameric form of leghemoglobin:



Step 5: Superimpose specific regions of two proteins

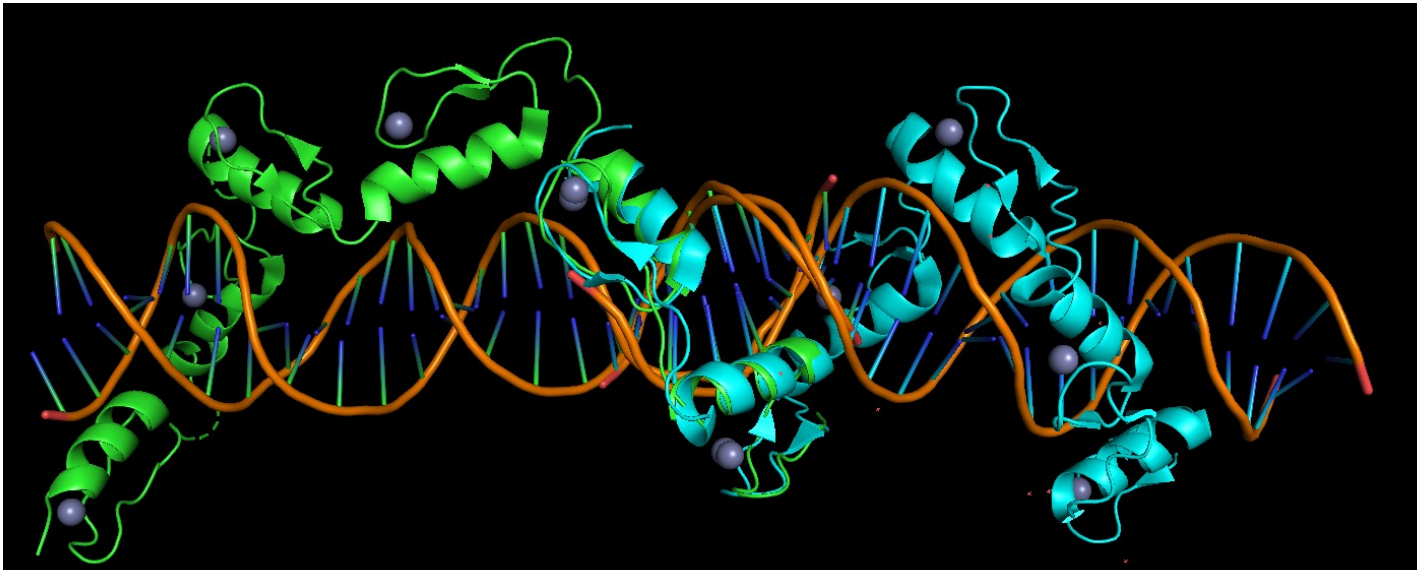
CTCF is a zinc finger transcription factor that plays a fundamental role in the regulation of gene expression. The DNA binding domain of CTCF is made of 11 zinc finger domains. Since this is a long DNA binding domain, there are no available structures for the entire DNA binding domain. However, there are structures that contain part of these DNA binding domains. By superimposing these structures on the parts that they overlap we may be able to reconstruct a model of the DNA binding site of CTCF.

Open the two structures in the CTCF directory inside the P3_directory (5t0u.pdb and 5yel.pdb). These files contain different parts of the DNA binding domain of CTCF. If you visualize the sequence of these proteins you will see that 5t0u goes from amino acid 293 to amino acid 461, while 5yel goes from amino acid 405 to amino acid 576.

We want to superimpose these two structures, but we have to be sure that we only superimpose the regions that are overlapping. To do this we have to specify what residues we want to include in the superimposition. Also, we will use the align command because the two structures that we want to superimpose belong to the same protein. Therefore, the amino acid sequences will be identical. Use the next command:

```
align 5t0u and resi 405-460, 5yel and resi 405-460
```

See that we are forcing the superimposition between amino acids 405 and 460. Since the two structures belong to the same protein, they match perfectly. See that we are superimposing also the DNA molecules that are in the structures.



