Reconstruction of a macro-complex using interacting subunits

Lydia Fortea Juan Luis Melero

Contents

Chapter 1

Background

The aim of the project is to reconstruct a marco-complex having only the pair interacting chains, using a standalone program created by ourselves. The program we created is based on several bioinformatic features, including structural superimposition and sequence alignment, among others.

1.1 Protein-Protein Interaction and Complexes

An important point of the project is understanding the Protein-Protein Interaction and Complexes. In the quaternary structure of a protein, there are more than one separated chains of proteins that interacti between them. The interaction of these chains can involve a lot of intermolecular bounds, such as hydrogen bounds, electrostatic interactions, pi stacking, cation-pi interaction, etc. This diversity of interactions makes the protein-protein interaction very common in order to stabilized the molecule and generate a biological function.

The whole structure, where two or more chains are combined and have one or different functions, is called a complex. The formation of a complex can be made by protein-protein interaction only or nucleotides (DNA or RNA) can also be part of a complex if there is DNA-protein or RNA-protein interactions.

Focusing on the project, having the protein-protein interaction by pairs, we want to reconstruct the whole macro-complex.

1.2 Sequence Alignment

Pairwise sequence alignment is done for several reasons. One of them is because we want to compare the chains from the same interacting file to know if it is a homodimer or a heterodimer. Another reason is to compare different interacting files, in order to know if they are the same pair of interacting chains or they are different. The last one, is to know which chains must be superimposed, in case the interacting chains are different heterodimers (see section Modules and Packages, subsection Homodimers and Heterodimers).

1.3 Structural superimposition

We cannot assume that the protein-protein interacting pairs are well oriented in the space. Therefore, in order to give to each part the correct orientation, we do a structural superimposition. Structural superimposition allows us to put the chains in the correct spatial orientation, and we use sequence alignment to know which chains must be superimposed. With superimposition, all the chains will be well positioned.

Chapter 2

Algorithm and Program

2.1 Inputs and Outputs

2.1.1 Input files

The program takes interacting pair chains for building the macro-complex. However, in the command line, the input must be one of the following:

- The list of PDB file names which contains the interacting pair chains.
- A directory that contains all the PDB files of interacting pair chains.
- By default, if no input is indicated, it takes the current directory as input, and takes all the PDB files as interacting pair chains.

Input files must be all in the directory specified in the arguments or in the current directory by default. The program will read all the PDB files, so it is necessary that all the PDB files in the working directory are the subunits of the macrocomplex and nothing else.

2.1.2 Output file

The output file will be one PDB file in which there will be the coordinates of the atoms of the macro-complex. The output file will be located in the directory specified in the arguments. Depending on the inputs (homodimer o heterodimer), the output will have different names in the current directory (homodimer.pdb or heterodimer.pdb). If the current directory is the same in which there are the input files, be careful if you want to exercute the program twice or more times at the same time, since it will take the output file also as one of the input files.

2.2 Modules and Packages

Biopython is the principal package used, as well as *sys*, *os*, *re*, *argparse*, *subprocess* and the self-created modules *homodimers* and *heterodimers*.

2.2.1 Biopython

Biopython is the main open-source collection of tools written in Python to work with biological data. From Biopython we take the following subpackages:

- Bio.PDB, to work with PDB files. From Bio.PDB we use the following subpackages:
  - PDBParser, to parse PDB files and obtain Structure Objects to work.
  - CaPPBuilder, to create the sequences taking into account Ca-Ca distance.
  - PDBIO, to save the structures into a PDB file.
  - Superimpose, to execute structural superimposition between structures.
- Bio.pairwise2, to align protein sequences one by one

2.2.2 sys

Sys package is the System-specific parameters and functions. This package is used to read the arguments in the command line (sys.argv) and to have access to the three channels of communication with the computer: the *standard in* (sys.stdin), the *standard out* (sys.stdout) and the *standard error* (sys.stderr).

2.2.3 os

Os package is the Miscellaneous operating system interfaces. With this package, the program can access synonimous commands of the shell, allowing the program to, for example, change working directory (*cd* in shell, *os.chdir* in python). This package is usefull to call command lines from the system but without consuming as much CPU.

2.2.4 re

Re package in the Regular expression operations package. It allows to work with regular expressions with python. In the program, it is used to find PDB and FASTA files, searching the extension of FASTA (.fa, .fasta) and PDB (.pdb) as regular expression at the end of the files.

2.2.5 argparse

Argparse package is the Parser for command-line options, arguments annd sub-commands. This packages allows to include the options for the user. The options included in the program are described in section *Options and Arguments*.

2.2.6 subprocess

Subprocess package is the subprocess manager that allows us to use bash and shell commands. In our program is used to call Chimera Software if the option -vz, --visualize (see section Options and Arguments) is active.

