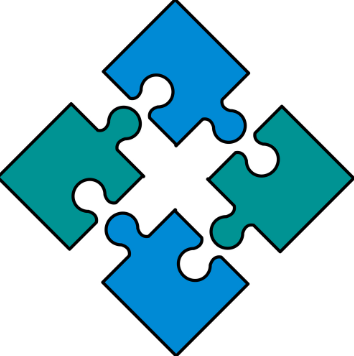
Protein Puzzler

By Aleix Matabacas & Austin McKitrick

**Background**

Proteins are the building blocks of life. They are long chains of amino acids, held together by peptide bonds, and consist of one or more polypeptide. To fully understand a protein's role, one needs to know their function, sequence, and structure. However, not all three of these are always known.

Generally, less is known about the structure of proteins. In bioinformatics, specifically in the Protein Data Bank (PDB), the number of new proteins discovered is greater than the amount of known quaternary structures of proteins. A possible solution to this could be understanding how to model pairwise interactions. Much like the pieces of a puzzle, the pairwise interactions between chains can be combined to create multi-subunit complexes that make up proteins. Using these pairwise interactions to model could provide a unique approach to assembling proteins and studying their structure.

Therefore, the purpose of this project was to create a standalone program to assemble multi-subunit complexes from its individual pairwise interactions.

**Theory/Methods**

Theory

The approach used in this program involved working with individual chains, finding their relationships to other chains, related pairs, and using their relathionships for fitting ...and fitting them together like puzzle pieces. The program locates the chain with the least number of relationships and starts with that chain first — much like beginning a puzzle by looking for the corner pieces. The model is built by checking which chains share relationships, running a collision check between atoms, and merging the chains one-by-one at the same time in all their possible conformations until the program is exhausted and all possible models are produced.

Program Highlights: ##add to this

* + 1. Compatible with protein-protein, protein-RNA, and protein-DNA interactions

in theory,dna-dna ,rna-rna and dna-rna will work too

* + 1. Utilizes the pickle function in Python to store the data more efficiently

for don’t require parse again the fails if you whant run again the program changing the arguments

* + 1. Chains are added one-by-one

are added at the same time creating a new model if exist different possible conformations between them

* + 1. Calculates the center of mass of each atom

this is used for center the starting protein to the 0,0,0 point

* + 1. example generator

Methods

*Data Management & Storage*

Pairwise interactions are provided in PDB files. These files list the atoms within each protein, the 3D coordinates, chain information, etc. First, the PDB files are parsed to separate the two chains in the pairwise interaction. To keep track of which interactions have already been processed, two dictionaries are created to serve as a check.

1. The pairs dictionary lists individual chain objects present in each PDB file. The keys for this dictionary for a PDB file containing chain A and B would look like this: {AB: {A: Chain A object, B: Chain B object}}.
2. The relationships dictionary contains information on each individual chain’s relationship to other chains in the files. The keys for this dictionary are each individual chain processed, and the values are each chain that is related. This dictionary would look like the following: {A: C, B: A,C, C: A,B}.

These dictionaries are stored using a pickle function in Python. This allows for the program to parse the files just once instead of repeatedly if you whant repeat the same model with different arguments. The dictionaries are stored in the user’s working directory.

*Pre-Model Processing*

Next, the program chooses the first chain to start processing. To do this, the program searches the relationships dictionary to find the chain with the lowest number of relationships. If more than one have the minimum of relathionships a brief simulation is made for try too guess what chain creates with less steps a model with all the avaiable chains. This chain is designated as the starting chain for the program.

###pairwise alignment/similarity test

In order to identify regions of similarity between two chains, before make a superposition, we check if the two chains with the same name are equal a pairwise global alignment of the two chains’ sequences are ran by the program.

If the sequences have differences in their chain sequence and :If the alignment score/similarity is greater or equal to the similarity threshold designated by the user at the start of the program, a structural alignment of the two chains is conducted. For this, the residues in first sequence are mapped to their equivalent residue in the second sequence. After this, a two lists of atoms are gathered from the program. These lists of atoms are then used to build the model superimposed.

