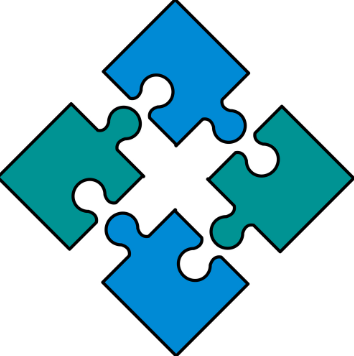
Protein Puzzler

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, 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 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
    2. Utilizes the pickle function in Python to store the data more efficiently
    3. Chains are added one-by-one
    4. Calculates the center of mass of each atom

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. 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. 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, a pairwise global alignment of the two chains’ sequences are ran by the program.

If the alignment 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.

*Model Building*

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 list of “moving” atoms are then rotated and translated appropriately using the rotran attribute of the Superimposer object.

Next, the coordinates of the two lists of atoms are checked for collisions. This is done using the NeighborSearch function in BioPython to return a 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.

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.

After the chains have been superimposed, this complex becomes the main model and then process starts over again to add another chain.

This is done until all chains available are merged.

*Model Analysis*

#pymol

######center of mass

##Tutorial

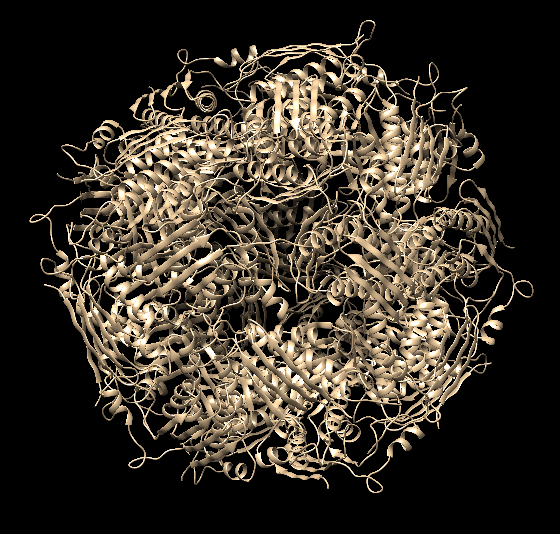
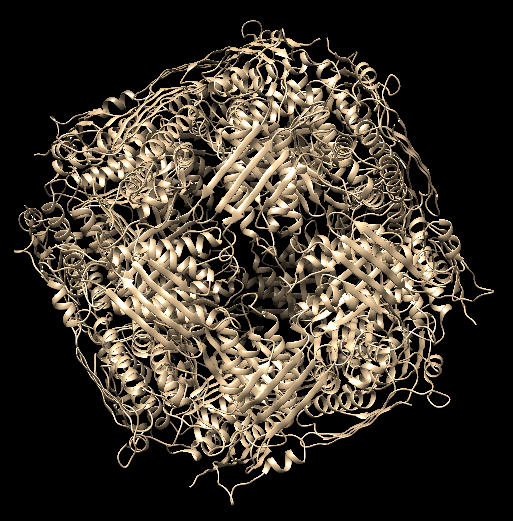
how to install program

