SBI-project

A python package for protein complex modeling from protein pairwise interactions.

Table of Contents:

Objective

This python package takes a set of PDB files of pairwise protein interactions and returns a PDB file with the multi protein complex that is formed with these interactions.

Theoretical background

- Superimposition
- Clashes specifications
- Interaction specifications

Input

The input is a set of PDB files holding pairs of proteins interacting. This input does not have to be perfect:

- There may be many protein complexes that can be formed using the interactions provided.
- The sequences of the same protein appearing in several PDB files may not be the same although they have to be very similar.
- The same pair of proteins can interact in different ways.
- A pair of proteins provided by the user does not have to end up in the protein macro-complex.

The user can also introduce some information about the output that wants to obtain. For instance, the stoichiometry of the final complex. They can also provide a subunit limit to define the size of polymeric complexes which otherwise would be virtually endless.

Steps

Classifying the chains

In an input, the same chain or very similar chains may be labelled in a non-consistent way. Because of this, one of the firsts operations performed is the classification of the chains based on their amino-acid sequence. Proteins with a percentage of similarity higher than 95% will be considered the same chain.

Classifying the interactions

At this point, we can know which types of chain are interacting in every input file. We would like to eliminate repeated interactions to reduce the ways we can add chains to the building complex. To do so, we compare pairs of interacting chains and remove from the input pairs of chains that when superimposed to another pair get an RMSD under a certain threshold.

Macro-complex assembly

Once we have processed and classified all the pairwise interactions from the input set we start constructing the macro-complex using superimposition.

The approach adopted in this package is based on a recursive function which will attempt to add as many interactions as each chain forming the macro-complex at that stage can.

By doing so we start with all the different chains the user has found in the input and recursively add chains on top of them. Each node of the recursive tree has an identifier indicating the interactions occurring at that stage, this identifier is saved for further usage. This identifier enables us to assess if a macro-complex at a specific node has already been processed in a previous node and, therefore, stop that branch.

Before we add a new chain to the macro-complex we check that it is not clashing with the other chains already in the structure. The criteria we use to find clashes is that at least one alpha carbon should be at less than 1.2 Angstroms from any atom from another chain.

Furthermore, once we add it we also register the interactions this new chain has with the surrounding ones so as to not attempt to superimpose that interaction in nodes to come thus, reducing processing demand. To find those interactions we check if at least 8 alpha carbons are at less than 8 Angstroms from another alpha carbon from another chain. This ensures that two identical structures will have the same identifier, allowing us to compare them properly and avoiding doing the same process twice. Nevertheless, this process is only done if the option intensive is chosen, because is the only scenario where we compare complex identifiers.

Once the recursive function has finished we are able to build the macro-complex/es obtained from the identifiers at the final nodes of the recursive tree. This enables us to work only with one structure to which we add and remove chains every time we move up/down the tree, therefore, minimizing the memory usage of the computer.

Lastly, we optimize the model using **modeler conjugate gradient**. This function tweaks the side chains so as to minimize the overall energy of the macro-complex.

Macro-complex comparison

If the intensive option is chosen, the algorithm checks if we have already got the current structure. Instead of comparing the structures directly, we compare instances of a class we have created to hold the information regarding the chains and their conectivity in the structure.

This class is a graph with as many nodes as chains the structure have. Each node points to other nodes representing the chains with which it is interacting. The pointers a node uses to relate to other nodes are the kinds of interactions that a protein can stablish.

The comparison is done in three parts. The first two parts are to check if the identifies have the same number of chains and that they have the same number of each type. The third part checks if the connectivity of those chains is the same. The problem is that two different structure can have the same 'first level' interactions.

To be able to diferenciate structures of these characteristics we use the deepness of the comparison. In the figure 1, the two strucures have the same first level

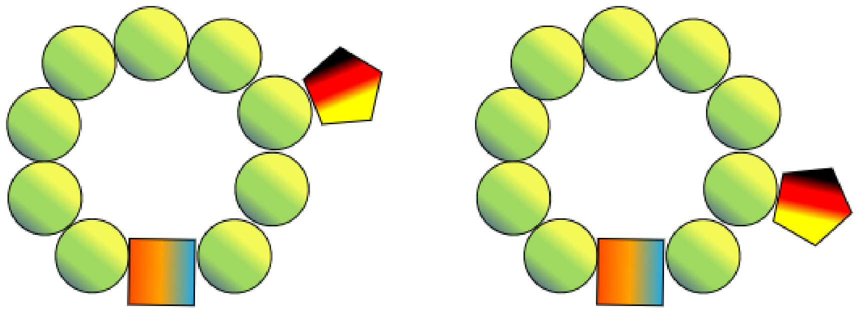


Figure 1: complex comparison

conectivity, but if we check a second level connectivity we see that in the second structure there is one green circle interacting directly with the pentagon and that is two chains apart from the square. There isn't any chain with this second-level connectivity patern so we can say that they are different structures. The proper connectivity deepness to be used depends on the sctructures to compare, the deeper the more reliable is the comparison. By default 4 levels of conectivity are checked

Evaluation

In order to validate the model, we build the z-score energy profile using modeller's get_dope_profile() method/attribute and then using a window of 50 residues to smooth it. This energy profile graph is printed out for the user.