Questions from the tutorial:

- 1) Get the sequences of the four compared monomeric globins (1ecd.pdb, 1lh1.pdb, 2lhb.pdb and 4mbn.pdb) and make a sequence alignment using clustalw. Then compare this alignment to the structure-based one that we obtained in the step 3 of this tutorial. Are they similar? Why is this happening?
- 2) Build a structure-based HMM using 4 templates for the sequence contained in the file "question2_target.fa". Can you use this HMM to search for homologous proteins of the target we are working with?
- 3) Build a structure-based HMM using 4 templates belonging to the globin protein family. Use the HMM from the PFAM database. Which are the differences between the obtained HMM and the ones in PFAM?
- 4) Try to identify the residues that are more relevant for the assembly of the lehmoglobin tetramer.
- 5) Try to build the tetrameric complex for some of the other globins in the tutorial (1ecd.pdb, 2lhb.pdb or 4mbn.pdb).
- 6) The same procedure we use to reconstruct protein-protein interactions via superimposition can be used to model protein-ligand interactions. In the protein-ligand directory you have two structures: A1_receptor.pdb and caffeine_receptor.pdb. The first structure contains an A1 adenosine receptor and the second contains an A2A adenosine receptor binding a caffeine molecule. Can you use superimposition to reconstruct the interaction between caffeine and the A1 adenosine receptor?
- 7) Use the commands align, super and cealign to superimpose the PDB structure 1bco with the chain B of structure 1c0m. What of the 3 commands is giving you better results? Why do you think this is happening?

Exercise 4: Modeling protein structures through comparative modeling and threading.

We propose the following problem: we want to study a protein for which we know the sequence but there is no available structure. To perform a complete study of the protein we will obtain models of the protein structure. This situation is very common because high-throughput technologies have produced a vast amount of protein sequences, while the number of high-resolution structures has seen a limited increase. In this tutorial we will learn how to obtain structural models from protein sequences using two different approaches: comparative modeling and threading.

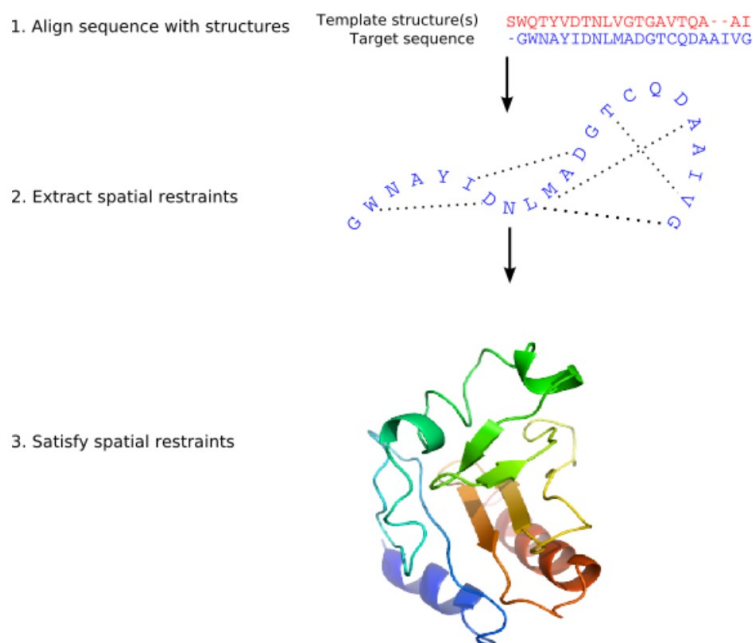
Theoretical concepts:

Comparative modeling (or homology modeling): it is the process of modeling the structure of one protein using as template/s the structure/s of homologous proteins. This process is based on the next facts:

- The number of protein folds in nature is limited. Many protein structures are obtained every year, but rarely these structures have an unknown fold.
- Proteins with similar sequences have an almost identical structure.
- Homologous proteins tend to share the same protein fold.

There are several programs to perform comparative modeling, in this tutorial we will learn how to use MODELLER. MODELLER is a program that models protein structures based on the satisfaction of spatial restrains. It works following the next steps:

1. The sequence of the target and the sequence of the template are aligned. This means that each residue of the target is assigned to one residue in the structure of the template.
2. Spatial restrains, such as Ca-Ca distances, are transferred from the templates to the target. Besides, probability density functions are obtained from the structural features of the templates.
3. A 3D model is obtained by satisfying all the spatial restrains as well as possible and optimizing probability density functions and energetic terms.



Using comparative modeling we can obtain many different models for the same target. Some of these models will be more accurate than others. These are some variables that can influence in model accuracy:

- The homology degree between the target and the template. The highest the homology, the highest the accuracy of the model.
- The quality of the alignment between the target and the template. If the residues of the two proteins are not correctly aligned the model is prone to be inaccurate.
- The optimization of the model. After applying the spatial restrains, MODELLER performs a fast simulation in order to optimize the location of all the atoms in the model. This process can produce different outputs. That is why MODELLER can produce one or more different models from the same template.

This rises the fact that we need some mechanisms to assess the quality of the generated models. During this lesson and in the following one we will explore two ways of evaluating the quality of models: the use of statistical potentials and the prediction of secondary structures.

