SBI-Python project

Constructing macromolecular complexes.

Ramon Massoni and Winona Oliveros

TABLE OF CONTENTS

Description

A python package for macrocomplex construction given protein pairwise interactions.

Problem

Given a set of interacting pairs (prot-prot), reconstruct the complete macrocomplex and return a PDB file (or files) with the possible protein macrocomplexes build.

Installation

RECOMENDED

You can download our package using Git with the next command. We also recommend to create a directory named "complexes":

```
git clone https://github.com/massonix/SBI-project.git
cd SBI-Project
mkdir complexes
```

The directory SBI-Project contains the following files and directories:

- README.md, README.pdf: the files containing the tutorial and information about our application.
- multicomplex.py: the command-line script.
- multifunctions.py: a module required by multicomplex.py that contains all the functions and classes.
- example1, example2 and example3: directories with several PDB files that serve as examples of input to the programme, as we will see later.

- complexes: an empty folder where the created complexes will be saved.
- raw_pdbs: the raw PDB files from which we extracted the example pairwise interactions.
- gui2.py

Installation via setup

To install the package you must unzip the protcomp-X.X folder

```
tar -xvzf protcomp-X.X.tar.gz
```

And then run the following command inside the protcomp folder:

```
sudo python3 setup.py install
```

Installation via PIP

You can also install the program using pip, the main python files (multicomplex.py and multifunctions.py) as well as some files with additional information will be installed in your default site-packages folder for python.

```
pip3 install protcomp
```

General Information

Input Files

This program needs an input of PDB files holding the protein pairwise interactions needed to reconstruct the desired macrocomplex. The program can handle those scenarios:

- The same sequence appearing in different PDB files has not to be identical, we can handle 95% of identity.
- The same sequence appearing in different files can have different names.

Background and algorithm

Many proteins consists of several polypeptide chains that assembly into multi-subunit complexes. Such assembly is known as the quaternary structure of a protein, and it plays a major role in defining a protein's function, localization and activity. Examples include proteins as important as the nucleosome, the spliceosome or the proteasome, among others. Nevertheless, understanding how individual subunits assemble into larger complexes is no easy task. In this project, we developed a stand-alone application to create multi-subunit complexes out of its individual pairwise interactions.

We approached the former problem by strucurally superimposing structures that had at least two identical subunits (polypeptide chains that have a pairwise sequence identity >= 95%). We considered as feasible complexes those superimpositions that had no clashes, which means that the backbone of one structure did not protrude the backbone of the other. We then called the function again giving as input the output of the first call (i.e. the resulting complex), establishing in this manner a recursive algorithm. Nevertheless, as several complexes could be found from the same input, the computational-cost grew exponentially. To overcome such challenge, we applied two restraints.

First, as the same complex could be achieved following different paths, we ensured that each new complex was unique before calling the function again. Again, we did this by superimposing each complex with all the former ones. If the complex was repeated, at least one superimposition resulted in an extremely low RMSD (< 2). In such cases, the complex was discarded.

Second, we were very restrictive with the stoichiometry of the complex. The stoichiometry of a protein complex refers to the composition of its subunits, and it is specified by a formula. Such formula consists of uppercase letters with integer coefficients. The letters refer to unique subunits, and the coefficients to the amount of each unique subunit. For instance, hemoglobin has two alpha and two beta subunits, and thus a stoichiometry of A2B2. Larger coeffiecients must be used first, i.e A2B3 is not valid, A3B2 is. We ask the user to provide the stoichiometry he or she desires to obtain in the final complex, and hence all the complexes that do not match such stoichiometry are ruled out, achieving a more efficient program. Only those complexes which meet the stoichiometry are saved in individual PDB files.

Finally, we also allow the user to optimize the resulting complexes, which means that the side chain and loops may be restructured to minimize the energy and end up with a more reliable complex. To do so we use MODELLER, a python program to produce homolgy modelling and to optimize the final structures.

Tutorial

The following section explains the user how to use our application to make the most out of it.

Command-line arguments

- -i –input: directory containing the PDB files that will serve as input to the program.
- -o –output: directory where the macrocomplexes will be saved as PDB files
- -v –verbose: whether or not the user wants to keep track of what the programme is doing.
- -st –stoich: the stoichiometry of the multi-subunit complexes.
- -opt –optimization: if the user wants to optimize the resulting complexes.
- -cn –contact_num: number of amino acids that are allowed to protrude after each superimposition (default=5).
- -f –files_num: maximum number of files (complexes) the user wants to obtain.

To get a better understanding of how to run the programme properly, we show 3 different scenarios that represent the heterogeneity of inputs that may be provided.