Output

The output of this function are PDB files of the macro-complex/es built from the pairwise interactions. If the user specified passing an already build structure it returns a new directory named XXXX_all_interactions containing the PDB files of the pairwise interactions.

Complementary software

This package includes complementary functions which can be set by the user. * Allows the user to pass PDB interaction files of more than 2 proteins. * Allows the user to pass a full protein complex to be broken into pairwise interaction PDB files. * Allows the user to pass the stoichiometry so as to specify the components of the macro-complex. * Allows the user to pass a number of subunits he wants the model to have in case the macro-complex is theoretically infinite.

Tutorial

In the following section there is a little tutorial in order to make the program of easy use and for the user to get a grasp of all the functionalities available.

In the package there is only one script thought to be executed directly from comand line. It is build_complex.py and can take several comand line arguments.

Command line arguments:

- -i –input: Input a directory containing only the pdb files with the interactions the user wants to process or a macro-complex pdb file if the break option is set.
- -v –verbose: Increase output verbosity, by default is is set to False.
- -k –subunit_limit: Subunit threshold. To use if the protein can theoretically be limitless since if set value but the complex requires more than k subunits to be completed it will halt and stop building it at that point even if it hasn't reached the end. By default it is set to False.
- -opt – optimization: Indicate if you want to optimize the model, by default it is True.
- -int –intensive: Indicate if you want to find all possible structures or just the first one found, by default it will return the first one found, on the other hand, if it is intensive the programm will attempt to find all the possible structures.
- -br –break: Indicate if you want to return all the pairwise interactions or just one of each type. If you pass 'all' the program will output all the pairwise interactions found and if you pass 'unique' it will only return one interaction of each type.
- -st –stoichiometry: If this parameter is provided the program will tell the user how many different chains have been found in the input. Then, one chain at a time it will ask the user to set its stoichiometry. The user will be prompt, having three options:
  - Introduce the absolute frequency of the chain in the final output
  - Pressing return to skip the current chain
  - Introduce 'q' to end the dialog and start the reconstruction process

1. Default settings

python3 macrocomplex_builder.py -i interactions_3kuy * interactions_3kuy is the path to the directory containing all the interaction pdb files that are to be used in the construction of the macro-complex.

2. Assembly specifications

python3 macrocomplex_builder.py -i interactions_3kuy -opt -int

This command is the same as before but this time specifying you want to perform an intensive search and to optimize models obtained. * -opt: specification for the final model to be optimized. * -int: specification for the program to return all the possible structures found.

python3 macrocomplex_builder.py -i interactions_1TUB -k 100

This command in turn specifies the number of subunits the macro-complex has to have. This feature is set for proteins such as tubuline that would otherwise be endless. * -k 100: indicates the macro-complex has to have a maximum of 100 subunits.

python3 macrocomplex_builder-py -i interaction_3kuy -st

This command allows the user to pass the stoichiometry he desires to the model. To do so the program will first detect all the non redundant chains and then return to the user one by one these chains. The user will then have 2 options: i) set a number, which will correspond to the number of chains of that type he wishes the final model to have and ii) press enter, which will be as if no stoichiometry is set for that chain, the program will try to fit as many as it can in the final structure.

3. Get interactions

python3 macrocomplex_builder.py -i interactions_3kuy -br all

This command is different form the previous ones. In this case you pass a fully built macro-complex and the output will be a directory with all the pairwise interactions found in the structure.

python3 macrocomplex_builder.py -i interactions_3kuy -br unique

This command, as the previous one, returns the interactions forming the complex but in this case without any redundancies. It will return a directory with non-redundant interactions.

Theoretical Background

Protein folding

One asumtion that we do is that proteins with similar sequences will have a similar fold.

The folding of a protein in an aqueous solution is a spontaneous process that takes places because of the reduction in the system energy that it implies. The folding is driven by the chemical nature of the residuals of the protein. Hydrophobic residuals tend to be away from the protein surface where they interact with themselves. On the other hand, hydrophilic residues are placed on the surface of the protein. The electric charge of the side chains also affects the way the protein folds. Residues with the same charge are pulled apart while the ones with opposite charge attract to each other.

In vivo there are a lot of other parameters that affect the protein folding, like chaperones, salt concentration, etc. Nevertheless, two proteins with the same or very similar sequence will generally have the same folding.

Superimposition

The superimposition is the main process that allows us to build the complex. It is a process that given two groups of coordinates calculates the linear map that minimices de distances between the points of the two groups. When applied to proteins those points are the atoms forming the backbone of the protein.

The more similar are the relative positions of these two groups of atoms the lower will be the minimal distance or Root-Mean-Square deviation (RMSD). It is calculated adding the distance between each atom and the nearest atom from the other chain.

$$RMSD(v,w) = \sqrt{\frac{\sum_{i=1}^{n}((v_{ix}-w_{ix})^2+(v_{iy}-w_{iy})^2+(v_{iz}-w_{iz})^2)}{n}}$$

Where v and w are two proteins and v_{ix} is the component x of the ith atom of the protein v.

