Single_UniProt_Code_Python3

September 26, 2019

0.1 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 nonterminal 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 the decision to be made whether to exclude a structure that has missing residues that are not terminal.

While preparing the ensembles for the 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 structure, whether there are other structures in a similar state to be used as template or where the missing residues are located, we recommend repairing structures with less than 6 consecutive residues missing. There is an example modeller script that can be used to repair structures. 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 most challenging ensemble we prepared for this study, it highlights the potential issues the users likely to encounter.

0.2 HOW TO

0.2.1 Requirements

Before we get started make sure you have installed the pdbParser package. You will find the other required python packages within github README.md

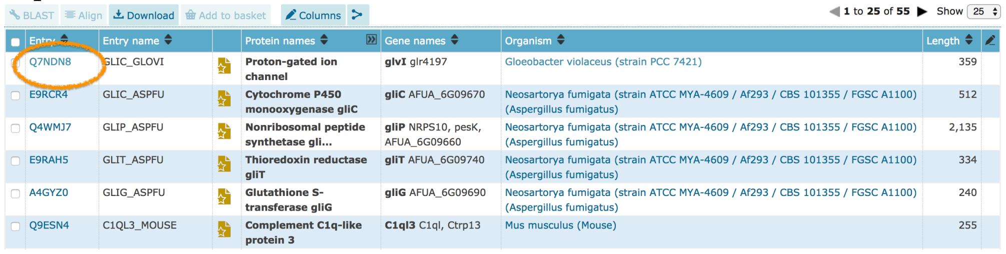
To determine the missing residues, pdbParser relies on sequence alignment. If you use pdb-Parser to prepare a start and an end structure for eBDIMs then the alignment is generated via BioPython. However if you are preparing an ensemble you must install Clustal Omega or run the extracted sequences through 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 clustalo executable if it is not already in your path.

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.

0.2.2 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 save the IDs.

pdbParser only requires the UniProt ID and how many chains are there in a biological assembly 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 IDs you can run the commands below to gather information on the reference sequence and the available PDBs. This routine is for homomeric ensembles, where a single reference sequence is sufficient. Heteromeric structures require a different routine that is covered in the Multi UniProt Code Tutorial. The logic is similar in both that only the shared CA coordinates should be kept, any broken structures should be fixed or excluded.

There are quite a lot of files to read and write, a work directory must be created before run time. If you use "." this will use current directory for read/write. Since you could be running this notebook from any location, it is a better practice to provide a full path.

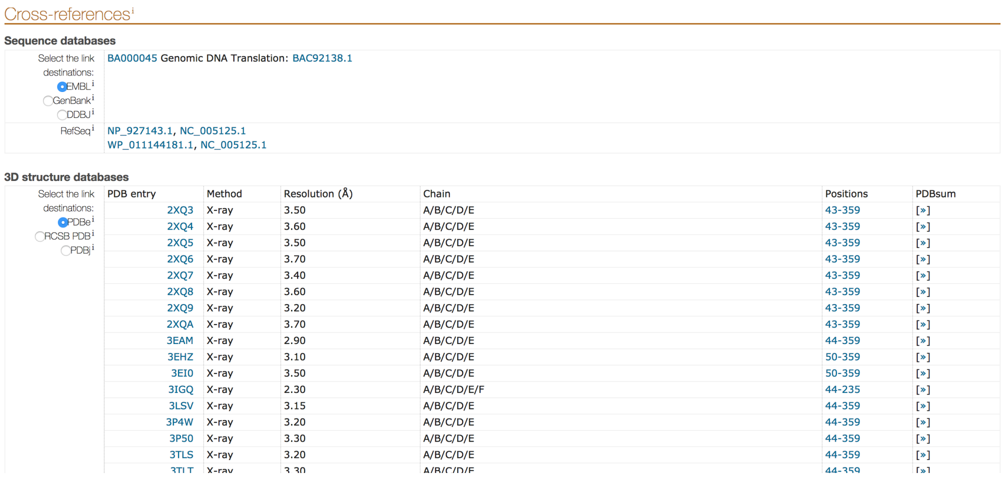
[1]:
```
query='Q7NDN8' #UniPortKB ID
mer=5 #Total number of chains
cwd="./" #This directory must exist
#exclude=[List of PDB IDs]
#If you already know some structures you want to exclude provide them as a list.
```

[2]:
```
from pdbParser import prepENS as pE
from importlib import reload
reload(pE)
info=pE.PDBInfo(query,mer)
#to exclude pdbs already from this list: pE.PDBInfo(query,mer,exclude=[ID1,ID2])
```

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

PDBInfo class, during initiation, 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 (no NMR, sorry!). * 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 instead of 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 have set, so it is eliminated early on. However there could be other incomplete structures like this. Visualising the ensemble, inspecting the alignment manually is always a good idea!


