INTRODUCTION

In Orellana **et al.** 2016, we have used all the crystal structures of a protein to create an ensemble. What we call an ensemble in this article is a set of structures that are not chimeric, represent the whole protein rather than a single domain of it, and do not contain more than 5 consecutive non-terminal missing residues. As much as we would like to automate this process, preparation of an ensemble often requires manual intervention due to problems like, chimeric constructs or the placement of subunits (clockwise vs. anticlockwise). There is also a decision to be made whether to exclude a structure that has non-terminal missing residues.

While preparing the ensembles for this article, we have taken care of those problems manually and made a decision to repair the structures if it was known to represent an important state which does not have any other representatives. Altough it very much depends on the state structure, and the location of the missing residues, we recommend repairing structures with only less than 5 consecutive residues missing. There is an example modeller script that can be used for this. It is upto the user to fix and include them in the ensemble later.

This tutorial will be using GLIC as an example. Because this was the most challenging ensemble we prepared, it highlights most of the issues the users are likely to encounter.

HOW TO

Requirements

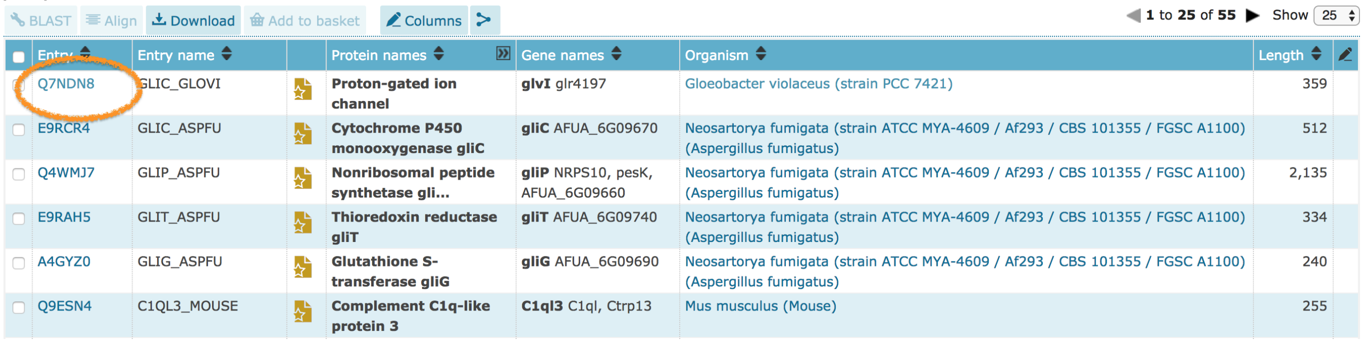
Before we get started make sure you have installed the pdbParser package, or you are running this notebook in a place where pdbParser is the subfolder. You will find the other required python packages within github (https://github.com/ozyo/pdbParser) README.md

To determine the missing residues, pdbParser relies on sequence alignment. If you use pdbParser to prepare a start and an end structure for eBDIMs then the alignment is generated via BioPython. However if you are preparing an ensemble you must install Clustal Omega or run the provided fasta sequence through a multiple sequence alignment tool and save the alignment in fasta format. The installation of Clustal Omega can easily be done via conda. After you install it you should provide the full path for the clustal executable.

Altough not a requirement, if you want to fix any broken structures you will need a modelling program. We have provided an example script for MODELLER.

Obtaining PDB Files

It can be daunting to find all the available PDB files, and the chains that actually belongs to your protein. The easiset way to do this is to go through the UniProtKB database and pdbParser requires the UniProt ID to prepare the ensemble.



Entry	Entry name	Protein names	Gene names	Organism	Length
Q7NDN8	GLIC_GLOVI	**Proton-gated ion channel**	**glvI** glr4197	Gloeobacter violaceus (strain PCC 7421)	359
E9RCR4	GLIC_ASPFU	**Cytochrome P450 monooxygenase gliC**	**gliC** AFUA_6G09670	Neosartorya fumigata (strain ATCC MYA-4609 / Af293 / CBS 101355 / FGSC A1100) (Aspergillus fumigatus)	512
Q4WMJ7	GLIP_ASPFU	**Nonribosomal peptide synthetase gli...**	**gliP** NRPS10, pesK, AFUA_6G09660	Neosartorya fumigata (strain ATCC MYA-4609 / Af293 / CBS 101355 / FGSC A1100) (Aspergillus fumigatus)	2,135
E9RAH5	GLIT_ASPFU	**Thioredoxin reductase gliT**	**gliT** AFUA_6G09740	Neosartorya fumigata (strain ATCC MYA-4609 / Af293 / CBS 101355 / FGSC A1100) (Aspergillus fumigatus)	334
A4GYZ0	GLIG_ASPFU	**Glutathione S-transferase gliG**	**gliG** AFUA_6G09690	Neosartorya fumigata (strain ATCC MYA-4609 / Af293 / CBS 101355 / FGSC A1100) (Aspergillus fumigatus)	240
Q9ESN4	C1QL3_MOUSE	**Complement C1q-like protein 3**	**C1ql3** C1ql, Ctrp13	Mus musculus (Mouse)	255