*Model Building better put this before talk about the aligment*

After the starting chain is selected, the model building begins. Within this procedure, the relationship dictionary is checked again to find what other chain(s) the starting chain is related to. A list of “fixed" atoms from the starting chain and a separate list of “moving” atoms of the chain are created.The common chain are superimposed, and in the complementary chain to that the rotran generated with the “common” superimposition is applyed” The list of “moving” atoms are then rotated and translated appropriately using the rotran attribute rotation and translation of the Superimposer object.

Next, the coordinates of the two lists of atoms of all the model and the chain with the modified coordinates are checked for collisions. This is done using the NeighborSearch function in BioPython to return a list of tuple of coordinates where the two atoms interact. In this step, the radius value that the user manually inputted in the command line is used. If the number of collisions is less than or equal to the threshold indicated by the user, the program continues. The chain is stored for process it in the merge process

Then, the “moving” atoms are superimposed onto the “fixed” atoms using the Superimposer object. This function also minimizes the root-mean-square-deviation, or RMSD, guaranteeing the atoms are as close to each other as possible. You talk about that earlier, better merge with the previous paragraph

The merge part consist in obtain all the possible configurations of the “candidates” (chains with the coordinates changed, waiting for be joined) we can have if the last(s) chains added was sorrounded by all of their posible relathionships. once the possible configurations are gated, taking in account the clashes between the “candidates” one model is generated for each configuration

After the chains have been superimposed,merged this complex becomes the main model the program is runned again for all the generated models in a recursive way and then process starts over again to add another chain(s).

This is done until all chains available are merged.

*Model Analysis*

#pymol

######center of mass this only is used for center the starting chain to the 0,0,0 point and don’t work whith unecessarily long numbers

##Tutorial

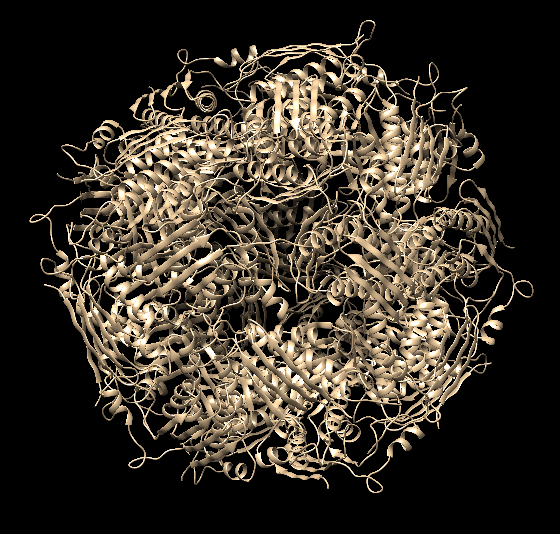
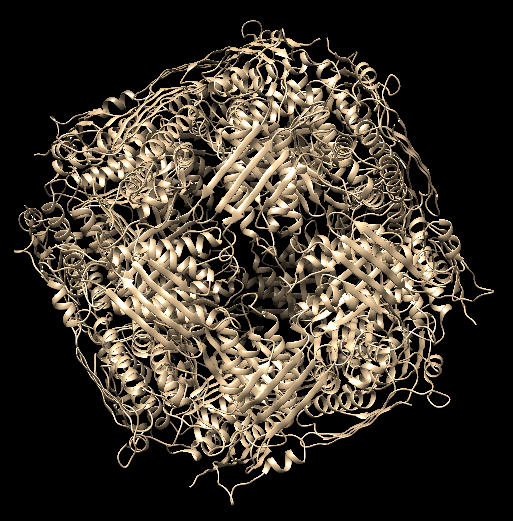
how to install program

how to run program

* pymol
* radius, collisions accepted

**Examples/Analysis**

Example 1

The first example corresponds with the 23 input files provided by Prof. Javier Garcia. These PDB files provide information on a histone protein. Using the program, the following structure was obtained in ~50 seconds. Each image shows the protein at a different angle.