Threading: is the process of modeling the structure of one protein by assigning to that protein an already known fold. It works by fitting a protein sequence into an already known structure. This strategy has not as good results as comparative modeling, but it can be a useful resource in case you want to model one protein for which you find no templates.

Threading programs typically have a database of protein structures. So, instead of making one model they make a lot, using their different available structures. Thus, here we have the same problem that we found in comparative modeling: we have a lot of models and we should differentiate the accurate ones from the rest. One way to do this is to use statistical potentials.

Statistical potentials: statistical potentials are probability functions derived from the analysis of a reference set of known protein structures. This function will provide good scores for the protein models that have similar structural properties to the ones in the reference set. Then, we make that the structures in the reference set are in a native state. By doing so, good scores will tell us that a protein model has similar structural features than our set of native state proteins. These good scores are telling us that our model is accurate and reliable. Keep in mind that low scores are good scores, and high scores are bad scores. So, the lowest the best.

This function is obtained by calculating the inverse of the Boltzmann law using the probabilities found in the reference set. This allows the representation of probabilities as energy values. Therefore, statistical potentials scores can also be called energies. Low scores will involve low energies and high similarity of the assessed model with the native structures in the reference set.

$$P = (1/z)(e^{(-E/kT)})$$

$$E = -KT \ln P + KT \ln Z$$

The structural properties that are taken into account by the statistical potentials that we will use today are:

- Distances between amino acids. This distance can be computed between the two closest atoms of each residue pair or between specific atoms of the amino acids. In the second case, it is common to compute the distances between the beta carbons, because by doing so we take into account the orientation of the side chains. In most programs this feature is used to compute the “PAIR” energy.
- Degree of exposure (or solvation) of amino acids: amino acids can be placed in the core or in the surface of the protein. We expect to find hydrophobic amino acids in the core and polar amino acids in the surface. In most programs this feature is used to compute the “SURFACE” energy. Solvation stands for the degree of interaction of amino acids with water molecules. The most exposed an amino acid is, the more water molecules it will interact with.

It is important to keep in mind that statistical potentials are relative measurements. This means that we cannot apply a score threshold to discriminate good from bad models. They have been intended to compare models between them or with experimentally determined structures.

A good strategy to assess whether a protein has been correctly modeled is to compare the statistical potentials energies between the model and its template. We expect the template to have a good energetic profile, since it is an experimentally determined structure. Therefore, we can use the template as a reference. If the model has higher energies than the template, it is very likely that it has been wrongly modeled. This same strategy can be used to identify wrongly modeled regions inside a model, instead of evaluating the whole model at once.

Tutorial:

Step 1: Find templates

Move to the “MODELLER” folder inside the “exercise_4” directory. We will obtain models for the sequence contained in the file “P11018.fa”. The first step in homology modeling is always to find good templates for our target protein. During exercises 2.1 and 2.2 we learnt how to use BLAST and Hidden Markov Models to do this, and we explored different strategies to find reliable templates. Today we are going to search for templates in a very simple way for the sake of time. We are going to run BLAST into the PDB database with our target sequence using the following command:

```
blastp -query P11018.fa -db ~/Documents/databases/pdb_seq  
-out P11018.out
```

From the BLAST output we can retrieve the PDB IDs of the proteins that we are interested to use as templates. We can use one or more templates to model our target protein. With the PDB IDs we can get the structures of the templates from the PDB database.

1. Go to the PDB website in: <https://www.rcsb.org/>
2. Copy the PDB ID of the template in the search window and click enter.

3. Go to the PDB entry that has the template ID. Once in there, click on the download files button. Choose to download a file in PDB format.

By default, by doing this the PDB file of the template will be stored in your downloads directory. Copy the downloaded PDB file/s into the MODELLER directory.

It may be the case that from the whole PDB file we are only interested in one of the chains. Besides, PDB files can have parts that are not well understood by other programs. To solve these two issues we use PDBtoSplitChain. This program splits one PDB file in different chains and corrects the PDB format. It also provides us the fasta sequence of each one of the chains. Run the following command for each one of the templates you want to use:

```
perl ~/Documents/perl_scripts/PDBtoSplitChain.pl -i template.pdb -o prefix
```

Step 2: Build a protein model

We are going to obtain models for our target protein using a program called modeller. To execute MODELLER you need three files:

- Target file (target.fa): contains the target sequence.
- Alignment file: contains the alignment between the target and the template/s in PIR format.
- Script file (modeling.py): contains the executing commands for MODELLER.