how to run program

* pymol
* radius, collisions accepted

**Examples/Analysis**

Example 1

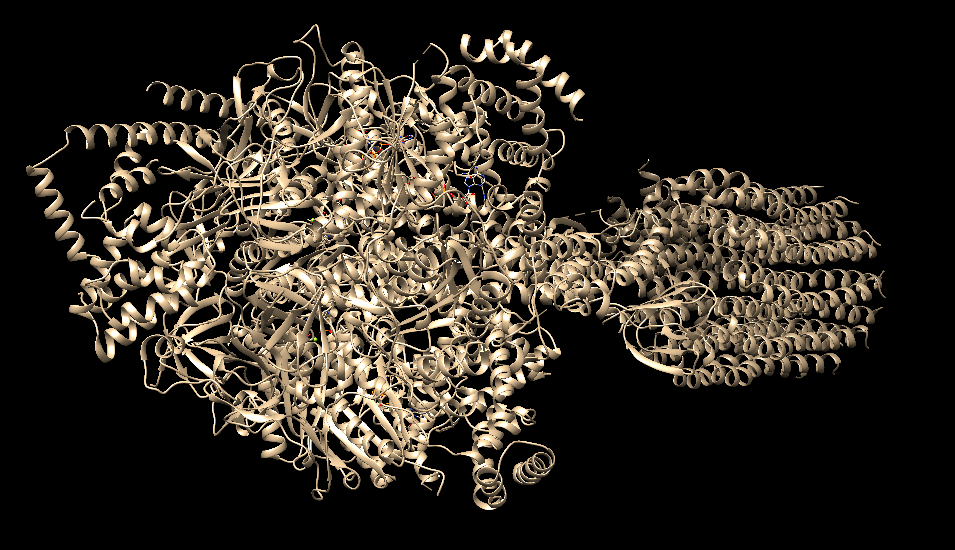
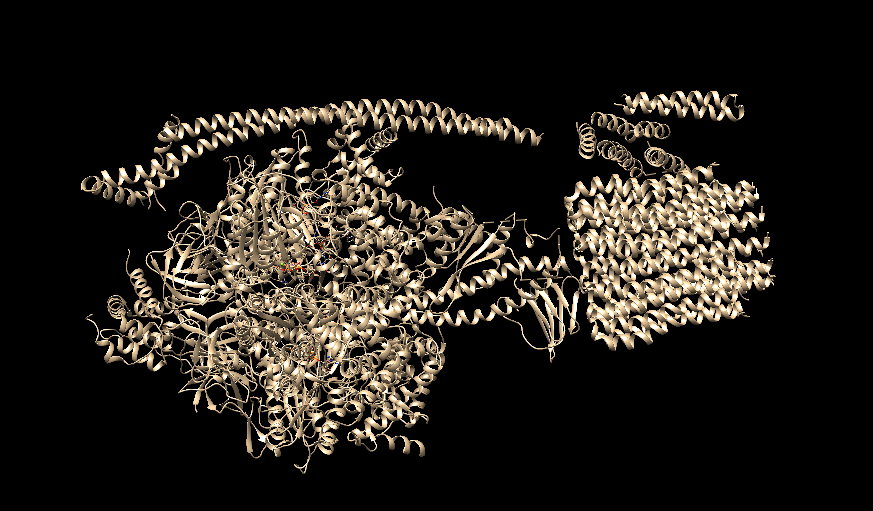
The first example corresponds with the 23 input files provided by Prof. Javier Garcia. These PDB files provide information on a histone protein. Using the program, the following structure was obtained in ~50 seconds. Each image shows the protein at a different angle. 

1.

1.

2.

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.

4.

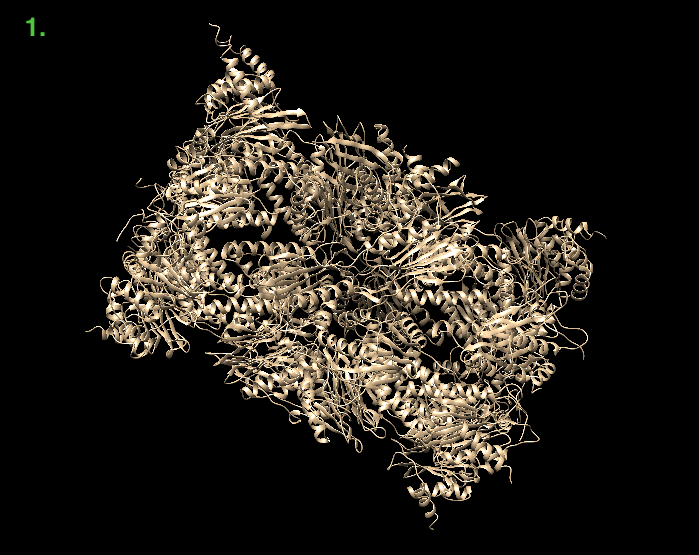
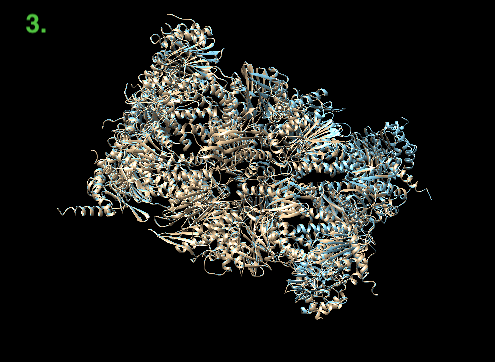
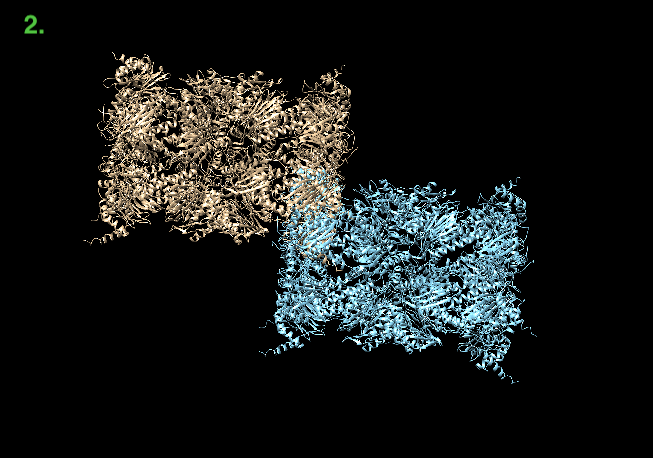
3.

