MacroPy

by Alejandra Omaira Morcillo Nieto, Nicolás Díaz Roussel and Francho Nerín Fonz

TABLE OF CONTENTS

INTRODUCTION **1**

OBJECTIVE **2**

MacroPy **2**

THEORETICAL FRAMEWORK **3**

Consider two chains to be the same 3

Create ModelChains starting from the longest chains 3

First chain of the complex 4

Check clashes 4

Filling of missing atoms and minimization by Modeller 5

TUTORIAL **6**

Requirements 6

Installation 6

Program usage 6

EXAMPLES AND ANALYSIS **8**

Hemoglobin: PDB 1gzx 8

NFkB, IRF7, IRF3, AP-1 and ATF-2 bound to the IFN-β enhancer: (PDB 2o61) 9

RNA polymerase III: PDB 5fj8 10

Microtubule: PDB 5syg 11

Enterovirus 71 empty capsid: PDB 3j23 12

Phosphate dehydratase: PDB 2f1d 13

LIMITATIONS AND FUTURE PERSPECTIVES **14**

Construct many possible complexes 14

Whole sequences to model missing residues 14

REFERENCES **15**

1. INTRODUCTION

Nowadays, we know that protein–protein interactions (PPIs) are of great importance for all living organisms. Both the underlying association of proteins into functional protein complexes and their dissociation is a very dynamic process which is regulated by different cellular mechanisms [1]. Altogether, PPIs form a huge and complex network known as "interactome", which greatly contributes to the regulation and execution of the majority of biological processes [2], including cell-to-cell interactions and metabolic and developmental control [3]. Despite its key role in the functionality of metabolism, the interactome is a field in which only 2-11% of the estimated interactions that occur in an organism are known [4].

For this reason, there is a need to understand how proteins interact in order to have a better understanding of the underlying signal transduction pathways and, thus, biological processes. Besides, the function of unknown proteins could be identified from studying the interaction of these unknown proteins with known protein targets with known function [5].

There are different experimental approaches to elucidate PPIs, such as mass-spectrometry, co-immunoprecipitation, the two-hybrid system or affinity chromatography. However, these methods have limitations in terms of cost and time, and the resulting datasets are noisy, with many false positives. Therefore, computational methods are being developed, such as sequence-based or structure-based approaches, with which it is possible to reduce the set of potential interactions to a subset of most likely interactions [6].

Once the PPIs have been identified, it is necessary to know the three-dimensional (3D) complex structures, which are essential for a complete understanding of the interaction mechanism. There are two main approaches for the computational modelling of PPIs: template-based modelling and *ab initio* modelling. Template-based methods use known structures of the complexes between homologues of the target proteins. In contrast, *ab initio* methods use statistical or thermodynamic energy information to find the best predicted disposition of the target complex structure [7].

It is in this last stage of elucidating the complex structure, that our programme can be situated. The main scope of this project is to reconstruct biological complexes starting from individual pairwise interactions, using computational tools.

2. OBJECTIVE

The main objective of the program is to reconstruct a whole biological macro-complex using only pairwise interactions as input, either protein-protein, protein-DNA or protein-RNA.

3. MacroPy

Our program can be divided in two main steps. The first one is the preprocessing of the pdb structures and then the construction of the protein complex is performed.

First of all, our program only works with PDB input files whose name structure is either <structure_name>_<chain1>_<chain2>.pdb(.gz) or <Uniprot ID>.<DNA or RNA>.<PDB name>_<chain1>_<chain2 (and chain3)>.pdb(.gz). Besides, each chain ID in the name of the file has to be stored also inside the structure file with the same ID. A stoichiometry file can also be passed as input, and a complex will be returned even if the way in which our program constructs the complex isn't able to meet the stoichiometry requirements.

For the preprocessing step, we have created a new class called *ModelChain*. Each object of this class is defined by the first chain that it is added to it, its sequence and an identification code (from A to ZZ), and it stores all the chains among the different PDB files that are considered the same as this first chain of the *ModelChain* (similar sequence and structure, as defined in Theoretical framework, Consider two chains to be the same) as well as their corresponding interactions with other chains, to use in the reconstruction. We start assigning the chains to *ModelChains* from the longest to the shortest chains (see Theoretical framework, Create ModelChains starting from the longest chains) and we consider that two chains of the same PDB file are interacting if they have at least one pair of atoms that are at a distance of under 5 Å [8].