Example 1

The first example corresponds to the phosphate dehydratase (pdb id: 2F1D). As we can see in the splited file we provide, (2f1d_split.pdb), this is an Homo 8-mer (stoichiometry: A8). To achieve this complex we can run:

```
python3 multicomplex.py -i example1 -o complexes/ -st "A8" -f10 -v
```

Where example1 is the directory containing all input files, complexes is the directory where the output files will be saved, “A8” is the stoichiometry (i.e. 8 equal chains), -f10 means that the programme will stop after 10 complexes, and -v means that the standard error will be printed. In this case, we can see how multiple complexes are output. This is because at the recurssion level the function stopped (k=8), the programme could still superimpose more structures without finding clashes. Interestingly, we see how the complex9.pdb is the same as the 2f1d_split (inside raw_pdbs), except for one chain.

As more chains can be added, we can rerun the programme with a higher stoichiometry: A24. This time though, it will take longer to execute, so we will limit the max number of files to 1. Before doing so, we must move all the previous complexes to a different file or change the outdir:

```
python3 multicomplex.py -i example1 -o complexes/ -st "A24" -f1 -v
```

As in this example we only have one unique chain, if we try to get a hetero k-mer, we will get an error:

```
python3 multicomplex.py -i example1 -o complexes/ -st "A2B2" -f1 -v
```

Example 2

The second example was kindly provided by Prof. Baldo Oliva. In this case, we have two unique chains, so we can build hetero k-mers:

```
python3 multicomplex.py -i example2 -o complexes -st "A6B6" -f1 -v
```

As we can see, it completes the recursion pretty fast. However, if we try to add two more chains the resulting complexes will clash, so it will take too long and do not give any result:

```
python3 multicomplex.py -i example2 -o complexes -st "A7B7" -f1 -v
```

Thus, if we are not interested in any particular stoichiometry, it is a good practice to start with few recursions, check the output, and increase it or not accordingly. Finally, in the case were we have more than one unique chain, we can still try to find Homo k-mers

```
python3 multicomplex.py -i example2 -o complexes -st "A6" -f1 -v
```

Example 2 with optimization

This example shows how to run the program with optimization:

```
python3 multicomplex.py -i example2 -o complexes -st "A6B6" -f1 -v -opt
```

As we can see, the program will create several additional files on the folder from which we are calling the program execution: * complexX.rsr : this file contains the restraints used to optimize the structure. * complexX.ini : this file contains the initial MODELLER model. * complexX.D00000001 and complexX.D9999XXXX.pdb : those files contain the progress of optimization. * complexX_optimized.pdb : This file contains the final optimized structure.

Example 3

The third example corresponds to the 20S proteasome (pdb id: 1G65). This represents an extreme case, where every chain is different. The proteasome is a Hetero 28-mer (stoichiometry: A2B2C2D2E2F2G2H2I2J2K2L2M2N2). For our purposes, we splited in two and obtained a stoichiometry of: A1B1C1D1E1F1G1H1I1J1K1L1M1N1 (1g65_split, inside raw_pdbs). We will input increasing stoichiometries and assess the scalability of the programme accordingly:

```
python3 multicomplex.py -i example3 -o complexes -st "A1B1C1D1" -f4 -v
python3 multicomplex.py -i example3 -o complexes -st "A1B1C1D1E1F1G1" -f4 -v
python3 multicomplex.py -i example3 -o complexes -st "A1B1C1D1E1F1G1H1I1J1K1L1M1N1" -f4 -v
```

The last example gives exactly half 20S proteasome. However, it took extremely long to execute (~15min). Thus, if one wants to obtain a complex with a great deal of unique chains, it is a wise idea to let the programme run and go have a coffee ;).