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.

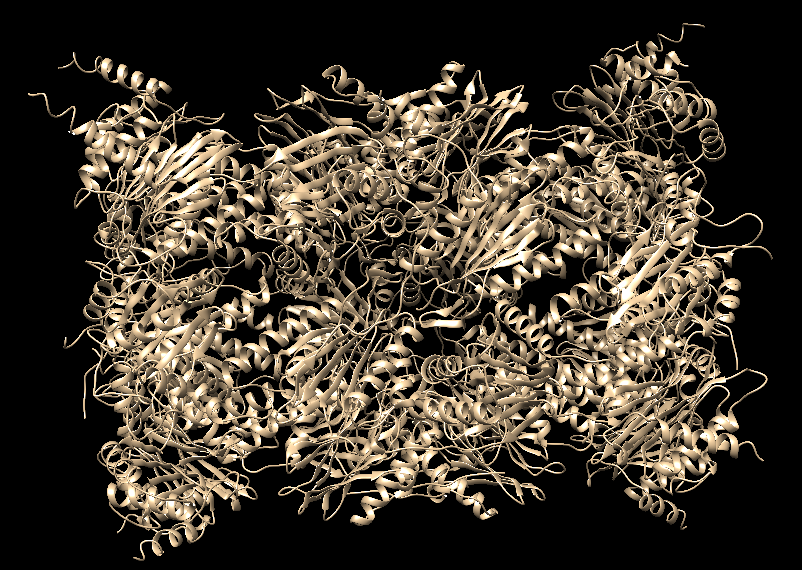
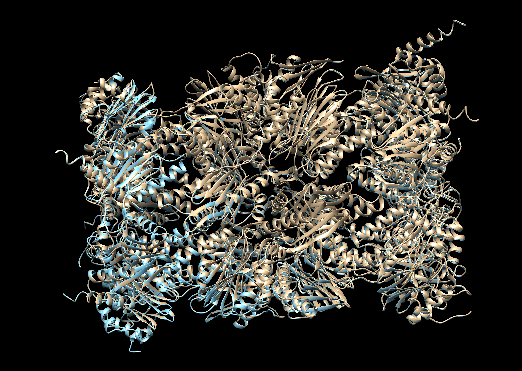
6.



7.