After finding the UniProt ID you can run the commands below to gather information. Besides the UniProt ID, you also have to provide the total number of chains that are in a biological assembly. Heteromeric structures require a different routine which are not yet supported in pdbParser. However similar logic applies in preperation of such an ensemble that only the shared CA coordinates should be kept, any broken structures should be fixed or excluded.

You should also provide a folder that exits for inputs/outputs to be read/written. If you use "." This will take the current directory for outputs.

In [1]:

```
query='Q7NDN8' #UniPortKB ID
mer=5 #Total number of chains
cwd="../../test/" #This directory must exist
#exclude=[List of PDB IDs]
#If you already know some structures you want to exclude provide them as a list.
```

In [2]:

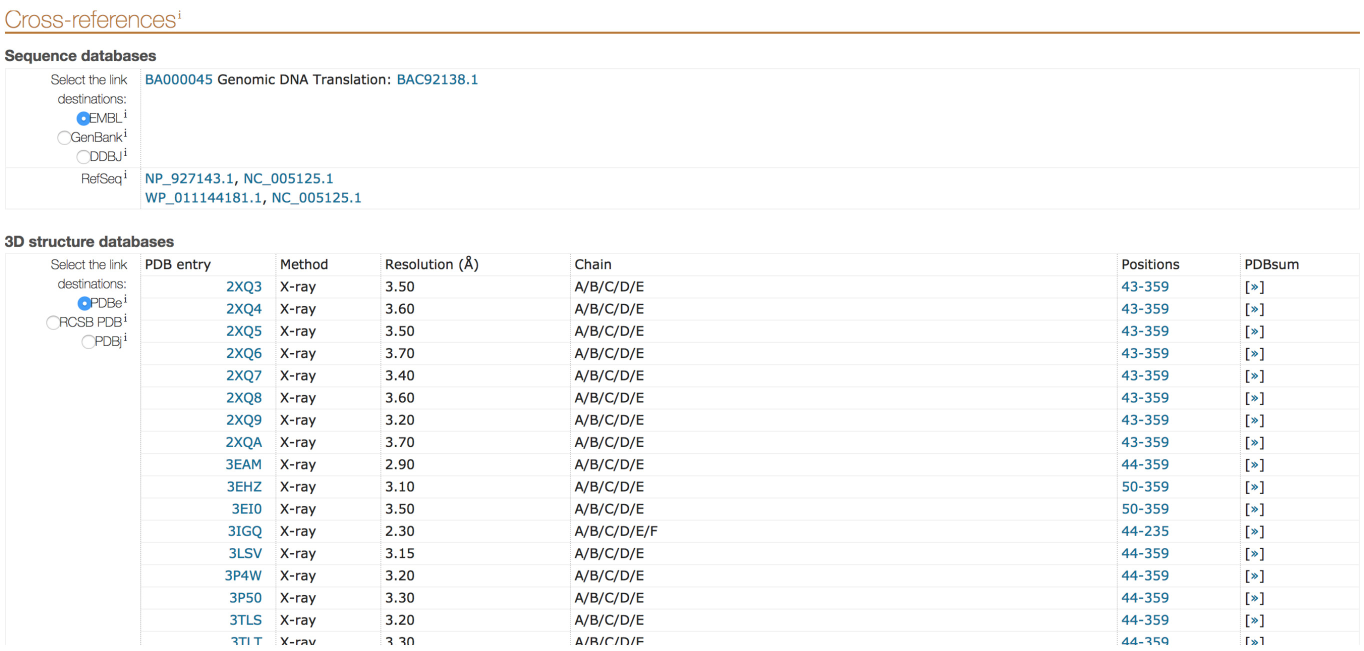
```
from pdbParser import prepENS as pE
reload(pE)
info=pE.PDBInfo(query,mer) #Add the exclude argument to the end.
```

```
CRITICAL:root:Cannot process PDB id 3IGQ. It does not contain comple
te set
```

