# 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.

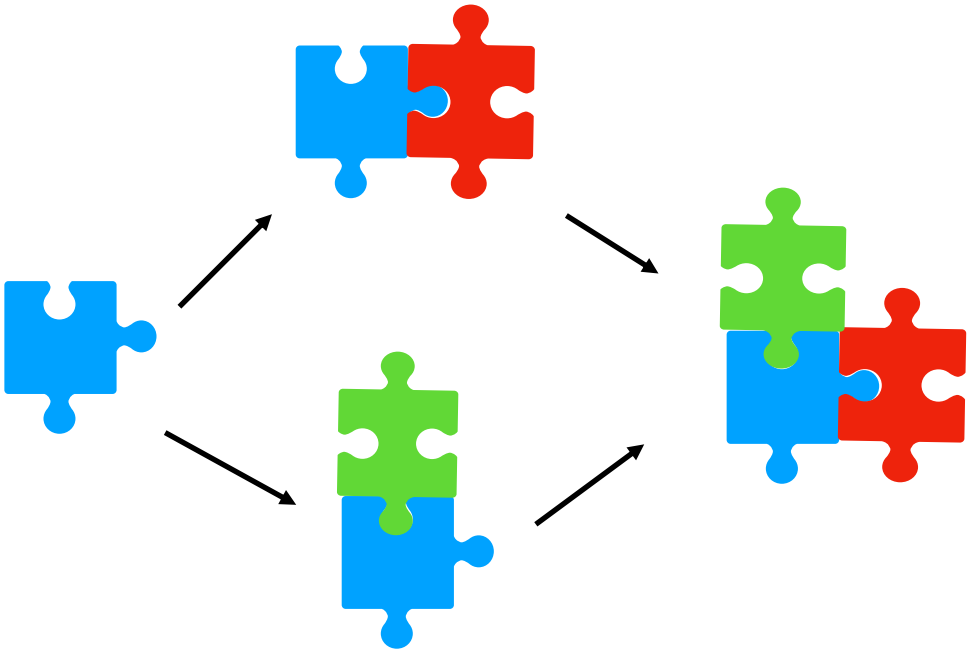
Generally, less is known about the structure of proteins. In bioinformatics, specifically in the Protein Data Bank (PDB), the number of new proteins discovered is greater than the amount of known quaternary structures of proteins. A possible solution to this could be understanding how to model pairwise 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 related pairs of chains and using those relationships to fit the chains 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 at the same time with all their possible conformations. This process is conducted until the program is exhausted and all possible models are produced.



Program Highlights:

1. Compatible with protein-protein, protein-DNA, protein-RNA, DNA-DNA , RNA-RNA and DNA-RNA data
2. Utilizes the pickle function in Python so files can be parsed only once instead of every time the program is ran
3. Chains are added at the same time. This allows for new models to be considered if different possible conformations exist.
4. Example Generator tool that can parse a PDB file into pairwise interactions

## 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 dictionnaires 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 multiple times in case the user wants to repeat the same model with different arguments.

### *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 chain have the minimum of relationships, a brief simulation is made to guess which chain can create a model with all the available chains in the least amount of steps. This chain is designated as the starting chain for the program.

### *Model Building*

After this step, the relationship dictionary is checked again to find what other chain(s) the starting chain is related to. The last added common chain in the model is superimposed. In the complementary chain, the "rotran" (rotation and translation) is generated when the "common" superimposition is made by the Superimposer object. Next, the

non-common chain is moved This function also minimizes the root-mean-square-deviation, or RMSD, guaranteeing the atoms are as close to each other as possible.

During the Superimposer step, the program checks if two chains with the same name are equal. This is done by running a pairwise global alignment of the two chains' sequences. 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.

The coordinates of atoms in 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 tuples containing the x,y,z 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 chain, or "candidate", is stored for merging.

Next, the program obtains all possible configurations of the "candidates" if the last chain added was surrounded by all of its possible relationships. Once the possible configurations are found and collisions have been checked, the "candidates" model is generated for each configuration.

After the chains have been merged, the program continues running recursively until all the possible models are produced. Once finished, the program starts over to add another chain(s).

**Installation/Tutorial**

Installation

To install Protein Puzzler, enter the following line into the terminal.

```
sudo python3 ppzzer_installator.py install
```

The design of the Protein Puzzler package is displayed below:

```
ppzzer.py
p_puzzle/
        __init__.py
        constructor/
                checker.py
                core.py
                start.py
        ex_generator/
                generators.py
        parser/
                pdb_files.py
        support/
                settings.py
```

Tutorial

To begin using Protein Puzzler, create a folder that contains the ppzzer.py script, p_puzzle folder, and the PDB file you want to model. To become familiar with the available arguments in the package, the user should run the following command in the terminal:

```
ppzzer.py -h
```

The available arguments and other information will be displayed in the terminal. It is recommended that the user use the “-v” or “-vv” to enable verbose mode. This will allow for additional information to be displayed while working with the program.