Then, the reconstruction starts by choosing a starting chain, and it performs cycles of adding the interactions of the chains that are added to the complex in each cycle (starting from this first chain), iteratively. The interactions of the chains are added by superimposing the chain that is present in the complex with the chain of the *ModelChain* whose interaction we want to add, and then we use the same rotation-translation matrix to transform the coordinates of the interacting chain and attempt to add it to the complex. The interacting chain is added if in its new coordinates it does not clash with the existing chains (see Theoretical framework, Check clashes), and interactions are added until the stoichiometry or the maximum number of chains is reached, or until the program isn't able to add more chains. The first chain is chosen from the *ModelChain* that gathers the highest number of different PDB IDs (that contains chains that come from PDB files with different PDB IDs) or, if any of the chains is a nucleic acid, from the *ModelChain* of the longest nucleic acid molecule (see Theoretical framework, First chain of the complex).

The program outputs a MMCIF file to the structures folder, and there are optional arguments to also save a PDB file (with up to 25 chains of the complex) and/or to perform an energy minimization in vacuum (after filling any missing atoms) by a Conjugate Gradients algorithm (needs Modeller [9]), until convergence or up to a desired number of steps, outputting the minimized structure of the complex and also energy information of the minimization process to the analysis folder (see Theoretical framework, Filling of missing atoms and minimization by Modeller).

4. THEORETICAL FRAMEWORK

4.1. Consider two chains to be the same

In each *ModelChain* object, we group chains from different PDB files that we consider to be the same/similar/equivalent. We consider two chains to be similar (e.g., the chain that represents a *ModelChain* and a chain we are preprocessing to decide if it goes inside this *ModelChain* or not) if the alignment "score" of their sequences is over 95% and the RMSD of their superimposition is below 2.5 Å.

We extract the sequences of the chains from their residue (or nucleotide) list, so we don't take into account missing positions that don't have any ATOM or HETATM entry. Then, we use the *PairwiseAligner* class of the Biopython module *Align* and perform a local alignment that scores positively gaps but penalizes mismatches and gaps (and gap extensions) equally. This way, we will always retrieve alignments that try to find the longest continuous alignment between the two sequences (e.g., the alignment of ATGG inside the longer sequence TTTTTATGGTTTTT). Then, the score provided by *PairwiseAligner* for these "good" alignments is in most cases equal to the number of positions aligned (mismatches at either end aren't penalized, as it is a local alignment). Finally, to define our particular alignment "score" we differentiate whether the alignment is between nucleotides or proteins. In the first case, we are interested in finding shorter sequences inside longer ones, so we divide this number of positions aligned between the length of the shortest chain in the alignment. For proteins, we want that the "score" is below the threshold when they don't have similar length and sequence, so we divide by the length of the longest chain. This alignment "score" is between 0 and 1 and we set the threshold to consider that two chains have a similar enough sequence at 0.95, which we have observed avoids both false positives and false negatives, for the most part, and that can be modified by the user ("Identity threshold" parameter, *-it*).

After the sequence comparison, if the "score" is above the threshold, we compare the structures of the chains by superimposition and we consider them to be essentially the same if the resulting RMSD is below 2.5 Å, which is also a reasonable value for our purpose that we have set empirically and that can also be specified by the user (*-Rt* parameter). We use the *Superimposer* class from the Biopython module *PDB* and we only superimpose the aligned positions returned by the previous alignment that meet the "score" requirements.

On this note, during the construction, when it's time to add the interactors of a chain (a ModelChain in our program, which has a list of interactions) of the complex in construction, we add the interactor at every position in which the chain that interacts with it in the original PDB aligns with the chain that is present in the complex in construction (with an alignment "score" above the threshold). This is crucial to construct protein-DNA complexes, in which a protein can interact with a short nucleotide chain with a certain sequence (e.g., ATGG) and the nucleotide chain that is present in the complex in construction may contain this sequence more than once (e.g., TTTTTATGGTTTTTATGGTTTTT), so this way the protein will be added at every position.