We already have the target file and the script file in our working directory. So we will start by creating the alignment between the target and the template/s. In previous exercises we have seen many strategies to build alignments, in particular we learnt that alignments built using Hidden Markov Models (with hmmalign) have more quality than the ones produced by clustalw or other agglomerative alignment programs. For the sake of time we will build this alignment using clustalw. In the next practical lesson we will make this alignment using Hidden Markov Models and we will compare the results.

Now, we concatenate the sequences of our target and the template/s that we want to use by running the next commands:

```
cat P11018.fa > target_template.fa
```

```
cat template.fa >> target_template.fa
```

Now, we align these sequences using clustalw:

```
clustalw target_template.fa
```

We have obtained a MSA called "target_template.aln". Now we need to modify the format of the alignment file, from clustal to pir format. We will do so by using aconvertMod2.pl:

```
perl ~/Documents/perl_scripts/convertMod2.pl -in c -out p  
<target_template.aln>target_template.pir
```

We must be very precise fulfilling all the requirements from MODELLER regarding the PIR alignment, otherwise the program will not work. MODELLER requires the target, the alignment and the script files to have the same names and codes regarding the files that will be handled. Here is an example of how these files should look like:

P11018.fa

```
>P11018
```

```
MNGEIRLIPYVTNEQIMDVNELPEGIKVIKAPEMWAKGVKGKNIKVAVLDTGCDTSHPD  
LNQIIIGGKNFTDDDDGGKEDAIISDYNHGHTHVAGTIAANDSNGGIAGVAPEASLLIVKVLG  
GENSGQYEWIINGINYAVEQKVDIISMSLGGPSDVPELKEAVKNAVKNGLVVCAGNE  
GDGDERTEELSYPAAYNEVIAVGSVSVARELSEFSNANKEIDLVPGENILSTLPNKKYG  
KLTGTSMAAPHVSGALALIKSYEEESFQRKLSESEVFAQLIRRTLPLDIAKTLAGNGFLY  
LTAPDELAEKAEQSHLLTL
```

target_template.pir

```
>P1;P11018  
structurex:sp|P11018|ISP1_BACSU:1: : : : 319 : : : -1.00 :-1.00  
MNGEIRLIPYVTNEQIMDVNELPEGIKVIKAPEMWAKGVKGKNIKVAVLDTGCDTSHPD  
LNQIIIGGKNFTDDDDGGKEDAIISDYNHGHTHVAGTIAANDSNGGIAGVAPEASLLIVKVLG  
GENSGQYEWIINGINYAVEQKVDIISMSLGGPSDVPELKEAVKNAVKNGLVVCAGNE  
GDGDERTEELSYPAAYNEVIAVGSVSVARELSEFSNANKEIDLVPGENILSTLPNKKYG  
KLTGTSMAAPHVSGALALIKSYEEESFQRKLSESEVFAQLIRRTLPLDIAKTLAGNGFLY  
LTAPDELAEKAEQSHLLTL*  
>P1;1meeA  
structurex:1meeA:1: : : : 275 : : : -1.00 :-1.00  
-----AQSVPYGISQIKAPALHSQGYTGSNVKVAVIDSGIDSSHPDL  
N--VRGGASFVP--SETNPYQDGSSHGHTHVAGTIAALNNSIGVLGVAPSASLYAVKVLD  
S-TGSGQYSWIINGIEWAISNMMDVINMSLGGPTGSTALKTVVDKAVSSGIVVAAAAGNE  
GSSGS-TSTVGYPKYPSTIAVGAVNSANQRASFSSAGSELDVMAPGVSIQSTLPGGTYG  
AYNGTSMATPHVAGAAAALILSKHPTWTN-----AQVRDR---LESTATYLGSSFY  
GKGLINVQAAAQ-----*
```

modeling.py

```

# Homology modeling with multiple templates
from modeller import *          # Load standard Modeller classes
from modeller.automodel import * # Load the automodel class

log.verbose()    # request verbose output
env = environ()  # create a new MODELLER environment to build this model in

# directories for input atom files
env.io.atom_files_directory = ['.', '../atom_files']

a = automodel(env,
               alnfile = 'target_template.pir', # alignment filename
               knowns   = ('1meeA'),           # codes of the templates
               sequence = 'P11018')            # code of the target
a.starting_model= 1      # index of the first model
a.ending_model   = 2     # index of the last model
                    # (determines how many models to calculate)
a.make()              # do the actual homology modeling

```

The relevant elements of these files are highlighted in colors:

- Target name (Red): is the name of the target protein. It should be the same in the three files. It is written without the “.fa” extension.
- Template name (Blue): is the name of the template PDB. It is written without the “.pdb” extension on the .pir and modeling.py files. Besides, the corresponding file should be in the working directory.
- Start and end amino acid positions for the target and template proteins (Green): This is included in the .pir alignment. Is necessary that the number of residues in this file fit with the number of residues in the target sequence and with the number of residues in the PDB files from the templates.
- Chains for the target and template proteins (Magenta): this is included in the .pir alignment. It is used to select the chain you want to use as template for modeling from your template PDB file. In this case, our template only has one chain, so we can leave it empty.
- Name of the .pir alignment file (Orange): this is included in the modeling.py file. It tells the program the name of the input alignment between the target and the template/s.