Example 4

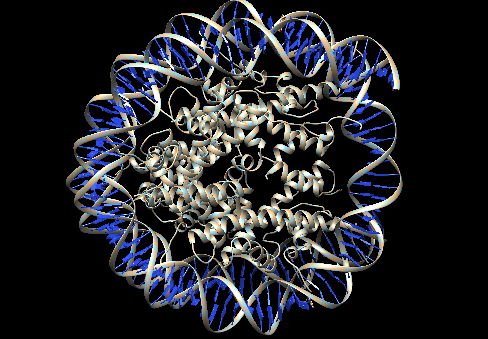
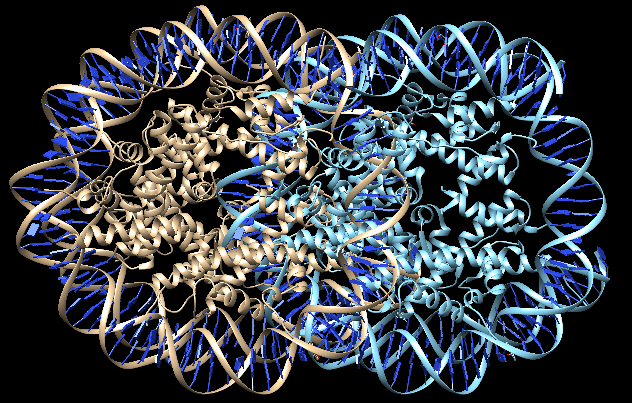
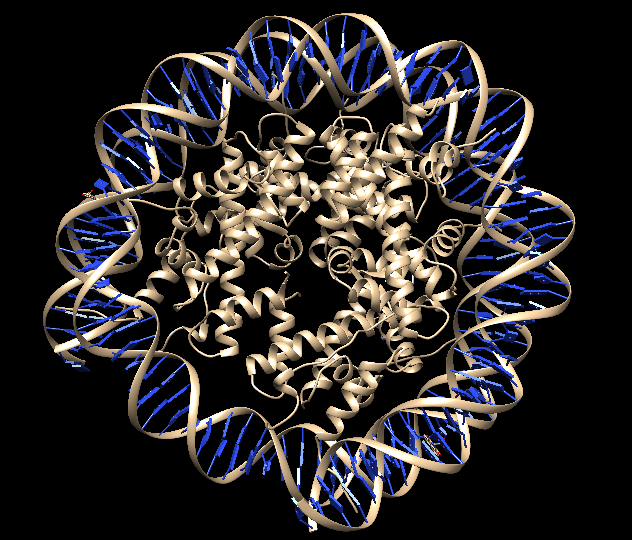
This example shows the proteasome used in Example 3, but this time the program was altered. Instead of merging every possible interaction of chains, the program was redesigned to not merge identical interactions that have already been merged. This improved the efficiency of the program by ~20 seconds, 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.



8.

9.

Example 5

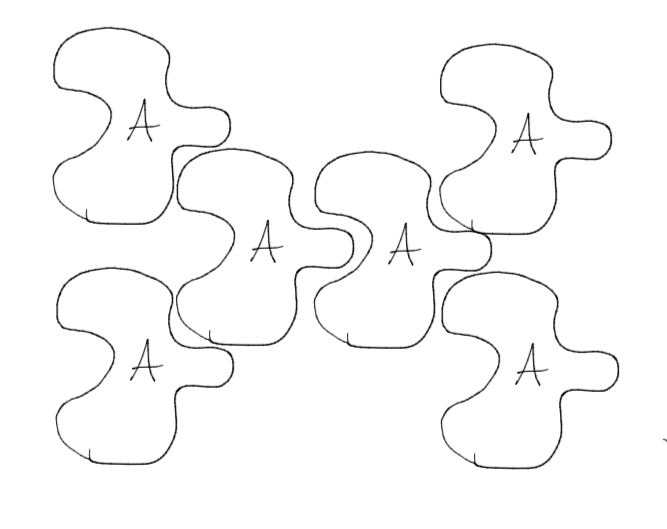
In addition, the program can model files containing information on protein-DNA interactions. The images below correspond to a nucleosome (PDB ID: 3kuy). The image on the left, which was produced in ~1.5 minutes, shows the output of the program. The structure from the PDB was loaded in next (shown in 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 to the main model. Multi-chain complexes can’t be added to the main model.

Another limitation is that the program doesn’t identify identical chains. This means that if the input files contain the same chains repeatedly, those chains will be treated as unique and added to the main model accordingly. The result could be something like the image on the right, where all the input chains are the same.

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