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 from the attributes assigned to this calss. If you want to manipulate this list, add or remove structures, or change the reference sequence you can do so at this stage. Please note that all the functions in prepENS module uses and modifes this class directly. So the attiribute names are important and they should remain unaltered. The values of the attributes , on the other hand, can be manipulated to improve the ensemble.

[3]:
```
print(info.__dict__.keys()) #If you want to see the attributes of the PDBInfo␣
 ↪class ;)
print(info.query) #UniPort ID used to initialize this class
#and it is also used for the output names.
print(info.refseq) #Contains the reference sequence.
```

```
dict_keys(['exclude', 'query', 'mer', 'result', 'refseq', 'broken',
'seqfilename', 'residmapfilename', 'alnfasta', 'coremer', 'coreresids'])
Q7NDN8
MFPTGWRPKLSESIAASRMLWQPMAAVAVVQIGLLWFSPPVWGQDMVSPPPPIADEPLTVNTGIYLIECYSLDDKAETFK
VNAFLSLSWKDRRLAFDPVRSGVRVKTYEPEAIWIPEIRFVNVENARDADVVDISVSPDGTVQYLERFSARVLSPLDFRR
YPFDSQTLHIYLIVRSVDTRNIVLAVDLEKVGKNDDVFLTGWDIESFTAVVKPANFALEDRLESKLDYQLRISRQYFSYI
PNIILPMLFILFISWTAFWSTSYEANVTLVVSTLIAHIAFNILVETNLPKTPYMTYTGAIIFMIYLFYFVAVIEVTVQHY
LKVESQPARAASITRASRIAFPVVFLLANIILAFLFFGF
```

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 for cleaning:

1. It checks for lines with TITLE to find and exclude 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, and non-consecutive numbering (an indication of missing domains, insertions/deletions). The sequence of the structures are extracted from CA coordinates, rather than PDB header. Because a residue might have missing CA coordinate, despite being listed in the sequence records.

The residue numbering information is only used to extract the residues from the the PDB file itself and it is not used in any other way in the alignment.

In the example below you will see that there are more warnings, stating that more structures will be excluded from the ensemble because they are chimeras. Since these structures cannot be used, their information is removed from the info.result dictionary automatically and those files are deleted.

If you check the output folder at this stage, you will find two text files: "_seq.txt" and "_residmap.txt“.”_seq.txt" contains the sequences and "_residmap.txt" holds the corresponding residue numbering. These will be used below in the sequence alignment.

[4]:
```
pE.downloadPDB(info,cwd)
```

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

0.2.3 Multiple Sequence Alignment and Identification of the Shared Region

Multiple sequence alignment of the sequences together with the reference sequence is used for cleaning of the terminal residues that are not shared between the structures (most likely terminal tags, or undetermined regions). Since we use a reference sequence we can also identify structures that contain a non-terminal gap, mark them as broken and remove the assembly from the ensemble. If you have Clustal Omega installed you can proceed with the alignment generation which in the same function parses the alignment as well.

If you chose to run the alignment somwhere else, you have to do is to clean any headers that it might contain and save it in the *fasta* format. The names of the fasta identifiers must be exactly in this format: PDBID_ENSEMBLENR.pdb|CHAINID| , as given in the example file in this tutorial. The alignment should also contain a reference sequence with an identifier starting with “refseq_”. The sequences extracted by pdbParser already have these identifiers, so that you don’t have to make any modifications. When you have your alignment file you can, then, run the function below with the argument alnf=“alignmentfile.fasta”. The alignment file must be placed where *cwd* is set to. The function will only go through the file to determine the broken chains. It does not run clustalo, so you don’t have to provide the path for it or have it installed.