PDBInfo function goes into the UniProt page of the given ID, and retrives the 3D structures table (as seen in figure below) and the reference sequence to be used for the alignment. For the reasons below it is much easier to prepare the ensemble via a database like UniProtKB:

- Because there is a possiblity of all the structures having the same missing residues, the reference sequence is a good control to have.
- Within 3D structure table, only the ones with the label "X-ray" are downloaded. Currently we don't support multi-model pdb files.
- The relevant chains for this protein is also extracted from this table, this is much easier than reading PDB headers.

In the example above, you already see that 3IGQ structure is skipped. That is because this file had 6 subunits vs. 5. If you go into the RCSB Database and look at this structure you will see that it contains only the extracellular domain and not the whole structure. To our advantage this one had more subunits than we required, so it is eliminated early on.



Cross-references

Sequence databases

Select the link destinations: ● EMBL ○ GenBank ○ DDBJ	BA000045 Genomic DNA Translation: BAC92138.1
RefSeq	NP_927143.1, NC_005125.1 WP_011144181.1, NC_005125.1

3D structure databases

Select the link destinations: ● PDBe ○ RCSB PDB ○ PDBj

PDB entry	Method	Resolution (Å)	Chain	Positions	PDBsum
2XQ3	X-ray	3.50	A/B/C/D/E	43-359	[»]
2XQ4	X-ray	3.60	A/B/C/D/E	43-359	[»]
2XQ5	X-ray	3.50	A/B/C/D/E	43-359	[»]
2XQ6	X-ray	3.70	A/B/C/D/E	43-359	[»]
2XQ7	X-ray	3.40	A/B/C/D/E	43-359	[»]
2XQ8	X-ray	3.60	A/B/C/D/E	43-359	[»]
2XQ9	X-ray	3.20	A/B/C/D/E	43-359	[»]
2XQA	X-ray	3.70	A/B/C/D/E	43-359	[»]
3EAM	X-ray	2.90	A/B/C/D/E	44-359	[»]
3EHZ	X-ray	3.10	A/B/C/D/E	50-359	[»]
3EI0	X-ray	3.50	A/B/C/D/E	50-359	[»]
3IGQ	X-ray	2.30	A/B/C/D/E/F	44-235	[»]
3LSV	X-ray	3.15	A/B/C/D/E	44-359	[»]
3P4W	X-ray	3.20	A/B/C/D/E	44-359	[»]
3P50	X-ray	3.30	A/B/C/D/E	44-359	[»]
3TLS	X-ray	3.20	A/B/C/D/E	44-359	[»]
3TLT	X-ray	3.30	A/B/C/D/E	44-359	[»]

After the information is gathered, info object is populated with the possible structures to be used and the reference sequence. You can reach the list via commands below. If you want to manipulate this list, add or remove structures, or change the reference sequence you can do so at this stage. Please not that all the functions in prepENS module uses and modifes this class directly.

In [12]:

```
info.query #UniPort ID used to initialize this class
#and it is also used for the output names.
info.mer # Total number of chains
info.result #Contains the information from the 3D structures table.
print info.refseq #Contains the reference sequence.
```

```
MFPTGWRPKLSESIAASRMLWQPMAAVAVVQIGLLWFSPPVWGQDMVSPPPPIADEPLTVNTGIYLIE
CYSLDDKAETFKVNAFLSLSWKDRRLAFDPVRSGVRVKTYEPEAIWIPEIRFVNVENARDADVVDISV
SPDGTVQYLERFSARVLSPLDFRRYPFDSQTLHIYLIVRSVDTRNIVLAVDLEKVGKNDDVFLTGWDI
ESFTAVVKPANFALEDRLESKLDYQLRISRQYFSYIPNIILPMLFILFISWTAFWSTSYEANVTLVVS
TLIAHIAFNILVETNLPKTPYMTYTGAIIFMIYLFYFVAVIEVTVQHYLKVESQPARAASITRASRIA
FPVVFLLANIILAFLFFGF
```

The next command downloads the structures from the RCSB database. Depending on the number of structures this function might take a while to finish because after it downloads the files it goes through the steps below to clean them:

1. It checks for lines with TITLE to find the chimeric structures.
2. It seperates the biological assemblies within the same PDB file
3. It keeps only the CA coordinates, which are the only relevant atoms for comparison with eBDIMs trajectories. Obtaining only CA coordinates is also a way to clean the PDB files from bound ligands, ions, waters, etc.
4. It extracts the sequence and the residue numbering from the CA coordinates and writes them to text files.

pdbParser relies on multiple sequence alignment to determine the missing residues. This also allows us to ignore the differences in residue numbering between different structures, non-consecutive numbering. The residue numbering information is only used to extract the residues from the the PDB file itself. The sequence for the alignment is extracted from CA coordinates, rather than PDB header. Because a residue might have missing CA coordinate, despite being listed in the sequence records.

In the example below you see that there are more warnings, stating that more structures will be excluded from the ensemble because they are chimeras. The function checks for two keywords in the title 'CHIMERA' and 'FUSED'. Since these structures cannot be used, their information is removed from the info.result dictionary automatically and those files are deleted.

In [4]:

```
pE.downloadPDB(info,cwd)
```

```
CRITICAL:root:PDB ID 4YEU is a chimera, skipping this file
CRITICAL:root:PDB ID 5OSA is a chimera, skipping this file
CRITICAL:root:PDB ID 5OSC is a chimera, skipping this file
CRITICAL:root:PDB ID 5OSB is a chimera, skipping this file
CRITICAL:root:PDB ID 4X5T is a chimera, skipping this file
```

Multiple Sequence Alignment and Identification of the Shared Region

If you check the output folder, you will find two text files: "_seq.txt" and "_residmap.txt". We need the information within these files for the multiple sequence alignment and the subsequent cleaning of the terminal residues that are not shared between the structures. It is possible that some structures still contain a terminal tag, that would interfere with the alignment and introduce unncessary gaps. Any chain that contains a non-terminal gap at this stage is marked as broken and the assembly they belong to is skipped.

If you chose to run the alignment somwhere else, you must clean any headers that is written to it. And the names in the fasta header must be in the same format as in the example file in this tutorial. The alignment should also contain a reference sequence with a header refseq. When you run the function below with the argument alnf="alignmentfile.fasta" , it will only go through the file to determine the broken chains. It does not run clustal, so you don't have to provide it. The alignment file must be placed where **cwd** is set to.

In [5]:

```
reload(pE)
clustalopath="/Users/cattibrie_fr/anaconda/anaconda/bin/clustalo"
pE.msa(info,cwd,clustalopath)
```

```
CRITICAL:root:4NPQ_2.pdb|H| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_2.pdb|I| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_3.pdb|K| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_3.pdb|L| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_3.pdb|M| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_4.pdb|S| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:3UU3_1.pdb|A| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:3UU3_1.pdb|B| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:3UU3_1.pdb|C| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:3UU3_1.pdb|D| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:3UU3_1.pdb|E| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
CRITICAL:root:4ILA_1.pdb|E| contains insertions or missing residues,
skipping this chain and the assembly it belongs to.
```

Noticed how I have given only the class instance as an argument? Since I have initialized the input/output filenames and created attributes within this class with the relevant data, I don't need to pass objects individually. So if you need to change names of the sequence file, for example, you can do so by changing the info.seqfilename . If you need to add or delete chains/structures you must modify both the text files and the info.result dictionary.

As you see above there are quite a few chains that have missing residues or insertions. All these identified broken chains are stored in info.broken object. This list is used in the subsequent function that extracts the shared CA region. But before we do that we should take a look at the alignment file.

Altough the assemblies with broken chains will be skipped, the CA coordinates and the FASTA alignment information of those are kept. For example from the fasta alignment file, we saw that chain S in 4NPQ PDB file, assembly 4 has two missing residues. But the rest of the structures in this assembly is intact. Because there are not many "possible closed state" structures of GLIC, we decided to repair this assembly. You can use the example MODELLER script to rebuild those missing residues. After you have fixed the structure you should follow the steps below:

1. Modify the info.broken object to exclude that chain.
2. You must also add it back to the info.coreresids and info.coremer.
3. You must rename the final, fixed file to it's original 4NPQ_4.pdb