4.2. Create ModelChains starting from the longest chains

After we finish reading the PDB files, we start creating the *ModelChains* from the chain with the longest sequence. This is related to the construction of protein-DNA complexes, since we enforce that the longest nucleotide sequences, which will probably contain the sequences of

shorter nucleotide chains inside them, will be the ones to represent their *ModelChains*. Thus, they will be processed before shorter chains. Without this order in the processing of the chains, it may happen that when the longer nucleotide chains are processed, they align with short nucleotide chains of the existing *ModelChains*, they qualify to be considered the same sequences and are put inside these *ModelChains*. In this case, it is likely that there will be other short chains of different *ModelChains* that are contained in these longer nucleotide chains too, but that will end up not having these longer nucleotide chains inside, and therefore the program won't be able to detect the relationship between them and we will lose this information.

4.3. First chain of the complex

By choosing the first chain of the complex to be the one of the *ModelChain* that gathers the highest number of different PDB IDs (i.e. that contains chains that come from the highest number of different PDB IDs from the different files), we try to ensure that the highest amount of information is used for the construction of the complex, as it will probably contain a higher number of different interactors and types of interaction, and thus will integrate more information about the different input files.

4.4. Check clashes

While constructing the complex, before adding a new interacting chain we make sure that it doesn't clash with the existing chains of the complex in construction. For that, we use the *NeighborSearch* class of the Biopython module PDB, to check that the coordinates of the atoms of the existing chains aren't too close to atoms of the chain we are trying to add. To speed up the process, first we select only the chains that are closest to where the new interactor is trying to be added, and we do this by performing a search using only the α-carbons from the proteins and the backbone phosphates of the nucleic acid chains. We consider that a chain is close when the distance between one of its atoms and any atom of the interactor is less than 5 Å (α-carbons or phosphates), which is a value that we have also set empirically and that can't be changed, as it is just used for selecting the chains that are closer which will later be more exhaustively scanned for clashes.

Then, using all the atoms from the closest chains (and all the atoms of the interactor), we perform another search and this time we decide that two atoms of different chains are clashing if the distance between them is less than 1.8 Å [10], which is a parameter that can be set by the user (*-cd*), and that the interactor is definitely clashing and shouldn't be added if the number of clashes is above 20 (to allow a limited number of clashes, also a parameter (*-nc*)).

We also use NeighborSearch for a different purpose than checking clashes. We use it to make sure that the chains present in the same PDB file are indeed interacting when processing them. For this, we decide that two chains are interacting if at least one pair of atoms is found at a distance below 5 Å [8].

4.5. Filling of missing atoms and minimization by Modeller

If the *-min* argument is passed, Modeller [9] is used to fill the missing atoms of the complex and performs an energy minimization in vacuum by Conjugate Gradients algorithm, until

reaching convergence or up the maximum number of steps specified after the argument (*-min*) (the script is taken from the Modeller manual, with some modifications). The filling of missing atoms is performed by the *complete_pdb* function, and it is required to run the energy minimization, in which broken molecules or incomplete residues would provoke aberrant behaviour. If a protein has missing loops, given that we don't have the information of the complete sequence, the function joins the two ends of the missing loop so that the protein becomes a single molecule without any broken bond. Then, the minimization is run using the *ConjugateGradients* class of the Modeller module *optimizers*, either until the maximum number of steps is reached or until convergence. Convergence is said to be reached when the maximal atomic shift is less than 0.01, meaning that none of the atoms are moving considerably anymore. The progress log of the minimization process, with information such as the decrease in energy of the system or the value of the energy gradient, is printed out every 10 steps to a file in the analysis folder.

5. TUTORIAL

This section explains how to execute the program by running a script from the command line. Both required and optional arguments will be explained.

