Interact Maker

Introduction

Enzymes are very efficient catalysts for biochemical reactions, they provide the suficient speed to the diverse needed reactions in living organisms to keep them functionin. Most enzymes are proteins but RNA can also fill catalytic functions. As life has become more complex so do the metabolic pathways and the enzymes necesssary to follow those big reaction chains, adding more steps in the pathways and more enzymes. Macro-complexes, then, originate as a way of clustering enzymes to speed up the chain of reactions that the metabolites need to go through. Thus, understanding how macro-complexes are structured and how enzymes interact became a key feature to understand.

Another example of the importance of enzyme macro-complexes is the allosteric between the enzymes in the complex. This adds up a whole world of interactions and behaviours of the enzymes depending on different conditions. One remarkable case is the hemoglobin, that each chain regulates the affinity to oxigen of the others.

Independently of the enzymes most of the proteins gain their functionallity when interacting with other proteins as dimers or trimers or any with any number of proteins. In this regard, protein complexes in the interactome provide practical drug targets for drug discovery, as by impeding the formation of the complex one can avoid the activation of its components.

Additionally, macro-complexes are key in the structure of organisms being necesary to build large protein entities that sustain the shape and architecture of their bodys. Such are microtubules or actine filaments, collagen or even the capsides of virus.

The objective of the software provided is to reconstruct pdb's from interactions. This feature is interesting for example when you want to build a huge macrocomplex such a virus capside or similar because they are extremely difficult to crystallize, so, usually, the approach followed is to crystallize only the asymmetrical part and then build the biological assembly using that.

Index

1- Theoretical approach used

2- Program Limitations

3- BioMacromplex Package explanation
    3.1 - PDB.py
    3.2 - PDBaligner.py
    3.3 - PDB_split.py
    3.4 - pymolmanager.py
    3.5 - Sequences.py

4- Gui explanation
5- Further improvements
6- References

1. Theoretical approach:

The aim of this project is to build a protein Macro-complex using as input the pair of interactions that happen in the inner core of the protein. These Macro-complexes can also contain DNA, RNA or other compounds such as ions and small organic molecules.

The idea behind is the following: if we choose one interaction pair as the starting template, we can then place the rest of the chains by superposition. Due to the fact that, in a macro-complex, at least another interaction pair has the same chain, then we can align those chains and after that, move the different chain responsible of the interaction to the template that we are building (Image 1 & 2). Just like a puzzle, if we keep aligning the similar proteins and moving the chains from the interaction to the template we can eventually obtain the correct macro-complex.

Image 1

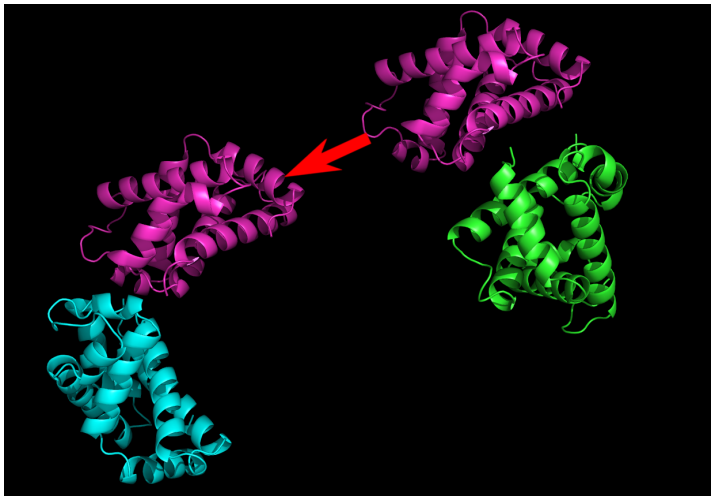
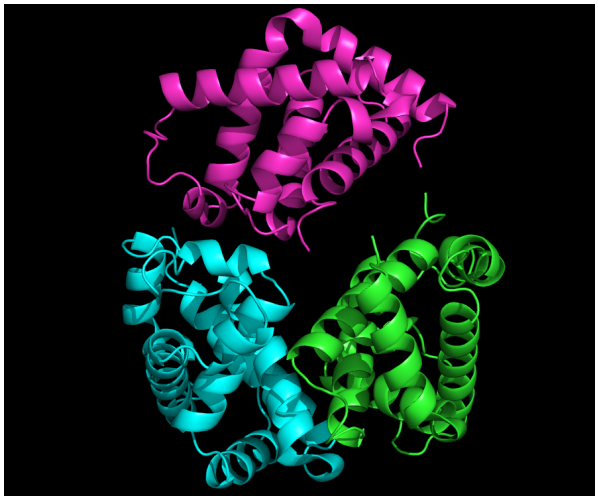


Image 2



To obtain the correct structure using this approach we need to know which proteins have to be aligned and in which order, to avoid making unnecessary clashes by adding more than one time the same chain, and also know when to stop adding chains to that protein because it already has all its interactions.