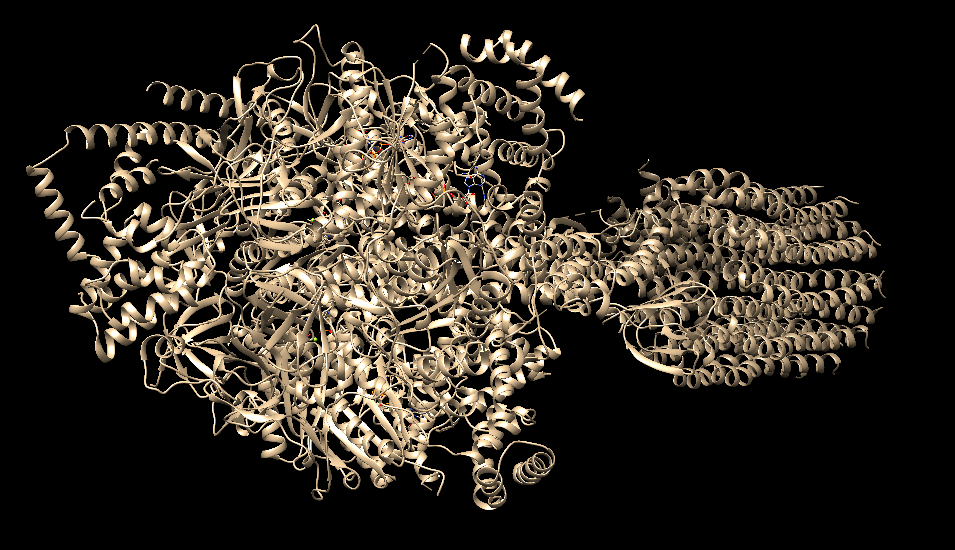
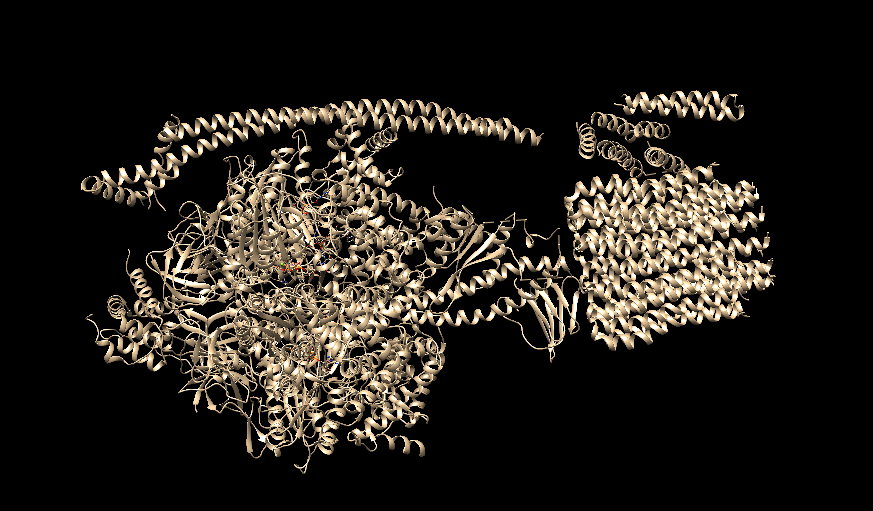
1.

1.

2.

Make a matchmaking with the original protein for be able of show that the obtained model is the expected one

Example 2

This example corresponds with ATP synthase, an enzyme involved in creating the energy storage molecule adenosine triphosphate, or ATP. The input for this structure was 28 PDB files containing individual pairwise interactions. The program produced the following structure in ~3.5 minutes. Each image shows the structure at a different angle. Again, make a matchkmaking with the original one

3.