If the input PDB file hasn't been parsed into pairwise interactions, the user will first need to do this using the Example Generator tool. In this example, the 4r3o.pdb file will be used. The user can split the file by giving each chain a different name (using exgen's "1" option) or they can split the file giving similar chains the same name (using exgen's "2" option). To do this, navigate in the terminal to the folder you created. Then, run the following command:

```
ppzzer.py exgen 4r3o.pdb 1
```

If the user check's their folder, they will see that a new folder has been created, named "4r3o" in this example, which contains all the input PDB files for pairwise interactions.

Next, the user will navigate to the new "4r3o" folder containing the PDB files. From here, they will input the following command into the terminal:

```
../ppzzer.py
```

After the program has completed, the pairs dictionary, relationship dictionary, and model PDB file will be saved to the working directory. Using a tool like Chimera, the user can open the model PDB file to observe the structure produced.

*Additional Information*

If the user wants to rerun the program with the same input files, the user can delete the model and add "-p" to the command line since the pickle already exists.

```
../ppzzer.py -p
```

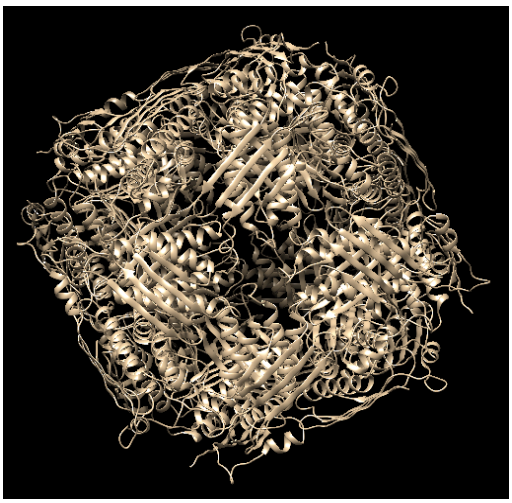
Note that the program runs with default values for similarity percentage, collisions accepted, and radius. In order to change these settings for modelling, the user will need to include “-%”, “-c”, and/or “-r” in the command line followed by the desired value. Below is an example:

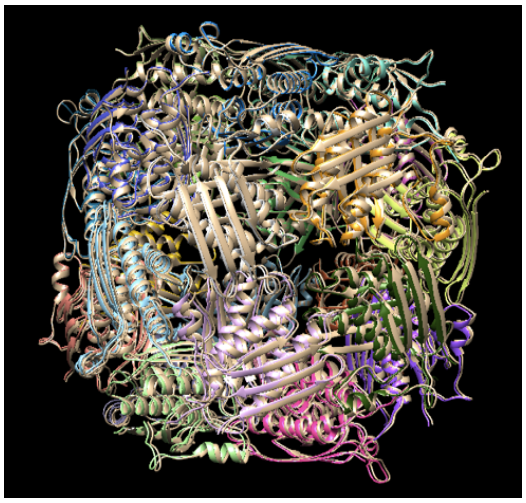
```
../ppzzer.py -p -c 15 -r 3
```

**Examples/Analysis**

Example 1

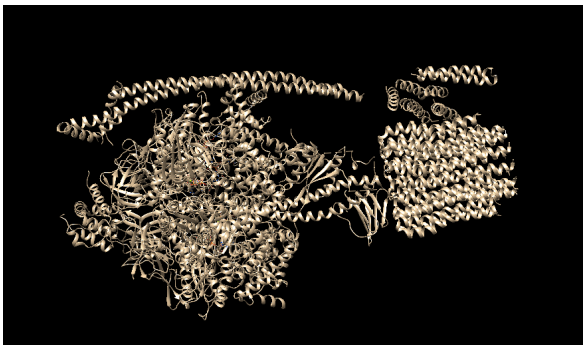
The first example corresponds with the 23 input files provided by Prof. Javier Garcia. These PDB files provide information on a histone protein. Using the program, the following structure was obtained in ~50 seconds (not including file parsing). The first image shows the structure produced by the program. The second image shows the structure aligned with all individual interactions using MatchMaker. The output by the program appears to be the correct one.