5.1. Requirements

In order to execute the program correctly you need Python version 3.7 or higher and the package Biopython. Modeller is necessary to run the *-min* option. Biopython can be installed with the following command:

```
pip install biopython
```

5.2. Installation

The program can be installed using the uncompressed .tar file or after cloning the GitHub repository. It can be installed as a package by executing the following command:

```
python setup.py install
```

5.3. Program usage

To run the program you have to execute the following Python script with the appropriate arguments:

```
usage: Macro.py [-h] -i INPUT_DIRECTORY -o OUTPUT_DIRECTORY [-c COMPLEX_NAME]
              [-f] [-s STOICHIOMETRY] [-v] [-min [MINIMIZATION]] [-pdb]
              [-mc MAX_CHAINS] [-it IDENTITY_THRESHOLD] [-Rt RMSD_THRESHOLD]
              [-ns NEIGHBOR_SEARCH_DISTANCE] [-cd CLASHES_DISTANCE]
              [-nc NUMBER_CLASHES]

MacroPy 1.0 - Reconstruct a whole biological macro-complex using PDB's of its
pairwise interactions as input, either protein-protein, protein-DNA or
protein-RNA.

required arguments:
  -i INPUT_DIRECTORY, --input-directory INPUT_DIRECTORY
                          Directory containing the input structure files
                          (default: None)
  -o OUTPUT_DIRECTORY, --output-directory OUTPUT_DIRECTORY
                          Create the output directories (default: None)

optional arguments:
  -c COMPLEX_NAME, --complex-name COMPLEX_NAME
                          Reconstructed complex name (default: Complex)
  -f, --force             Force overwriting if the output directory already
                          exists (default: False)
  -s STOICHIOMETRY, --stoichiometry STOICHIOMETRY
                          File containing the stoichiometry (default: None)
  -v, --verbose           Program log will be printed to standard error while
                          running (default: False)
  -min [MINIMIZATION], --minimization [MINIMIZATION]
                          Perform an energy minimization by Conjugate
                          Gradients algorithm with the specified number of
                          steps or until convergence (default: False)
  -pdb, --save-pdb        Besides the .cif file, save a .pdb file with up to
```

```
                          25 chains (default: False)
  -mc MAX_CHAINS, --max-chains MAX_CHAINS
                          Number of chains of the complex at which to stop
                          adding new chains (default: 180)
  -it IDENTITY_THRESHOLD, --identity-threshold IDENTITY_THRESHOLD
                          Minimum percentage of sequence similarity (between 0
                          and 1) to consider two PDB chains the same (default:
                          0.95)
  -Rt RMSD_THRESHOLD, --RMSD-threshold RMSD_THRESHOLD
                          Maximum RMSD value to consider two (similar) PDB
                          chains the same (default: 2.5)
            -ns      NEIGHBOR_SEARCH_DISTANCE,      --Neighbor-Search-distance
NEIGHBOR_SEARCH_DISTANCE
                          Minimum distance between two PDB chains to consider
                          that they are actually interacting (default: 5)
  -cd CLASHES_DISTANCE, --clashes-distance CLASHES_DISTANCE
                          Maximum distance between atoms of two chains to
                          consider that they have clashes between them
                          (default: 1.8)
  -nc NUMBER_CLASHES, --number-clashes NUMBER_CLASHES
                          Maximum number of close atoms to consider that two
                          chains are clashing (default: 20)
```

6. EXAMPLES AND ANALYSIS

In this section, we present some examples of the reconstructed complexes and the arguments used to obtain them, in order to get a better understanding of how to run MacroPy.

6.1. Hemoglobin: PDB 1gzx

This hemoglobin is a complex formed by 4 different chains (A, B, C, D) and to reconstruct it we use 3 pairwise interactions, being chain A the link to the rest of the other chains. MacroPy is able to reconstruct this complex using the default argument values. Here we show the content of the verbose mode.

```
$ python3 Macro/Macro.py -i examples/1gzx/ -o output_1gzx -v
```

```
Running MacroPy
Parsing input directory and input PDB files
Removing files from existing output directory
Processing chains
ModelChain A contains the chains: 1gzx_A_D:D, 1gzx_A_B:B
ModelChain B contains the chains: 1gzx_A_D:A, 1gzx_A_C:C, 1gzx_A_C:A,
1gzx_A_B:A
Building new complex
First chain in the complex: 1gzx_A_D:D
Chains in the complex: 1
Checking the 2 interactions of chain 1gzx_A_D:D
Added chain 1gzx_A_D:A
Added chain 1gzx_A_B:A
Chains in the complex: 3
Checking the 4 interactions of chain 1gzx_A_D:A
Added chain 1gzx_A_B:B
Checking the 4 interactions of chain 1gzx_A_B:A
Chains in the complex: 4
Checking the 2 interactions of chain 1gzx_A_B:B
The final complex has 4 chains
Complex saved as Complex.cif
Finished!
```