2.2.7 Homodimers and Heterodimers

Homodimers and Heterodimers are self-created modules which analyze the structures depending on the input files. If the pair of chains in the interacting files are the same (A-A), then it is considered homodimer. If the pair of chains in the interacting files are different (A-B) but all the files contain this heterodimer (files are A-B, A-B...), then it is considered *repeated heterodimer*. Finally, if the pair of chains in the interacting files are different and all interacting files are different (A-B, B-C...), then it is considered *heterodimer*. For homodimers and repeated heterodimers, homodimers module is used. For heterodimers, heterodimers module is used.

Homodimers module

Heterodimers module

Heterodimers module first extract all the sequences of the chains and align them with pairwise alignment and stores the percentage of identity (%id). Those chains with a %id greater than 99% are assumed to be the same and then, they are going to be superimposed. Once we know which chains must be superimposed, we do so taking one as reference chain and the other as

moving chain. Then the coordinates are updated and the reference chain is deleted in order not to have repeated superimposed chains.

2.3 Functions

In the program there are a lot of functions that will be used during the process of building the macro-complex.

2.3.1 get_input

This function handles the input argument. It uses regular expressions to find those files ended with ".pdb" and puts them into a list. If the input is a list of files, then it puts them directly into a list. The function returns a list that is the PDB interacting files that will be used to build the complex.

2.3.2 get_name_structure

This function uses regular expressions to return the filename of the PDB file.

2.3.3 get_sequence

This function deletes heteroatoms and returns the sequence of the proteins with more than 40 aminoacids (to avoid ligands). It uses CaPPBuilder package.

2.3.4 seq_comparison

This function is used to find identical sequences. It performs a pairwise sequence alignment and if the percentage of identity (%id) is greater than 0.95 it returns True, else, it returns False. It uses Bio.pairwise2 package to make the alignments.

2.3.5 chains_comparison

This function compares the chains of the interacting pairs and returns Frue if they are the same or False if they are different. It uses seq_comparison function.

2.3.6 get_pdb_info

This function analyze all the PDB files from the input. It extracts and compare the structures and their sequences in order to know if we have heterodimers o homodimers. It returns the pairwise interactions and True if there are homodimer interactiond and False if there are heterodimers.

2.3.7 temp_structure

This function creates a temporary structure with the interactions. It returns the built structure.

2.3.8 save_complex

This function saves the structure into a PDB file. It return nothing, but creates the PDB output file. This function uses Bio.PDBIO package.

2.3.9 heterodimer.align_sequences_heterodimers

This function aligns the sequences of the chains from different PDB files. Moreover, this function decides which are the more identical sequences to superimpose afterwards.

2.3.10 heterodimer.superimpose_structures_heterodimers

This function is used to superimpose different structures. The chains that will be superimposed are those with the best alignment from *heterodimer.align_sequences_heterodimers*. After the superimposition, one of the chains superimposed is removed not to have repeated chains in the structure. It returns the structures superimposed.

2.4 Workflow

First of all, we take the input files provided by the user using *get_input* function. We extract the information from these PDB files to know if we are dealing with homodimers (A-A interactions), repeated heterodimers (A-B, A-B...) or distinct heterodimers (A-B, B-C...). To do that, we use *get_pdb_info* function.

If the interactions are homodimers (A-A) or repeated heterodimers (A-B,

A-B...), then we use the standalone module *homodimers.py*. If the interactions are distinct heterodimers (A-B, B-C...), then we use the module *heterodimers.py*. You can see the workflow of each module in section *Modules and Packages*, subsection *Homodimers and Heterodimers*.

Finally, the program creates and saves a PDB file with the final structure using *temp_structure* and *save_complex* functions. If the option visualize (-vz, --visualize) is activated, it opens the PDB output file with chimera (see section *Options and Arguments* for requirements of this option).

2.5 Restrictions and Limitations of the Program

For heterodimers, one of the main restrictions is that all chains must be able to be followed. That is, we must be able to build a path joining all the subunits (A-B, B-C, C-D...). If it is not like that (A-B, C-D... for example), the program will crash. This is because one chain will be used as reference chain, and the other as moving chain. If it only appears once, then it is not possible to build the structure.

Chapter 3

How to use the program

3.1 Requirements

The main program requires for *Biopython* package and auxiliary modules *homodimers.py* and *heterodimers.py*.

3.2 Options and Arguments

In the program the following arguments are available:

- -i, --input; it takes the directory as input. By default, it takes the current directory.
- -o, --ouput; it takes the directory and the filename of the PDB file. By default, it creates a file called *output.pdb* in the current directory.
- -s, --sequence; it takes the directory and the filename of the FASTA file in which there is the sequence of the macro-complex. By default, it takes de value *None* and runs the program with default parameters.
- -v, --verbose; if this option is active, it prints the log to the standard error.
- -vz, --visualize; if this option is active, at the end of the script it opens the output file with Chimera. It will only work if Chimera is installed and the software is in `/usr/bin/chimera`, which is the path that the program calls Chimera Software.

All options are not forced to be, but it is highly recommended to use them, specially those related to the input/ouput files.