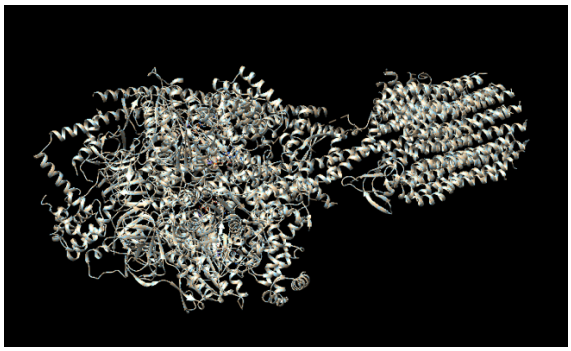




Example 2

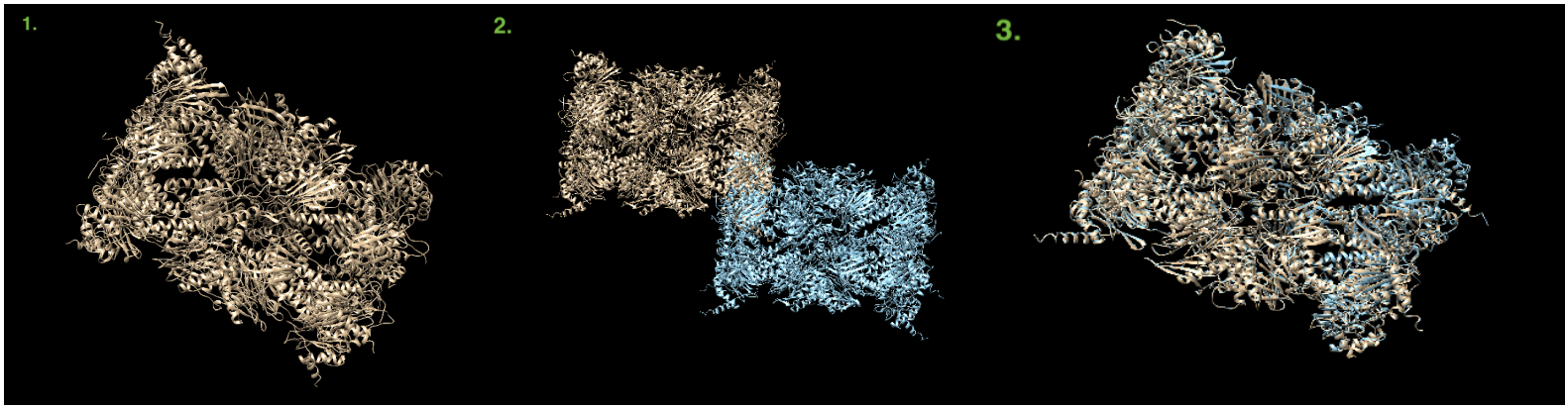
This example corresponds with ATP synthase, an enzyme involved in creating the energy storage molecule adenosine triphosphate, or ATP. The input for this structure was 28 PDB files containing individual pairwise interactions. The program produced the following structure in ~3.5 minutes (not including file parsing). Each image shows the structure. The second image shows the alignment using MatchMaker, indicating that the structure produced by the program is the one that is expected.





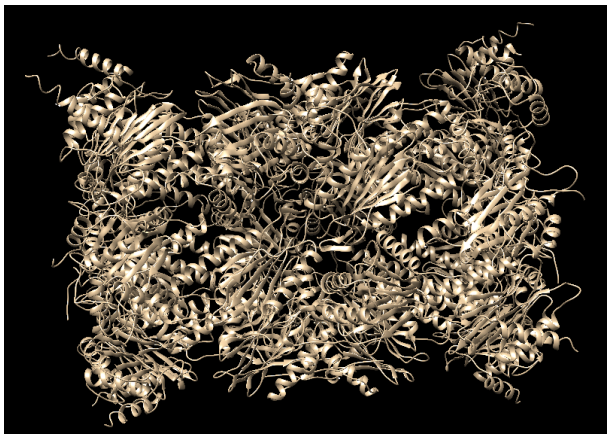
Example 3

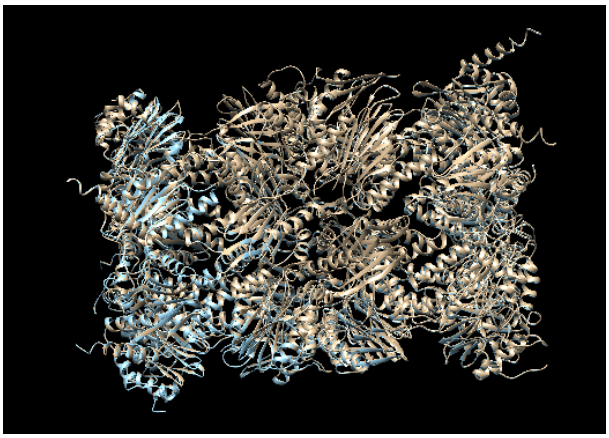
This example was for a proteasome structure (PDB ID: 4r3o). The input for this example was 27 PDB files, each with a pairwise interaction. As we see in the first image, the program successfully produced a model for the proteasome in ~2 minutes (not including file parsing). To compare, the PDB ID code 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.