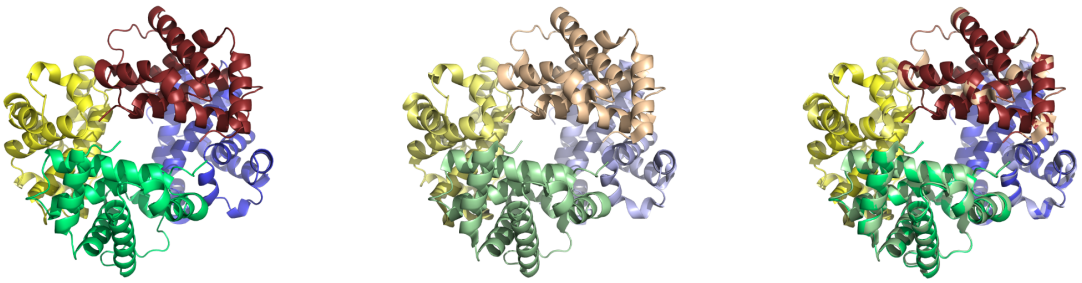

Fig 1. *Left: complex constructed by MacroPy. Center: PDB 1gzx. Right: superimposition.*

We can easily see that the result of our program is highly similar to the original one. We can see the 4 chains and how chain A (represented by red color) interacts with the other chains (B blue color, C green color and D yellow color).

6.2. NFkB, IRF7, IRF3, AP-1 and ATF-2 bound to the IFN-β enhancer: (PDB 2o61)

This structure is an example of a protein-DNA macro-complex, since it is a transcriptional activation complex where 5 proteins are involved. The PDB 2o61 contains 3 of these proteins (except the AP-1 and ATF-2 DNA binding motifs) also bound to the IFN-β enhancer, but we use it nonetheless to compare it with our constructed complex. Among the input files, a long DNA double strand is included, which corresponds to a larger portion of the IFN-β enhancer than the one present in PDB 2o61. This long DNA is used to reconstruct the transcriptional complex around it. The original PDB 2o61 has 2 protein chains, one that corresponds to the NFkB and a fusion protein with another NFkB, the IRF3 and the IRF7, all fused together. The stoichiometry file that accompanies the input files requires that the complex contains (besides the DNA double chain) two NFkB (P19838, Q04206), one ATF-2 (P15336) and one AP-1 (P05412) -which are not present in PDB 2o61-, and four IRF3 (Q14653), without the IRF7 protein.

```
$ python3 Macro/Macro.py -i examples/2O61/ -o output_2O61 -s examples/stoichiometry_files/2o61_stoichiometry.txt
```