Although the inconsistencies between file names and paths is the major source of errors while running MODELLER, the program can also fail in these situations:

- If the template sequence used in the alignment was obtained from Swissprot rather than from the PDB. Usually the sequence at the PDB is not the complete sequence. This would involve that the sequence from the structure is different from the one in the alignment. We can solve this problem by obtaining the sequence of the template running PDBtoSplitChain.
- If the template has segments with occupancy higher than 1. This means that this template has regions where more than one amino acid are being overlapped in the same position. Although this error can be solved, it is beyond the scope of this course. In such case the best option is to use a different template.

- The PDB format can have many artifacts and errors. So, in case you are having errors and you don't know why, you can always try to change the template and see if the program works. Some of these artifacts or errors can be solved if we run PDBtoSplitChain.pl on the template pdb file.

Once we have checked that our input files are correct we can run modeller. To run MODELLER execute the following command:

mod10.5 modeling.py

If everything works fine, you will obtain the next output files:

- XXXX.B9999000N.pdb → These files are the produced models, where XXXX is the input name and N is the number corresponding to the different models generated by a MODELLER run.
- XXXX.log → This file contains the information of the whole MODELLER execution. In case that there is an error, this would be reported into the .log file. Then, by looking to the .log file we should see the error and know why MODELLER has crashed.

Step 3: evaluating our models with prosa

After obtaining one protein model we should evaluate its quality to be sure that is a reliable model. We are going to do so by using statistical potentials. In particular, we will use one program called prosa that enables us to view the energy profiles of the generated models. This program is accessible through the internet at the following direction:
<https://prosa.services.came.sbg.ac.at/prosa.php>

Since statistical potentials are relative measurements, we should compare the prosa results for our model with the results of a reference protein. Our template is an excellent reference protein because is an experimental structure (therefore we know it is correct) and it is supposed to be similar to our model. Then, what we have to do is to run two prosa web executions: one for our model and another for our template. To do this, open two prosa web windows, in one submit your model and in the other submit your template.

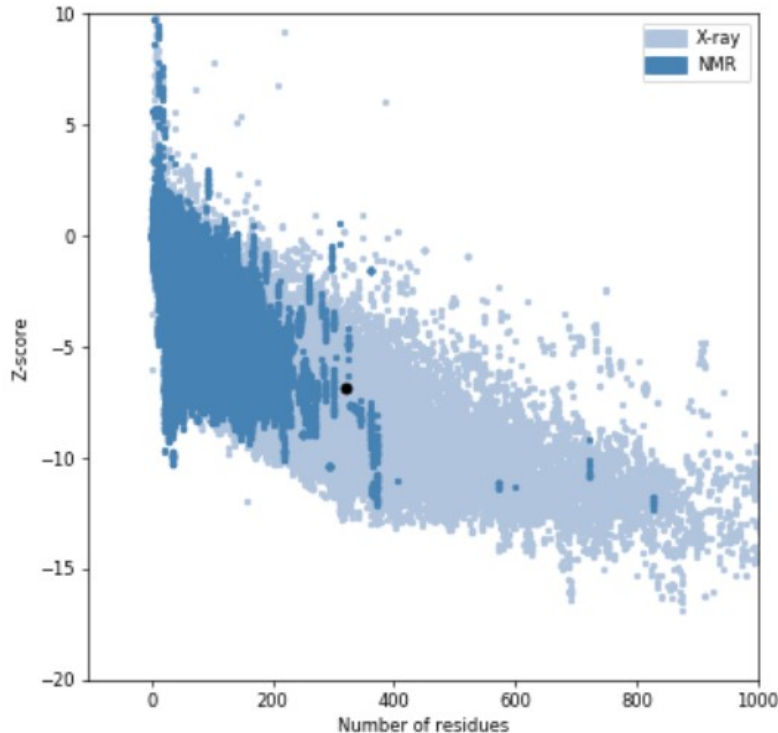
After running, prosa web will display a set of results for each analysed protein consisting of:

- A **Z-score** and a plot comparing how good is your structure in comparison with the structures in the PDB.