[5]:
```
clustalopath="/Users/cattibrie_fr/miniconda2/bin/clustalo"
pE.msa(info,cwd,clustalopath)
```

```
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.
```

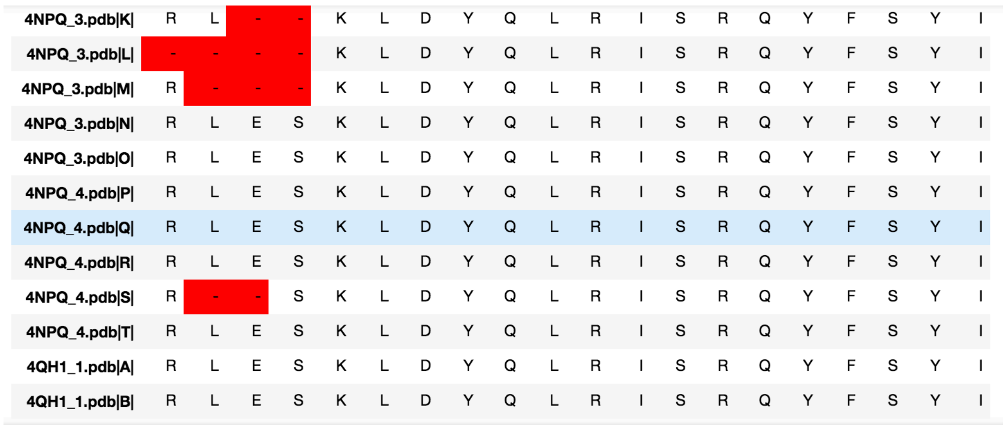
```
CRITICAL:root:3UU3_1.pdb|E| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:4ILA_1.pdb|E| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:4NPQ_2.pdb|H| contains insertions or missing 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:6HYR_1.pdb|A| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:6HYR_1.pdb|B| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:6HYR_1.pdb|C| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:6HYR_1.pdb|D| contains insertions or missing residues, skipping
this chain and the assembly it belongs to.
CRITICAL:root:6HYR_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 atrribute. This list is used in the subsequent function that extracts the shared CA region. Altough the assemblies with broken chains will be skipped, the CA coordinates and the FASTA alignment information of those are kept.

Before we proceeed we can take a look at the alignment file with info.core_show. You can specficy which region of the alignment file you want to display using positions. The positions correspond to the total alignment itself, so residue number 159 is not in position 159. Alignment file will help you identify which chains you would like to repair and keep.

[ ]:
```
info.core_show(positions=[220,240]) #Displays a part of the alignment
#It also adds alndata attribute to info, so that you can play around with this␣
↪data frame object.
```



4NPQ_3.pdb|K|	R	L	-	-	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_3.pdb|L|	-	-	-	-	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_3.pdb|M|	R	-	-	-	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_3.pdb|N|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_3.pdb|O|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_4.pdb|P|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_4.pdb|Q|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_4.pdb|R|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_4.pdb|S|	R	-	-	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4NPQ_4.pdb|T|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4QH1_1.pdb|A|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I
4QH1_1.pdb|B|	R	L	E	S	K	L	D	Y	Q	L	R	I	S	R	Q	Y	F	S	Y	I

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

Now that we have all the structures we want to use fixed and ready, 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 n the N termini. Some of the structures also have a missing residue in the C termini. This function will remove all of those based on the alignment file shown above.

[16]:
```
#%run -i 'fix_model.py' #You have to modify the contents of this file if you are working on your own example.
#There is a possiblity of crashing the kernel with this fix_model.py script. Hence it is commented out.
# It is for MODELLER and can be run outside.
print(info.__dict__.keys())
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|
```

```
dict_keys(['exclude', 'query', 'mer', 'result', 'refseq', 'broken',
'seqfilename', 'residmapfilename', 'alnfasta', 'coremer', 'coreresids',
'alndata'])
```

[17]:
```
pE.getcore(info,cwd)
```

0.2.4 Structure Alignment

Since we have extracted a common region, and kept CA coordinates only, all the structures have the same number of atoms. Regardless of the point mutations and the differences in residue numbering 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. There could still be unforseen problems.

For example some GLIC structures have a different subunit placement: anticlockwise vs. clockwise. This makes it difficult 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 aligning 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. In fact, to do the structural alignment you can use the ensemble file directly.

[19]:
```
%%bash
touch ./GLIC_ensemble.pdb
for pdb in `ls -1 ./correct*pdb`
do
echo "MODEL" >> ./GLIC_ensemble.pdb
cat $pdb >> ./GLIC_ensemble.pdb
echo "END" >> ./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