Requirements

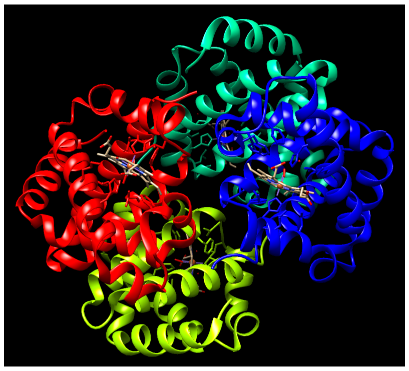
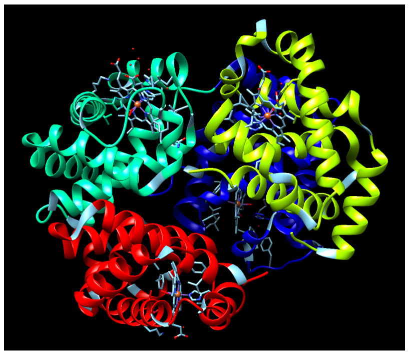
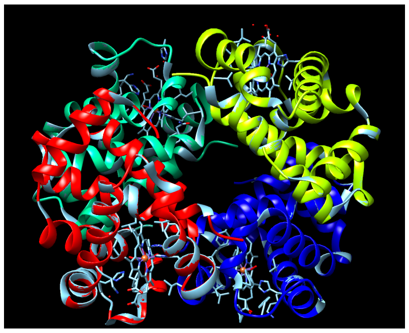
In order to run this package with all its functionalities the user must have several programs: * Python 3 * Python modules * Modeller version 9.19 * Biopython * numpy * gzip * re * copy * argparse

Analysis

In the following section we are going to discuss some examples of inputs-outputs and how it worked for each one.

1gzx - Hemoglobin

1gzx is an example of a complex formed by 2 different chains that form 3 types of interactions.This program is able to properly fully reconstruct this complex with the interactions given as we can see in the images below. Our program, thus, has no problem dealing with this type of interactions. We can see there are no how there are minimal to no differences between the superimposed original 1gzx and the ones the program has built.

		
Original	*Built-No Optimization*	*Optimization*

Original	*Built-No Optimization*	*Optimization*

3kuy - Nucleosome

This is an example of complex formed by 10 different chains, 8 protein and 2 DNA. In the following images we can see the original 3kuy, the 3kuy we built superimposed the real 3kuy and the optimized 3kuy with, again, the original 3kuy superimposed. No flaws are seen in the built complexes, therefore, we can say that the program can work with DNA as well as with protein chains with relatively simple structures.

Original	*Built-No Optimization*	*Optimization*

1g65 - proteasome

When we run a more complex model as 1g65 built out of 30 protein chains, the program can still rebuild it. Due to eliminating redundancy we can see that some chains are slightly shifted with respect to the original molecule but the complex is till built without missing any chain.

Original	*Built-No Optimization*	*Optimization*

5vox - Yeast V-ATPase

When we run a more complex model with a particular chain composition, 5vox, knowing the chain stoichiometry is necessary. If the stoichiometry is not set it will fit all the chains it can in the model even if the original model doesn't have them. If it is set then it will respect the number of chains but it won't necessarily respect the position some of these chains are in the original model. If its a lateral chain/chain compound, for example then it will place it in the first place it can.

Original	*Built-No Optimization*	*Optimization*	*Built-Not Opt-St*	*Built-Opt-St*

5oom - native assembly intermediate of the human mitochondrial ribosome with unfolded interfacial rRNA

5omm is another example of this programs robustness. As we can see in the images below the program was able to reconstruct this complex compounded of 53 chains. It did not place 2 small lateral alpha-helix probably because it didn't have the interaction file since these helices in the pdb files are made out of "unkown" residues and atoms. Therefore, our complex breaker didn't identify the interaction and was unable to incorporate it in the complex building process. Still, the program was able to rebuild it taking into account both protein and RNA chains.

Original	*Built-No Optimization*	*Optimization*

Limitations

The main limitations of this program are the following:

- When performing the reduction of repeated interactions in the input by similarity we set certain thresholds of similarity. These thresholds can lead to consider subunits formed by the same chains and that have the same interaction but that are slightly rotated, so as to fit in a barrel for example, to be deprecated. In these cases we only gather one of these interactions and ultimately when building the complex, barrel in this case, the last subunit may not be added due to clashes derived from an accumulation of little rotation mistakes in the rest of subunits forming the barrel.
- With the recursive approach the amount of processing time when many chains with many interactions are passed and an intensive search is spec-

ified can be very large. This is because the program tries to introduce the chains in all possible orders in order to get every posible structures from the input. Despite two identical structures can't be generated, an structure with complex interactions can generate a many versions of the complex.

- We would have liked to give the user a little more control over the parameters used to define clashes and interactions allowing him to pass a file with the specifications of the parameters. For example, if he wanted to use CA or CB to measure distances, minimum distance to consider a clash/interaction, etc...

Acknowledgement

All the protein visualization was done using Chimera. All the code of this project was programmed using Python 3.6.3.