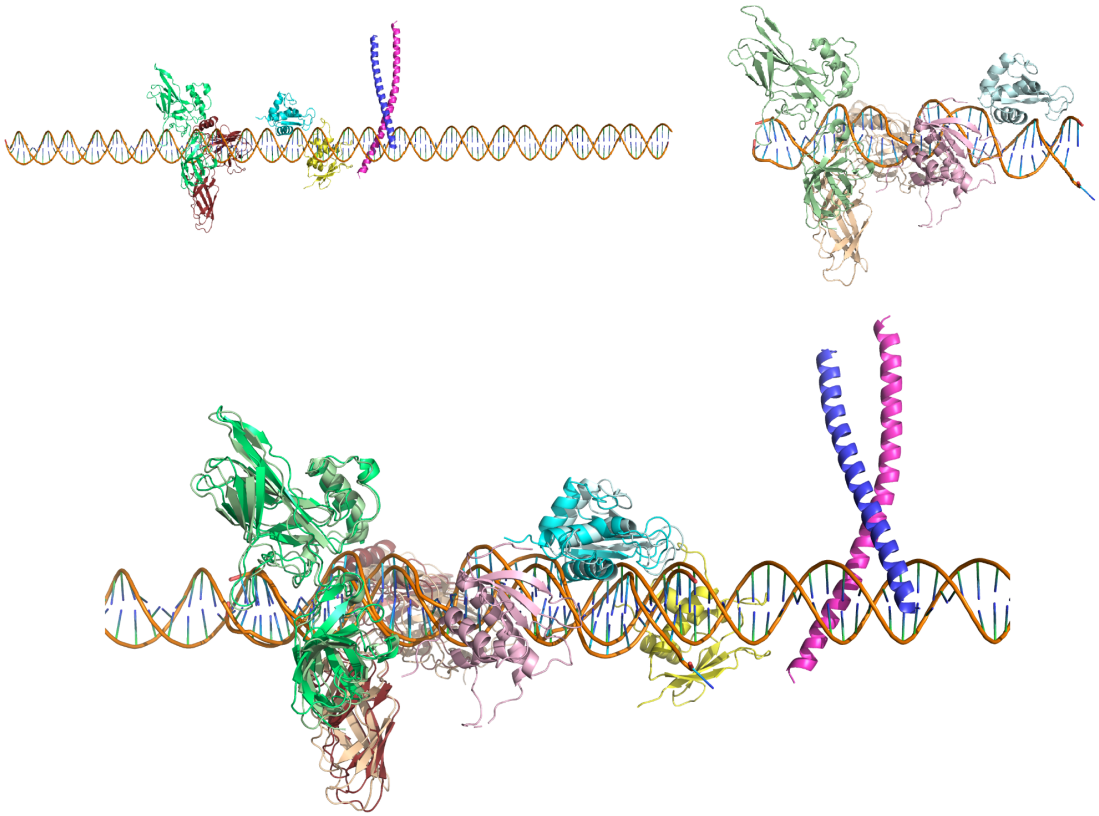

Fig 2. *Top left: complex constructed by MacroPy. Top right: PDB 2o61. Bottom: superimposition.*

In the way that our program constructs the complex, it isn't able to enforce the desired stoichiometry, so it prints the message "*Stoichiometry couldn't be satisfied*" but still saves the complex it achieved. Still, it is still able to use eight out of the ten desired protein chains: it places the two proteins that weren't in PDB 2o61 (AP-1 and ATF-2, magenta and blue) together

with the two NFkB and two IRF3 (the stoichiometry requirement was four IRF3). The NFkB proteins of the constructed complex superimpose with the ones in the PDB (green and red) and so does one of the IRF3 proteins (cyan), while the second one is in different places in the constructed complex and in the PDB (pink and yellow).

6.3. RNA polymerase III: PDB 5fj8

In this case, we also have an example of protein-nucleic acid interaction. This example is a heterotrimer composed of 17 proteic chains, 2 DNA chains (R and S chains) and RNA 1 (T chain). In total, there are 44 pairwise interactions files. We introduce the *-min* optional argument to perform an energy minimization.

```
$ python3 Macro/Macro.py -i examples/5fj8/ -o output_5fj8 -min
```