Example 4

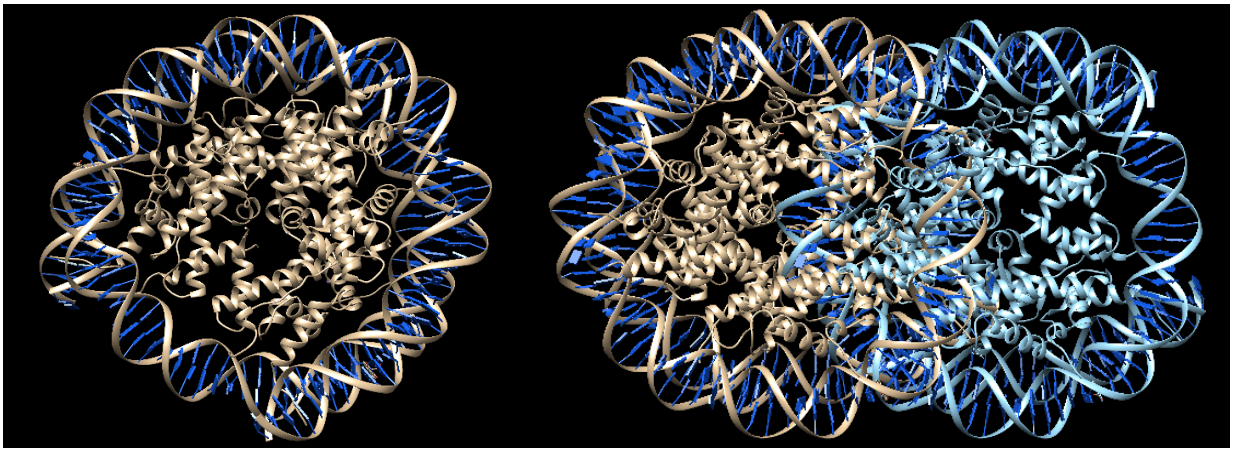
This example shows the proteasome used in Example 3, but this time the example generator program was altered. Now, instead of giving a new name to each chain in the original PDB file, the loaded proteins are searched first and checked for similarity. If this protein has a similarity equal to or bigger than the threshold (95% used in this example), the protein is given the same name. The program then runs as described previously. Using this program update, the number of input files are reduced by half and the same output is obtained by the program. This improved the efficiency of the program by ~20 seconds as well. 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.

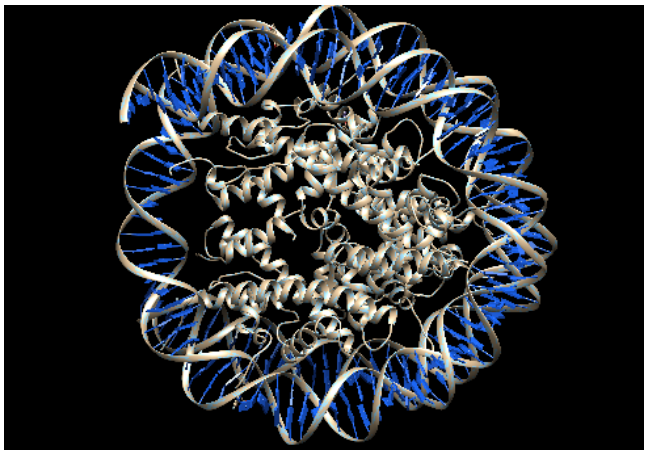




Example 5

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 (not including file parsing), shows the output of the program. The structure from the PDB was loaded in next (shown in the image on the right). The MatchMaker tool was used to superimpose the two in the image at the bottom. The program was able to produce an accurate model of the nucleosome.





**Limitations**

This program has some limitations when structuring pairwise interactions. For one, the program works by parsing the interactions into individual chains and adding all the possible additions to one new chain. Therefore, the program doesn't allow for multi-chain complexes to be added. The individual chains that make up the complexes are added but the configuration those chains in the complexes had might not be identical to their configuration in the new model.

Another limitation is that the program only can store the same pair once. This means that if the input files contain the same interaction repeatedly, those chains will be treated as unique will overlap the existing chains in the pairs dictionary. The result could be something like the image on the right, where all the input chains are the same. A solution to this problem, shown below, would be to rename the chains to allow the program to run