Overall model quality

[HELP](#)

Z-Score: **-6.81**

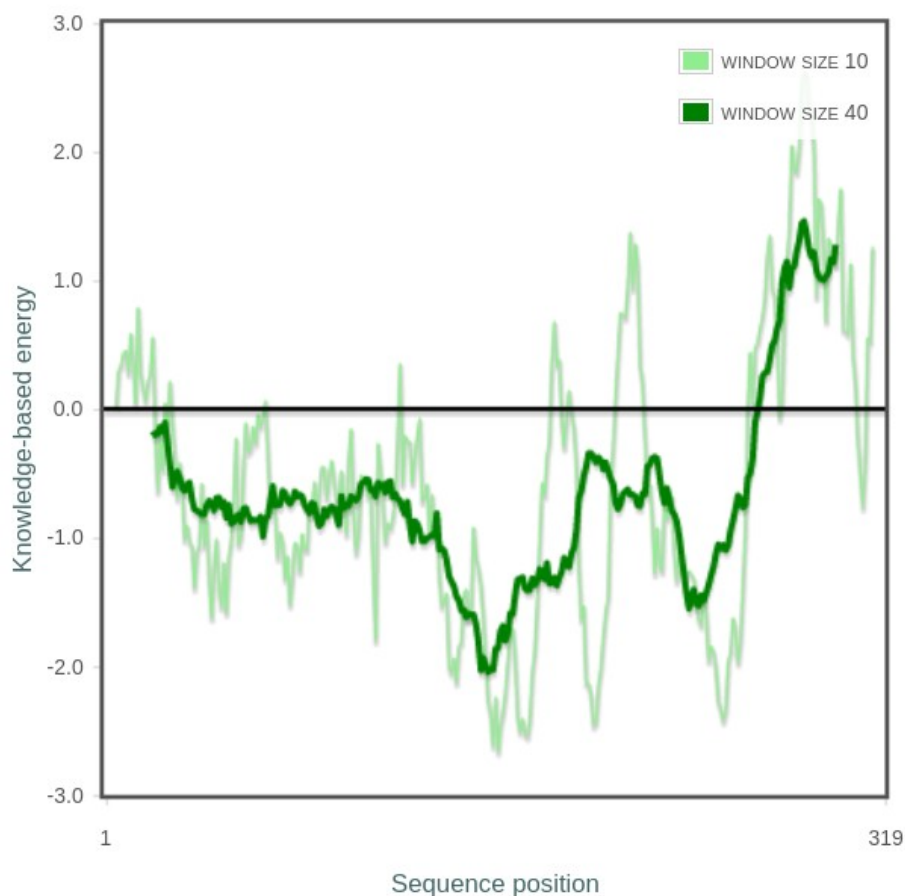


- An **energetic profile** showing the statistical potentials scores across the amino acid sequence. On the X axis we have the protein sequence, where each amino acid is represented by its position in the protein sequence. On the Y axis we have the energy values for our model. As you see, each amino acid has its own energy value.

See that in this energetic profile you have two curves: one that has very extreme values and another that is more flat. This is because each curve shows the same results but using a different sliding window. When we apply a sliding window to a line plot we make that each point in the plot has the average value of all the neighboring positions. For example, with a sliding window of 20 we are making that each point in the plot has the average value for the 10 preceding and the 10 following positions in the protein sequence. This smooths irregularities and makes easier the understanding of the plot. Typically, it is recommended to work with a window size that is approximately the 10% of the sequence length. Remember that as the window is bigger we miss details in the energetic profile.