GUI - UNDER DEVELOPMENT

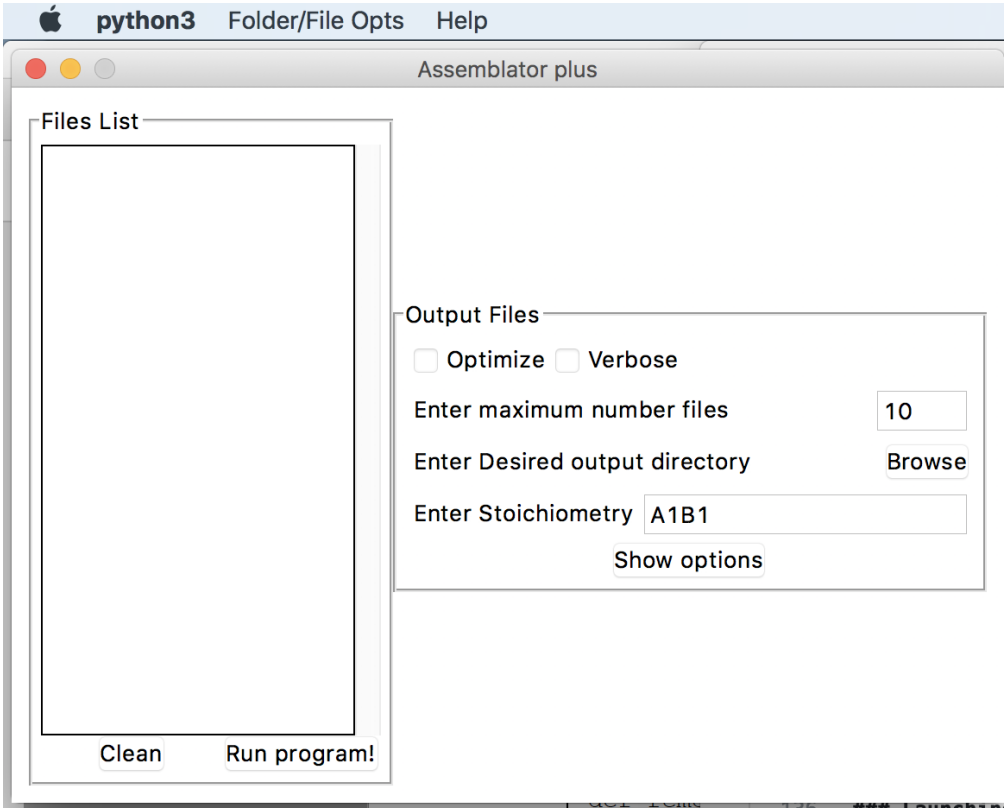
We are currently developing a GUI interface to run the program without command line. Here I will show the work done and the next steps.

Launching the app

To do so, you have to run the following command on the terminal:

```
python3 gui2.py
```

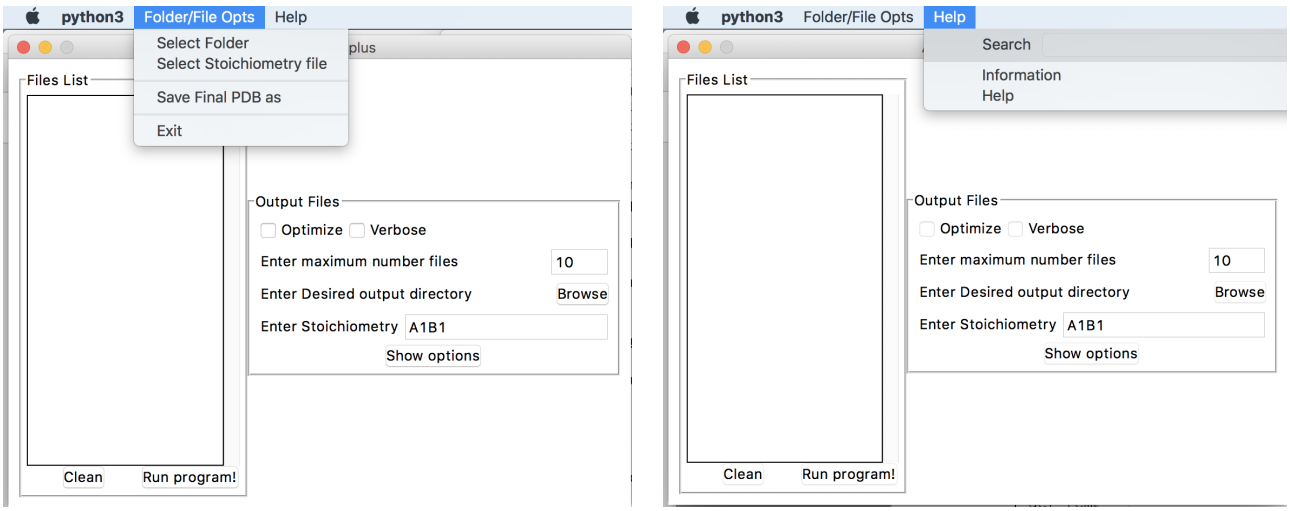
The next window will appear on your computer.



We can see a the window divided into two frames. On the left frame we will have a list with the files on the specified directory. Then there is a Clean button to restore all the options on the right and the Run program! button to launch the request. If we take a look on the right frame we can see the different options the user can specify. We can chech the optimize en verbose checkboxes, enter the maximum desired files, browse to choose the output directory and finally an Entry to specify the stoichiometry of the desired macrocomplex.