This can be achieved in different ways. The easiest one would be to use a brute-force approach, which means trying all the combinations and removing those combinations that generate clashes. However, this approach is too computationally dependent, especially when trying to build huge Macro-complexes since the amount of possible combinations grows exponentially. That's why we wanted to find another approach that could be faster and easier to understand, based on the biological properties of the proteins.

Our approach is based on the following: Proteins are usually on solution, and they obtain their folds and interact with each other in order to cover the hydrophobic regions, because by hiding the hydrophobic parts the entropy of the water decreases and the system is more stable. That means that, if two proteins are interacting by one side, that side of the protein will probably contain hydrophobic residues and so, it needs to be covered. That means that in the final macro-complex all the chains will probably have the sides that are interacting always covered.

Knowing that, the program loads into the memory all the residues that are interacting in each different protein and then starts building the macro-complex by forcing the condition that all the residues found to be interacting in a given protein must be interacting with something. Thanks to that, the program can check in each step which proteins are lacking interactions and which of the interaction pairs provided in the input can be used to cover those residues. By doing that we decrease the number of combinations that we have to try. Finally, when all the proteins have all the interactions, that means that the macro-complex has been build and the program knows that it have to finish.

This approach works also with non proteic chains such as DNA and RNA, Because this ensures that all the

nucleotides maintain their Watson and Crick interactions and because it also ensures that the protein is interacting properly with the DNA/RNA strand (interacting in the correct place of the major groove for example).

Another advantage of our approach is that if the user gives redundant interactions pairs or unnecessary ones (not incorrect) our program will still go as fast as if those interactions pairs where not provided. That happens because, as stated before, our program doesn't try all the possible combinations, so if two residues are found to be interacting twice, the program will just ignore one of them, and if more than the necessary interactions are given, those interactions will probably be fulfilled when placing another chain, so the program will not need to make a new superposition.

2. Program limitations

Despite the fact that our program is very robust. (it can deal with different kinds of chains, protein symmetry and even excessive input without affecting it's performance). However, in some special cases the program can fail to give the correct solution or just give a partial correct one, those cases are:

1. When there is more than one correct structure given the data. Our program ensures to give a good looking structure given the data, because it makes sure that all the interactions are on their place, which is very useful when there is only one correct structure. However, when the interactions provided can lead to more than one correct structure our program will give only one of them. The one that the program will give any of the two depending in the order in which the interactions pairs are provided.

2. When incorrect interactions are provided

Another limitation happens when the user give incorrect data, everything is build assuming that the data is correct, so if the data have problems is probable that the output will be incorrect. This will depend in which kind of incorrect data is given and in which position in the order of the input is placed. The only situation in which the incorrect data will not affect the output is when the interaction pair is made of proteins that are not part of the final structure and it's not the first input interaction given (because our program uses the first one as template for the reconstruction).

3. When the structure can grow unlimited such as a microtubul

Our program will be able to correctly build this structure but as the termination condition will never be meted the program will never end. This can be solved by saving the steps and just killing the program when the desired length is achieved, by using the GUI and stopping the job when the structure looks fine to you or by giving to the program a limit number of chains to place.

4. When not all the posible interactions are filled in the real protein

In some cases it happens that one chain in the macrocomplex has an interaction that the rest of the chains of the same type does not have. As our program forces the condition, all residues found to be interacting must be interacting in the final structure, so it will reproduce that interaction for all the chains. However, this is a minoritary case and its solution would be very hard to implement and would significally increase the complexity of the output and the computational time since it would require to make a recursion tree.

3. BioMacromplex

The package contains all the class definitions and functions needed to load, edit and work with pdbs. It also has the functions to manage pymol and to do the reconstruction process. The following is a summary of the capabilities of our package, to see an explanation of how to use each class and function check the Doc folder.

3.1 PDB.py

This is the file that contains all the class definitions to load pdbs and work with them. It was built in a similar way to the pdb parser of the Biopython module, so it can easily interact with other functions provided there, such as the Superimposer module, but it's more complete than the Biopython one. That's also the reason for which this part was built, because when we were developing our solution we found ourselves limited by how Biopython was built.

Advantatges:

Here we will state the advantages of using our pdb parser (PDB.py) rather than the biopython one.