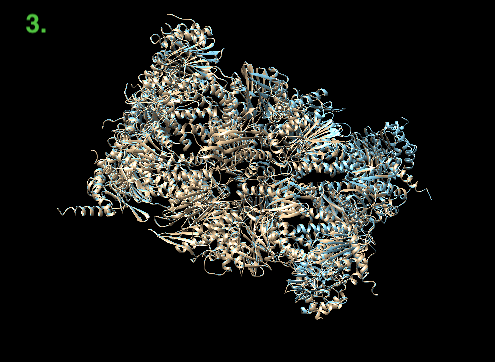
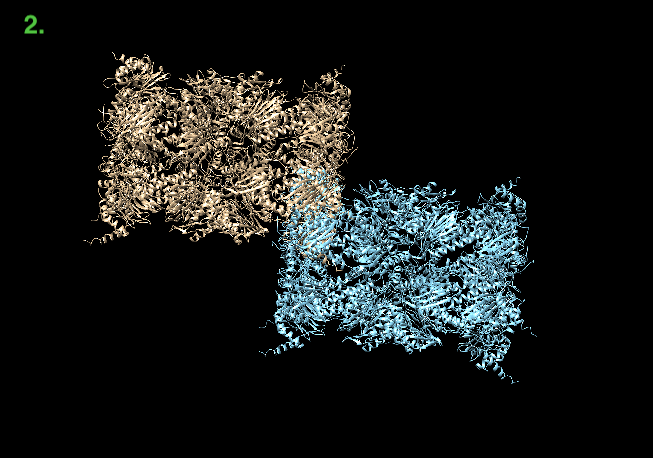
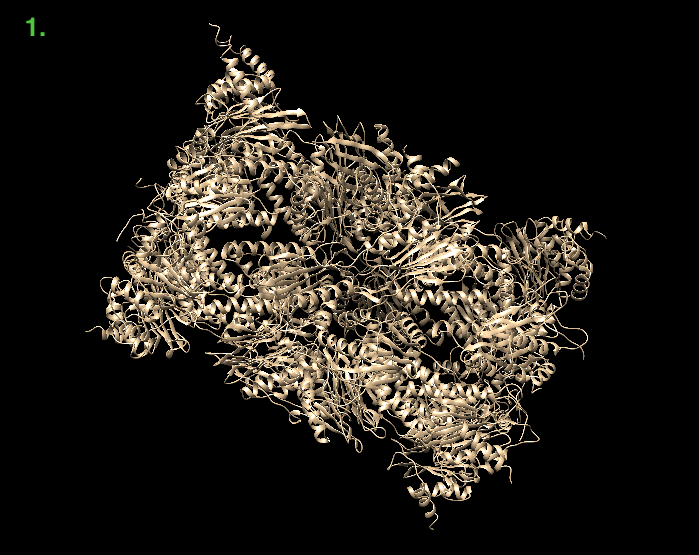
4.

Example 3

This example was for a proteasome structure (PDB ID: 4r3o). The input for this example was 27 PDB files, each with a pairwise interaction. As we see in the first image, the program successfully produced a model for the proteasome in ~2 minutes. To compare, the PDB ID code was used to fetch the actual structure in Chimera. The MatchMaker tool was used to superimpose the two structures and to check if the atoms aligned correctly. The third image shows the result of MatchMaker — a perfect alignment of the two structures.

5.

6.

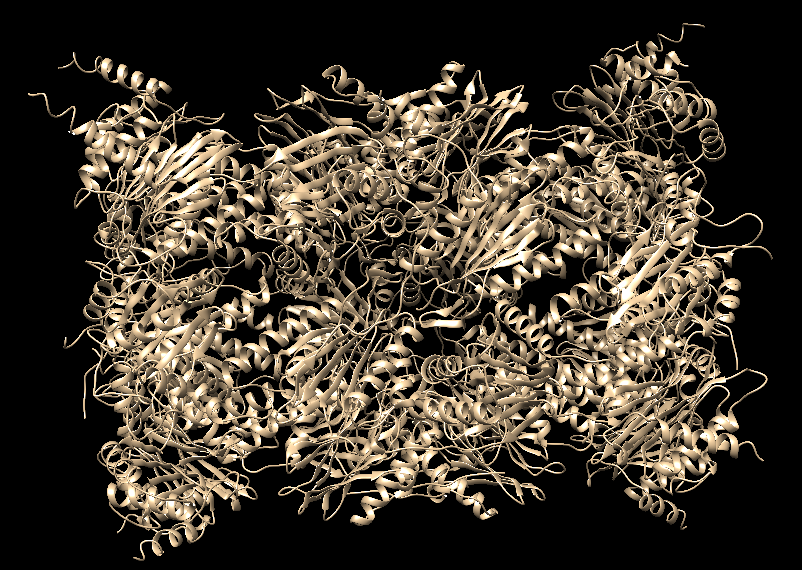
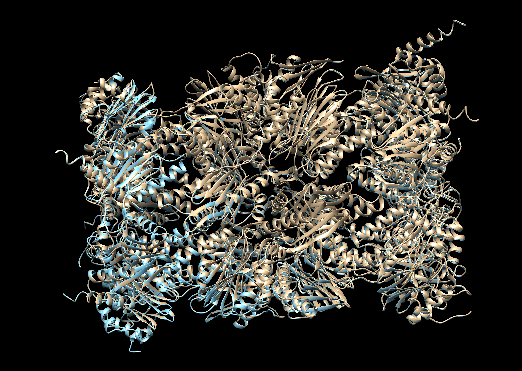