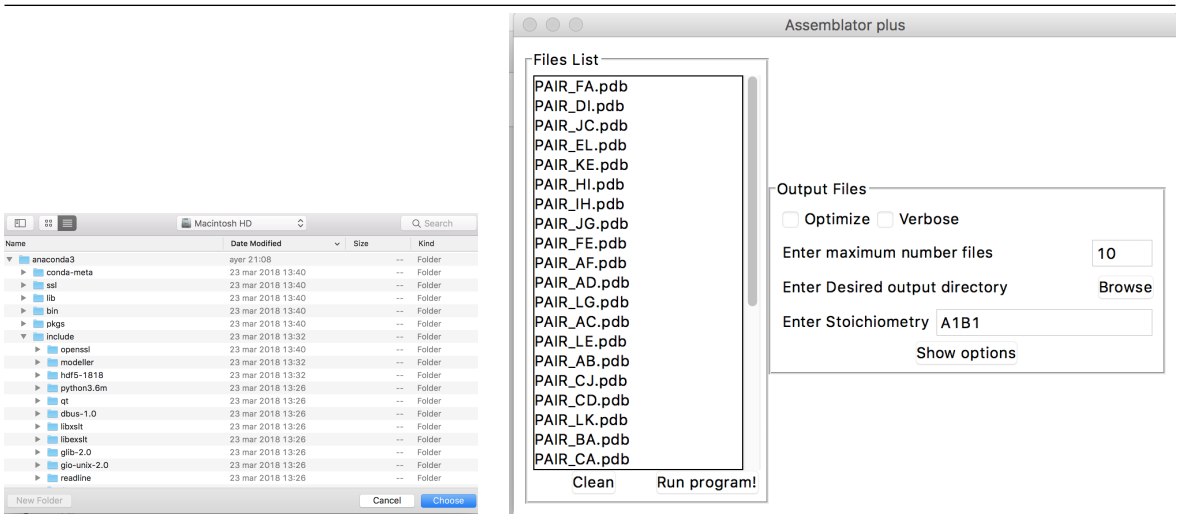
Finally looking at the upper part of the app there is the menu, with two main parts: “Folder/File Opts” and “Help”.

Folder/File Opts | *Help Menu*

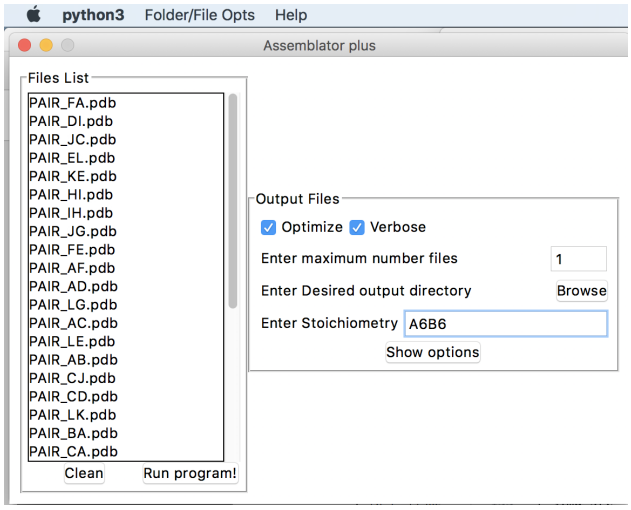
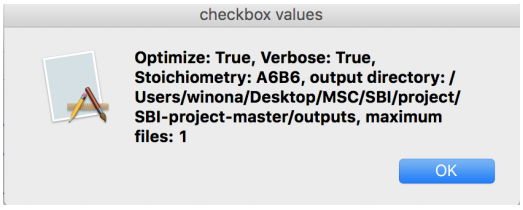


If we click on the option Select Folder from the Folder/File Opts, a window will appear for selecting the desired folder with the PDB input files. Once specified, on the left frame we will see a list of those files.

Selecting the input folder | *List with input files*



Then sepecifiying the desired options, we can click on the "Show options" button to see a summary of the different options.

Specified options	*Messagebox*
	

Next steps

On the following weeks we intend to finish the GUI interface. For the moment when clicking on the "Run program!" button the program seems to run, but gives a weird error message from the Biopython module.

```
/anaconda3/lib/python3.6/site-packages/Bio/PDB/Atom.py:125: RuntimeWarning: invalid value
encountered in sqrt
  return numpy.sqrt(numpy.dot(diff, diff))
```

We're trying to fix the problem and have the application working as soon as possible!

Requirements

In order to run this program with all its functionalities the user must have several packages dowloaded and working:

- Python 3.6
- Modules:
  - Modeller v.9.19
  - Tkinter (for the GUI interface)
  - Biopython
  - argparse
  - os
  - sys
  - numpy

Limitations

- Unable to model protein-RNA or protein-DNA interactions.
- Scalability: after certain recursion depth the program does not scalate too well. To improve the program, we could have used a dynammic programming approach, where out of several complexes only the most optimal is kept, and the next solutions are built onto that one.

- Stoichiometry: as powerful as it may be, the application relies too heavily on the stoichiometry input. On the one hand, this gives a greater control to the user over the output he or she expects but, on the other hand, it does not find complexes blindly, regardless of the stoichiometry.
- Without graphical interface for the moment.