1. In addition to our functions it also has the same function names than the Biopython pdb parser, and it provides the data in the same format. This allows our pdb parser to interact directly with the other modules from biopython without having to change anything.
2. It gives a more flexible tool to rename and handle sequence names, and also a function to filter atoms that the biopython does not provide.
3. It's able to handle the Hetatoms without any problem. This is a major improvement because the biopython lacks this capability, which in our opinion is essential because for most of the cases when you want to work with a protein that has hetatoms you are usually also interested in those atoms, such as the ATP, FADH or metallic ions. This addition is not only interesting for our project (to be able to place the hetatoms in our final macrocomplex) but also to work with pdbs in the future.
4. It has a clearer inheritance hierarchy than the biopython because you can directly initialize the object meanwhile the biopython is built in such a way that the objects are made by the parser so you cannot interact with them directly. This feature gives more flexibility to our parser.

Disadvantatges:

Despite being flexible and having some additional features, our parser is not perfect because it's not able to perform some of the things that the biopython can. Those limitations are the following:

1. Our parser is only able to work with pdb files meanwhile the biopython can also use other formats such as .mmcif. That's one of the reasons for which biopython doesn't allow you to directly declare the structure object without the parser.
2. Our parser does not save any information of the heading. This can be easily implemented in the future, but as it was not necessary for our project, and we had a limited amount of time, we had to leave it for the future.

3.2 PDBaligner.py

This is the core part of our project since it has all the functions needed to build a new pdb from its interactions. We have tried to divide it in different independent functions rather than a giant one to give them more flexibility to work with them in other projects. For instance, we made a function to check the interactions that are independent of the one that superposes the chain based on those interactions.

As stated before, the information about the functions usage can be found in the Doc folder.

3.3 PDB_split.py

Script that divides a pdb into it's different interactions pairs. It does not have a biological interest, but it is useful to make sample input data from existing pdbs in order to test the program.

3.4 pymolmanager.py

Simple script that launches the cmd of pymol to make images.

3.5 Sequences.py

Script made as part of the exercises of the python subject. It is used for the PDB.py to make sequence objects from the found chains in order to have a closer interaction with them, such as checking the molecular weight.

4 Graphical User interface (GUI)

We wanted to provide a GUI that was not only functional but also structured in such a way that it would be easier to escalate and also easier to reuse for other projects. For this reason the GUI is splitted in different classes, one for each frame in the GUI, in order to make them independent. The hierarchy is the following: Main app > Individual Frames > Widgets of each frame. All the individual frames are also built following the same structure in the class declaration, that is: Main variables (such as who is the parent and who is the main app) > widgets declaration > widgets placement inside the frame > other variables.

For example, our GUI currently has two interfaces, one for running the PDB aligner and the other for running the PDB_split, but more "modules" can be easily added, such as an interface to validate the structures or to run MD simulations.

Our GUI also features the capability to launch the jobs as threats, a system to interact with them and a way to avoid spanning multiple threads and to avoid leaving dead processes running in the background.

5 Further Improvements

Due to difficulties met in our approach we could not accomplish the optimization desired and the script escalates very badly. In the computers tested, a reconstruction of a viric capsid of 200 chains lasts about 2 hours to be finished (2:17:23). Thus, if we had more time to improve the program, the main priority would be the optimization. The algorithm checks clashes everytime we add a chain because sometimes, specially in complexes with asymmetric interactions between identical chains, it adds a wrong one that produces clashes.

Additionally, we wanted to perform an energy minimilization using an Amber force field as to evaluate (check energy levels) and improvement of the model. We did not include this in the script due to problems in the instalation of the external program. Depending on the type of molecule (Prot, DNA, RNA or mixed) it needs a different instalation so it limited the versatility of the script. If given more time we would also try to perform an energy optimization, ideally done in python as a module of out package.

6 References:

Since our project needed a way to determine if two residues where interacting or to check if a clash had been produced, we used the following values from the following references:

To check Hidrogen bond distance we use 3.5 A (middle point), taken from: Martz, Eric; Help, Index & Glossary for Protein Explorer, http://www.umass.edu/microbio/chime/pe_beta/pe/protexpl/igloss.htm Jeffrey, George A.; An introduction to hydrogen bonding, Oxford University Press, 1997.

To check Clashes we use 1.5 Amstrongs, which is smaller than the N,C and O vVnderwalls radius. This means that if we find several atoms at this distance they must be clashing.

Values of Vanderwalls radius taken from: http://ww2.chemistry.gatech.edu/~lw26/structure/molecular_interactions/mol_i