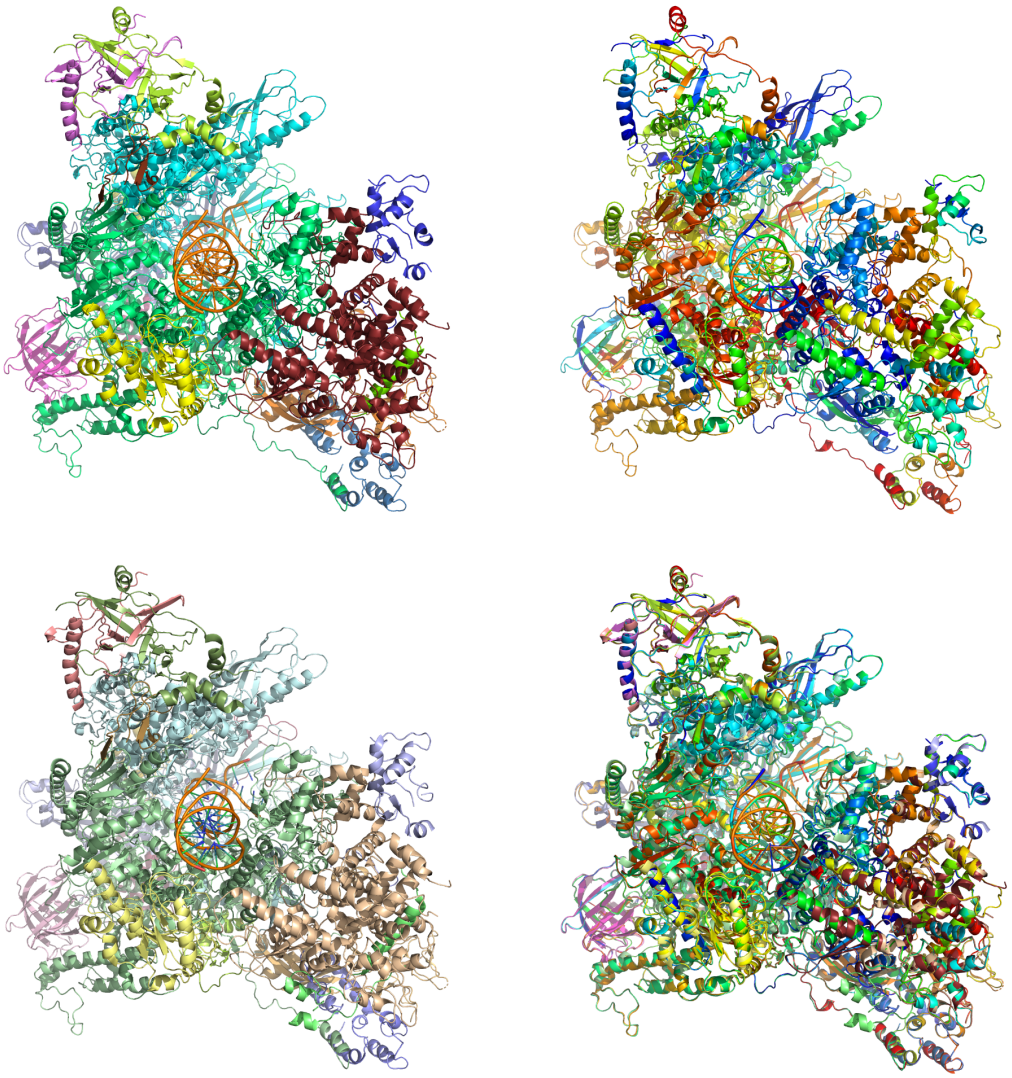

Fig 3. *Top left: complex constructed by MacroPy. Top right: complex generated using the energy minimization option. Bottom left: PDB 5fj8. Bottom right: superimposition.*

The complex is reconstructed perfectly and it shows that our program is able to handle single- and double-stranded DNA or RNA without any problem. The *-min* argument completes the protein (it doesn't add the missing residues, it just joins the ends of the missing loops and adds other missing atoms; see Theoretical framework, Filling of missing atoms and minimization by Modeller) and provides a minimized complex structure (Fig 3. Top right) and a log file of the progress of the minimization in the analysis folder:

```
minimization.log
# Conjugate gradients optimization
#  Step   Current energy Av shift Mx shift   Funcs         Gradient
      0    1330377.62500   0.0000   0.0000       1      3.80219e+09
     10      93601.94531   0.0049   1.0242      21        4.283e+07
     20      47927.33203   0.0037   0.5264      41      6.71732e+06
     30      30341.32031   0.0009   0.2364      61           932458
     40      22964.38672   0.0064   0.4893      81           984711
     50      19933.69727   0.0040   0.1790     101           393292
     60      18302.95312   0.0049   0.1703     121           234262
     70      17443.15234   0.0049   0.1213     141           169717
     80      16834.43164   0.0043   0.1224     161           105096
     90      16366.63086   0.0046   0.1170     181          82920.5
    100      16018.48047   0.0046   0.0708     201            60592
    110      15797.16992   0.0005   0.0289     221          28241.1
    120      15662.50391   0.0029   0.0832     241          37775.6
    130      15462.84473   0.0038   0.0829     261          46948.1
    140      15257.76660   0.0028   0.0427     281          53195.1
    150      15057.54980   0.0038   0.0554     301            39092
    160      14882.40234   0.0049   0.0626     321          29657.2
    170      14739.81543   0.0029   0.0377     341          43890.1
    180      14605.88574   0.0023   0.0298     361          25594.6
    190      14472.01758   0.0035   0.0499     381          25872.9
    200      14348.09277   0.0034   0.0508     401          22828.9
    210      14262.39648   0.0012   0.0193     421            17842
    220      14190.57520   0.0008   0.0172     441          13946.7
    230      14121.75977   0.0031   0.0462     461            20811
    240      14050.03711   0.0020   0.0352     481          21068.6
    250      13982.67188   0.0009   0.0148     502          9186.72
    251      13978.98145   0.0007   0.0098     504          7924.98
```