In [6]:

```
%run -i 'fix_model.py' #You have to modify the contents of this file if you are
 working on your own example.
info.broken.remove('4NPQ_4.pdb')
#Python indexing starts from 0
chains=info.result['4NPQ'][1][3]
#Chain informations are the 2nd element and within that list the 4th ensemble.
info.coremer['4NPQ_4.pdb']=chains
info.coreresids['4NPQ_4.pdb|S|']=info.coreresids['4NPQ_4.pdb|T|']
#Since this is a homopentamer, I have just used the information from other chain
 to set the residue numbers for |S|
```

In [7]:

```
reload(pE)
pE.getcore(info,cwd)
```

Now that we have all the structures we want to use ready, and the list of broken structures is updated we can move on to the next step to extract the common residues. As you can see for GLIC some files does not have the initial valine and serine residues ın the N termini. Some of the structures also have a missing residue in the C termini. The function below, uses the information returned from the msa function above. It finds which residues should be removed from the N and C termini by mapping back the sequence to the residue numbers written to "_residmap.txt". The next and the last step is to align the structures.

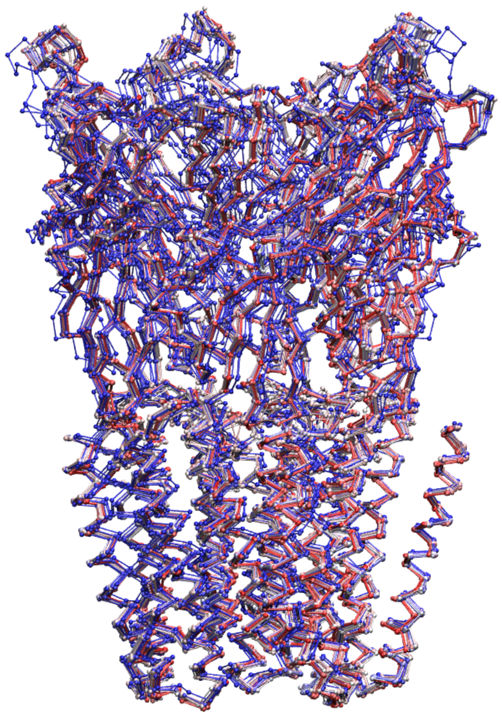
Structure Alignment

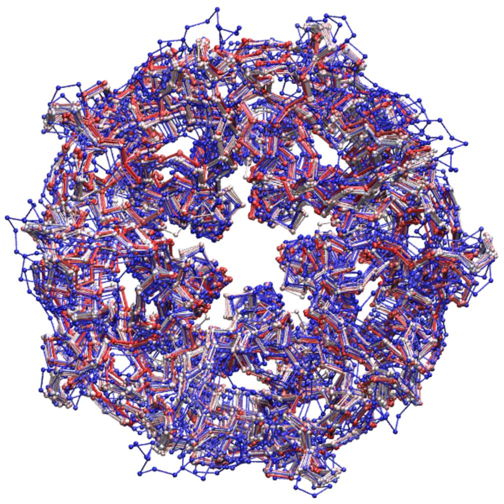
Since we have extracted a common region, and CA coordinates only, all the structures have the same number of atoms. Regardless of the point mutations and the absolute residue number differences we can use almost any alignment tool to superimpose them. For this step you can use your favorite visualizer. In fact it is a good idea to look at the ensemble, and check each structure individually. For example some GLIC structures have a different subunit placement: anticlockwise vs. clockwise. This makes it impossible to superimpose. To fix this issue you could reorder and rename the chains so that the placement matches the rest of the ensemble.

When you are finished the alignment of the structures, all you need to do is to combine them, then your ensemble is ready for uploading to the server.Since eBDIMs requires MODEL and END lines in the beginning and the end of the file, you can use a bash loop similar to this to combine them. Since all the structures have the same number of atoms, you can do the alignment with this combined file rather then individual files.

In [11]:

```
%%bash
touch ../../test/GLIC_ensemble.pdb
for pdb in `ls -1 ../../test/correct*pdb`
do
echo "MODEL" >> ../../test/GLIC_ensemble.pdb
cat $pdb >> ../../test/GLIC_ensemble.pdb
echo "END" >> ../../test/GLIC_ensemble.pdb
done
```





If you have any questions do not hesitate to contact us. pdbParser, ensemble preperation is a package being developed. Watch out for new features on [github (https://github.com/ozyo/pdbParser)](https://github.com/ozyo/pdbParser)

```
In [ ]:
```