README_SBI_PYT_Project

Júlia Mir, Helena Rodriguez

April 3rd, 2018

Modelling protein complexes

Aim of the project

The aim of this project is to create a program in python that is able to reconstruct a whole protein or complex from PDB fragments of pairs of interacting chains. In this case, first aligning the sequences to find good templates for the model and then superimposing them.
We will work with the PDB data a researcher gave us with interactions between protein chains and try to assemble the whole protein or complex.

Dependencies

- Biopython 1.70
- PDB database
- Position-Specific Initiated BLAST 2.2.31+
- CLUSTAL 2.1 Multiple Sequence Alignments

Usage

Input: Pairs of chains interacting (pdb format)
Considerations:
- The same chain has always the same name.
- Two different chains won't have the same name.
- Input names are different from template names.
- The program can deal with different protein structures and from different sizes (many number of chains).
- Input files can be named to convenience, despite this, we recommend to name the files as chain names (i.e. chainA + chainB → AB.pdb).
- Not all interactions existing in the structure must be in the input, but yes all chains. I.e., for a structure with interactions AB, AC, BC, possible inputs are: *AB.pdb and AC.pdb*, *AB.pdb and BC.pdb*, *AC.pdb and BC.pdb* and *AB.pdb, BC.pdb and AC.pdb*.

In this package we provide a script that models a protein structure using as input pairs of chains that interact. The used approach is superimposition to protein templates. The main script follows the pipeline (section 4) in order to solve the problem. The code is divided in modules (described in section 3). All modules are added in a folder called Modules, all functions can be reused for other programes if the modules are imported. All the code is writen in Python3.
To run the program from the terminal you can use a command of the following format:
python3 ModelStructure.py -i AB.pdb BC.pdb -d ./databases/pdbseqres.txt -v True
Seting verbose as True (-v True) is optional.
Be sure to run the program in the same folder where you have the input files (in the run folder you mus have: the ModelStructure.py script, the Modules folder and all input PDB files).

Note that many files will be generated as outputs from different programs (BLAST, ClustalW2, etc) and the final model PDB files will be provided to you as an output in the terminal.

Modules and functions descriptions

ModelStructure.py: Runs the programe following the pipeline and using handmade modules.
FileParsersGenerators.py: Contains functions to parse PDB files, split a PDB file by chains and create a PDB and a FASTA file for each chain and join several FASTA files: - *ParsePDB*: Parses PDB files using biopython by creating PDBParser objects, taking as input a list of PDB files. This function is used many times throughout the program. The output is a list of PDB objects.
- *SplitChain*: Splits a list of PDB files by chain creating one PDB and one FASTA file per chain. Its input is the list of PDB objects created in the previous function, and the output consists on a list with the prefixes of the files, a pdb for each chain and a fasta per each chain.
- *CreateJoinedFastas*: Joins many PDB objects and creates a FASTA file with all objects joined. The input is list of PDB objects whose sequence will be added to the FASTA file; and the output are FASTA files with all sequences from the same protein.
RunningAnalyzingPrograms.py: Contains functions to run BLAST and ClustalW2 and a function to analyze ClustalW2 outputs.
- *RunBlast*: Runs PSI-BLAST from python. The inputs are both the path of the database we want to align our query with, and the fasta of each chain from our query protein. The output is an XML for each fasta.
- *RunCustal*: Performs a multiple alignment running ClustalW from python, once per each FASTA file created in the previous function. The result is written in a .txt file.
- *AnalizeClustalScore*: Analyzes ClustalW output score files, and selects the chains that align with an specific score or higher. It requires the desired thershold of the score, the file with the scores and the name of the templates as inputs. Returns a list with all templates with a good alignment.
ProteinWorkingFunctions.py: Contains functions to select the best templates (from BLAST outputs), download templates from the PDB database, find interactions or clashes between chains of a protein structure, establish pairs of chains following some conditions and superimpose chains.
- *SelectTemplate*: Selects the best templates from the BLAST output (.xml) for all chains, choosing those with a minimum e-value. Returns a set with the name of all templates with the lowest e-value.
- *DownloadTemplate*: Downloads the desired template from the pdb database. The input is the output from the previous function, and the output is a PDB with for each template in .ent format.
- *FindInteractions*: Find interactions or clashes between chains. Takes the PDBs with the possible new proteins alreadey built and looks for clashes and interaction, using 5.0 A as the distance between interactions and 0.4 A as the distance between clashes.
- *AssignQueryToTemp*: Performs backtracking to assign each chain of the template to a chain of the target, taking into account the similarities between template-target chains and, as a condition, that if two chains from the target interact, the two assigned chains from the template must also interact. Its inputs are cand_list (the list contains tupples where the first element is one chain and the second element is a list of candidates for this chain), temp_chains (dictionary of each chain of each template as keys and None as values at the beginning corresponding target chains will be saved in this dictionary), Final_interactions: dictionary containing the equivalencies between target-template chains, the target interactions and the template interactions and target (the name of the actual template).
- *I_AssignQueryToTemp*: Performs an inmersion and performs the first recursive call. Assigns a list of candidate template chains to each target chain. It's inputs are targ_chain_list (the list contains the names of target chains), temp_chains, Final_interactions and temp.
- *Superimpose_chains*: Superimposes each target chain atoms to the corresponding template chain atoms. Their inputs are the PDB object of the current template, a list of target chains PDB objects and a dictionary with template-target chain pairs.
GeneralFunctions.py: Contains simple functions to clarify the code.
- *GetNameWOChain*: Gets the name without the chain of a protein ID (name will be of the format "abc"). The input is the name of the protein in format "abc_A".
- *GetTargInteractionKeys*: Returns the Target Interactions keys from the Final_interactions dictionary.

- *GetTempInteractionKeys*: Returns the Template Interactions keys from the Final_interactions dictionary.
- *GetChain*: Gets the chain name of a protein ID (chain name will be of the format "A"). The input is the name of the protein of format "abc_A".
- *GetTempInteractions*: Returns the Template Interactions values from the Final_interactions dictionary.
- *GetTargetInteractions*: Returns the Target Interactions values from the Final_interactions dictionary.

Pipeline

1. From the inputs, save each chain in a new PDB file and create a FASTA file for each chain.
2. Create PDB objects for each input file and for each chain.
3. Construct a dictionary containing the interactions in the input.
4. Run PSI-BLAST in order to find templates.
5. Download the templates from the PDB database.
6. For eacht emplate, create a FASTA file for each chain and add all input chains to this files.
7. For each FASTA file created in step 6, run ClustalW2.
8. Analyze ClustalW2 scores, finding similar chains.
9. Add template interactions in the dictionary.
10. Asign each template chain with possible target chains.
11. Add chain correspondencies (template-input) in the dictionary.
12. Superimpose the input chains to the target chains.
13. Look for clashes in the models in order to discard the ones that contain clashes.
14. Final models are returned by showing the name of the generated files in PDB format.

Example

For an example reconstructing Hemoglobin, please go to the example folder and read the report.