3.3 Logging

Activating the option for verbose (-v, --verbose), it prints to the standard error the log of the program. The point that are verbosed and the message if it is all right are the followings:

-

3.4 Running the program

To run the program in the terminal:
`$ python3 main.py -i [input files] -o [output file]`

You can activate as many options as you would like:
`$ python3 main.py -i [input files] -o [output file] -v`

Chapter 4

Analysis of examples

4.1 Tested examples

We tested two examples. One which is an heterotrimer (heterotrimeric G protein) and one repeated dimer (the example provided by Python teacher).

4.1.1 Heterotrimer

The example used is an heterotrimeric G protein, whose PDB id is 3AH8. We split it into two subunits (chain A - chain B and chain B - chain G). These are the subunit for the heterotrimer:

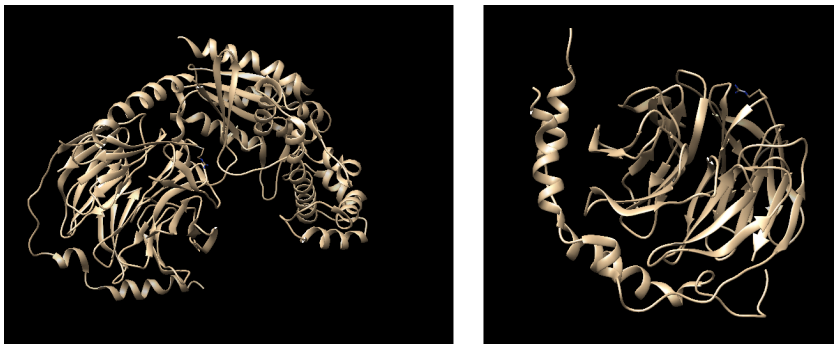

Figure 4.1: Subunits by pairs on a heterotrimeric G protein (3AH8). On the left, interaction of chains A-B. On the right, interaction of chains B-G.

After running the program, the heterotrimeric G protein looks like que original PDB file. RMSD is 0, because is a toy model and fits perfectly.

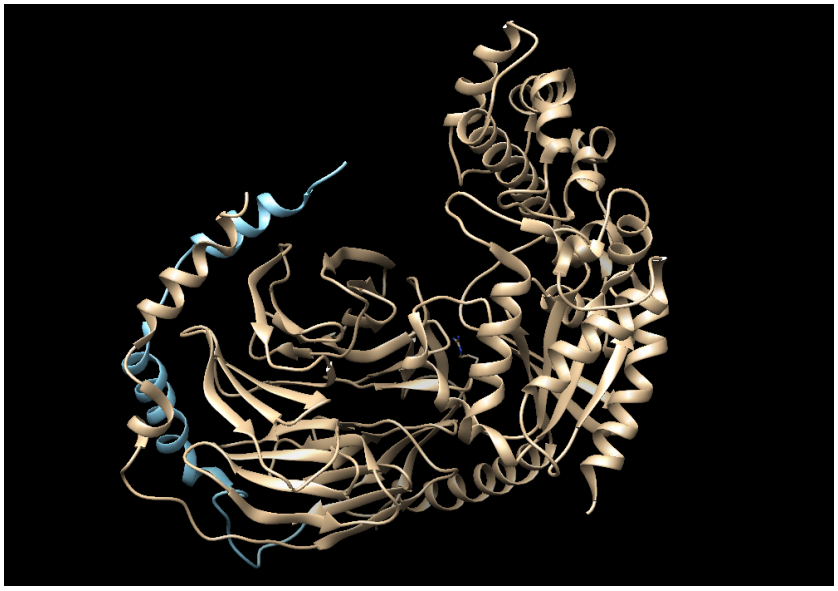

Figure 4.2: Complete structure of heterotrimeric G protein (3AH8), after running the program.

4.1.2 Homodimer

The example used for homodimers is the one provided by Professor Javier Garcia. This structure represents a nucleosome (and we were given the PDB files for interacting pairs of chains and those chains separatedly. We only use those interacting PDB files and NOT separated chains. There were 23 elements, homodimers and heterodimers repeated. Some of these interactions are represented in 4.3.

Figure 4.3: Two examples of input files out of 23.

After running our program the final structure was the following (Figure 4).

Figure 4.4: Final structure after running the program with the subunits given.

His3 is a gene of a nucleosome. Figure 4 seems to be a complex nucleosome. At least, it has the proper shape. Therefore, although we do not know the real output, we can conclude that the program worked well.

4.2 Generalisation of the program

Generally, the program will work perfectly for non-repeated/normal heterodimers and heteropolimers defined as it was done.

For homodimers or repeated heterodimers it is more difficult, because a structural superimposition will not rebuild the whole structure. Moreover, we cannot know the correct orientation nor the stoichiometry. For these reasons, the program may fail recontructing this kind of structures.

Chapter 5

Discussion of the project

Chapter 6

Conclusions