7.

Example 4

This example shows the proteasome used in Example 3, but this time the example generator program was altered. Instead of merging every possible interaction of chains, the program was redesigned to not merge identical interactions that have already been merged. Now, instead of give a new name to each chain in the original pdb file, the similar proteins are searched and gives to them the same name if the similarity is equal or bigger than the treshold(95% used in this example). In this example this is remarkable because we reduce the number of input files by half but we obtain the same output. This improved the efficiency of the program by ~20 seconds, while still producing the same result. The first image below shows the proteasome structure. The second image shows the structure aligned, via MatchMaker with the correct structure loaded from PDB. Once again, a perfect alignment is produced by the program.

In this example our program uses the superimposition, because not all the chains with the same name are equal



8.

9.

Example 5

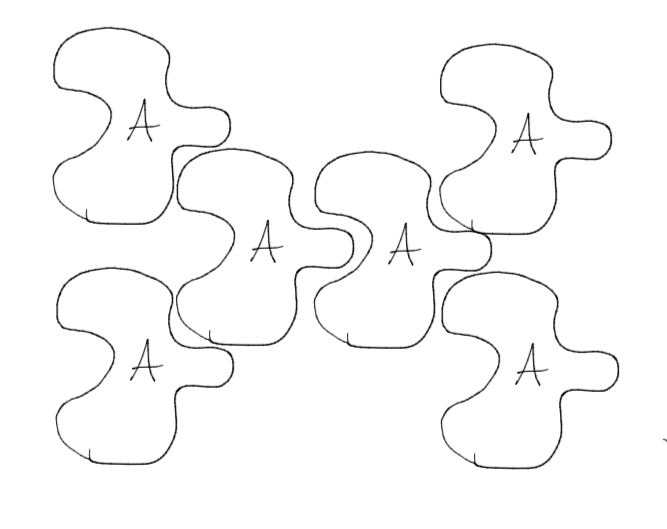
In addition, the program can model files containing information on protein-DNA interactions. The images below correspond to a nucleosome (PDB ID: 3kuy). The image on the left, which was produced in ~1.5 minutes, shows the output of the program. The structure from the PDB was loaded in next (shown in the middle image). The MatchMaker tool was used to superimpose the two in the image on the left. The program was able to produce an accurate model of nucleosome.

10.

11.

12.

**Limitations**

This program has some limitations when structuring pairwise interactions. For one, the program works by parsing the interactions into individual chains and adding the chains one-by-one adding all the possible additions to one new chain added to the main model. Multi-chain complexes can’t be added to the main model. In consequence, the case of a multi-chain complex that can limit the addition of chains in the same step of the chain than joins to the model. can’t be reproduced. A draw will be nice

Another limitation is that the program only can store the same pair once doesn’t identify identical chains. This means that if the input files contain the same chains interaction repeatedly, those chains will be treated as unique and added to the main model accordingly. overlaped in the pairs dictionary The result could be something like the image on the right, where all the input chains are the same. But you can rename the chains for allow the program to work, add the same img but with the names changed

In addition, our program is unable to produce a model if the radius allowed is below