Local model quality

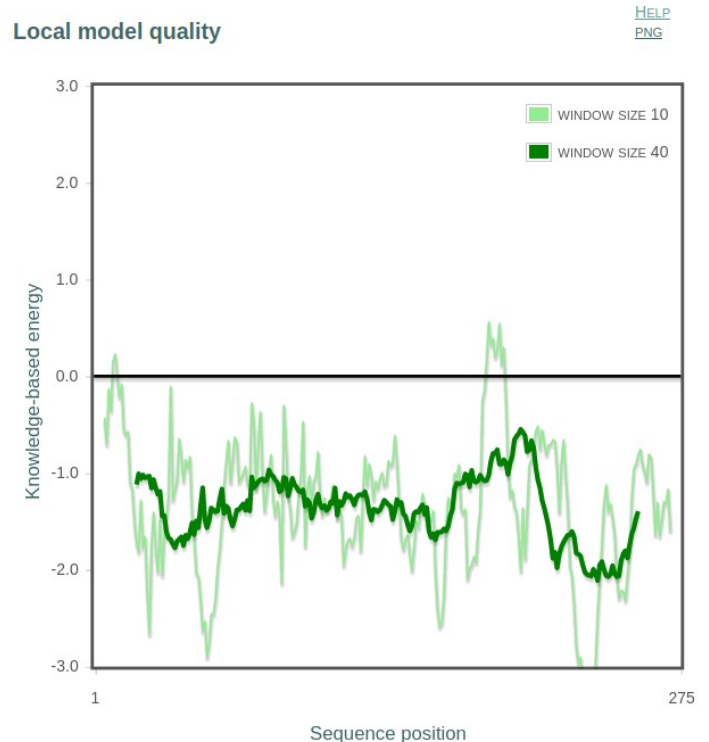
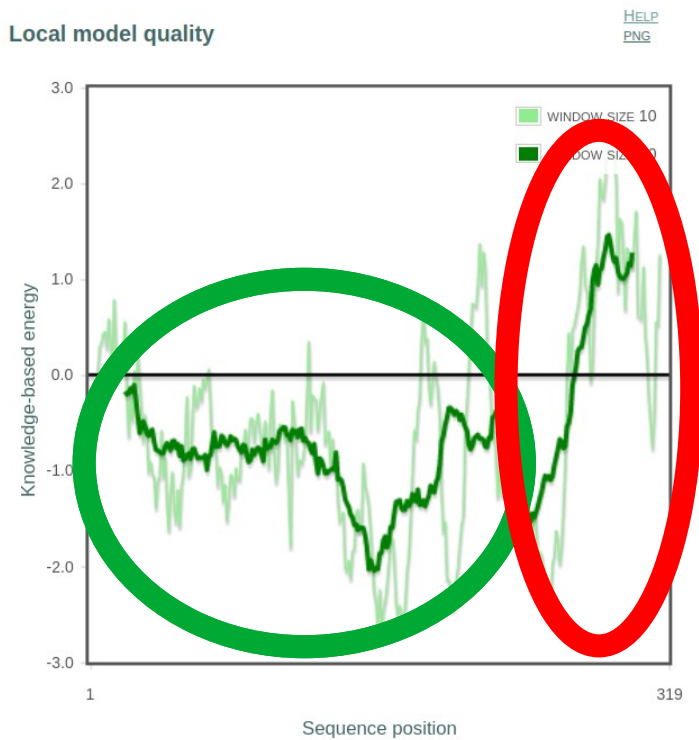
[HELP](#)
[PNG](#)



Prosa is providing three energetic values for the analysis of our protein model:

- PAIR energy: this energetic component is calculated taking into account the distances between pairs of amino acids. By default it uses distances between beta carbons. By doing so, prosa takes into account not just the location of amino acids, but also the orientation of the side chains.
- SURF energy: this energetic component is calculated taking into account the degree of exposure of amino acids. Prosa expects to find hydrophobic amino acids in the core of the protein and polar amino acids in the surface.
- COMB energy: it is a lineal combination of the PAIR and the SURF energies. Is the energy score that prosa web shows you by default.

By comparing the energy profiles we can identify that our model has some regions that are well modeled and some others that are not. We will see this by the energy scores. The lowest the energy, the most stable is the structure. For the model this is an indication that is has been well modeled. So, those regions that are showing clearly higher energies in the model in comparison with the template are likely to be wrongly modeled.



By looking at the plots (model in the left and template in the right) we identify two regions:

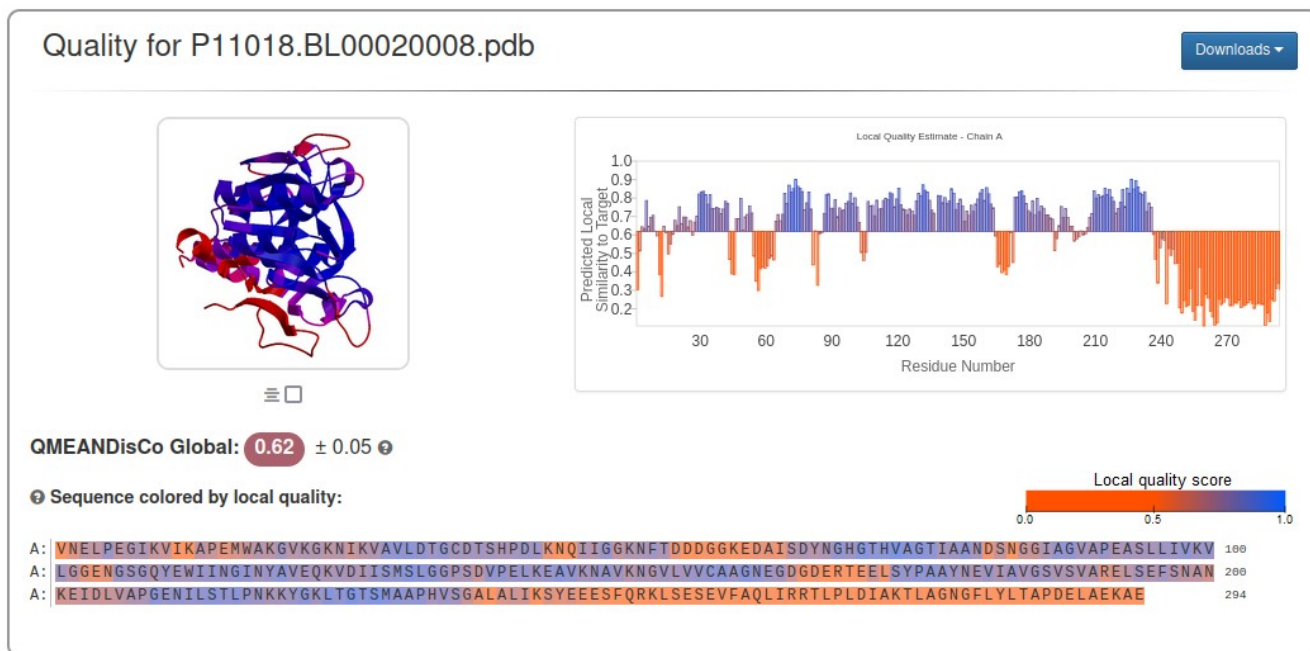
- A well modeled region (Green): here the energy values are similar between the model and the template.
- A wrongly modeled region (Red): here the energy values of the model are much higher than the ones of the template. These regions can be corrected, as we will see in the following practice.

We clearly see that one part of the model is alright and the other is wrong. We have two ways of dealing with this situation:

- Removing the wrongly modeled part of the model. By doing so we will have an incomplete model, but at least we will be sure that the modeled parts are right.
- Correct the model. There are many strategies to correct the model. We will see some of them in the next practical lesson.

Another website where you can use statistical potentials to analyze protein structures is QMEAN, from the University of Basel: <https://swissmodel.expasy.org/qmean/>

This page takes a little longer to run (10-15 minutes) but it displays the results in a more cool way. I strongly recommend you to use this page to analyze the effect of mutations in your project. On the other hand, for the practical classes and exams, since we want to be fast, prosa web is the best option. Here you can see an example of this display:



You can see that both methods, prosa and QMEAN, are identifying errors in the modeling of the C-terminal of our model.

Step 4: evaluating a set of threading models

So far, we have used prosa to compare a model with a template with the purpose of identifying wrongly modeled regions. We can also use prosa to compare different models between them. If we have produced a set of models and we want to know which model is the best, using prosa is a good strategy. Our hypothesis is that the model having the lowest energies is going to be the best modeled.

We are going to work with a set of models obtained by different threading programs. These models are inside the threading directory inside the “exercise_4” folder. You have this files inside the sitdoc cluster, so there is no need to copy them from the local computer. Then, copy the exercise_4 directory in your home directory and go to “exercise_4/THREADING/” directory.

Here we have threading models that have been obtained using different threading programs. These programs are I-tasser and phyre. You can access these programs through the internet and submit your queries. Remember how these threading programs work:

1. They contain a database of protein structures.
2. They fit the target sequence inside each one of these structures. For each fitting, energetic scores are obtained.
3. The structures showing the best scores are used to create a protein model.

Now, open a prosa web page for each of these models and compare their Z-scores and energetic profiles to know what is the best.

Questions from the tutorial:

- 1) Obtain models for other templates that you didn't use during the tutorial. Then compare the models. Do they look the same? Do they have the same scores profiles in PROSA? Why is this happening?
- 2) Use the target to identify 5 new templates. Then, use these templates to build a structure-based HMM as we saw in exercise 3. Chose one of these templates and align it with the target using the structure-based HMM.
- 3) Search in PFAM the HMM that fits the best your target protein. Then align the template that you choose in the previous exercise with your target sequence using the HMM you just retrieved from PFAM.
- 4) Use the structure-based alignment and the sequence-based alignments obtained in the exercises 2 and 3. Use these alignments to make models of the target. Then compare the models obtained with different alignments. Do they look the same? Do they have the same scores profiles in PROSA? Why is this happening?
- 5) Try to build models for the targets contained in the "extra_targets" directory.