6.4. Microtubule: PDB 5syg

The structure of the microtubule is formed by dimers of α-tubulin and β-tubulin, which are present in the 4 input files used of pairwise interactions. This complex has the special characteristic of being made up of repeats of these dimers, so that we can establish whatever number of repetitions we want using the *-mc* (max. number of chains) argument (the default is 180 chains). We also show the use of the *-f* argument to overwrite the contents of an existing output folder.

```
$ python3 Macro/Macro.py -i examples/5syg/ -o output_5syg
```

Increasing the maximum number of chains and forcing the overwriting of the output folder:

```
$ python3 Macro/Macro.py -i examples/5syg/ -o output_5syg -f -mc 300
```

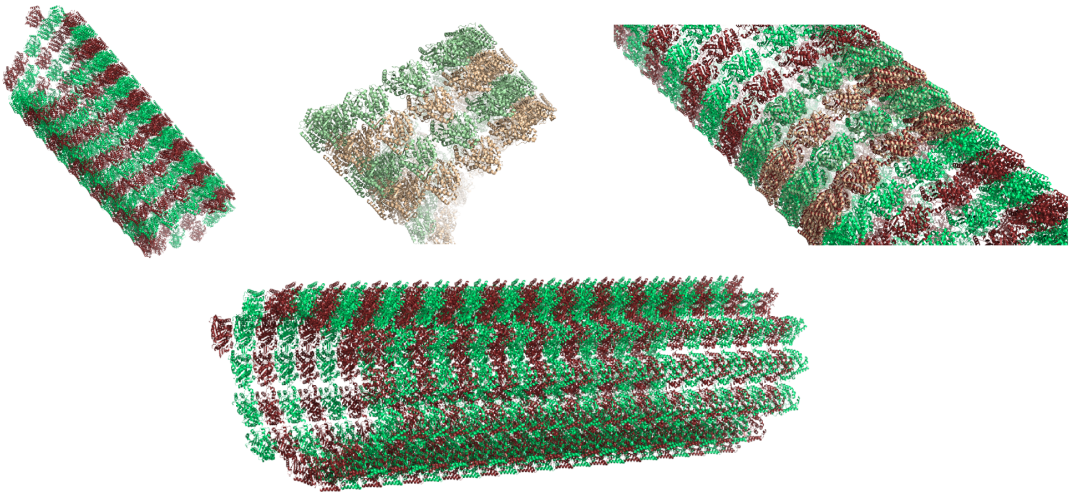

Fig 4. *Top left: complex constructed by MacroPy. Top center: PDB 5syg. Top right: superimposition. Bottom: complex constructed using 300 chains.*

With the default arguments, a fragment of the microtubule is successfully constructed. Then, increasing the maximum number of chains of the complex to 300, a longer fragment is returned. Of course, this takes more time to construct: the algorithmic complexity of our program is potentially quadratic, as we go over each chain and for each chain we analyse all its interactions, which could be as many as the total number of chains.

6.5. Enterovirus 71 empty capsid: PDB 3j23

The capsid of EV71 is made up of 3 proteins, VP1, VP0 and VP3, which form a heterotrimer that is repeated 60 times. With our input files, the whole capsid is constructed using 6 pairwise interactions. Using this example, we want to compare the *-min* optional argument with and without specifying a maximum number of minimization steps. Using *-min*, the minimization is carried out until convergence (see Theoretical framework, Filling of missing atoms and minimization by Modeller), but we can also specify a maximum number of steps (*-min X*).

```
$ python3 Macro/Macro.py -i examples/3j23/ -o output_3j23 -min
```

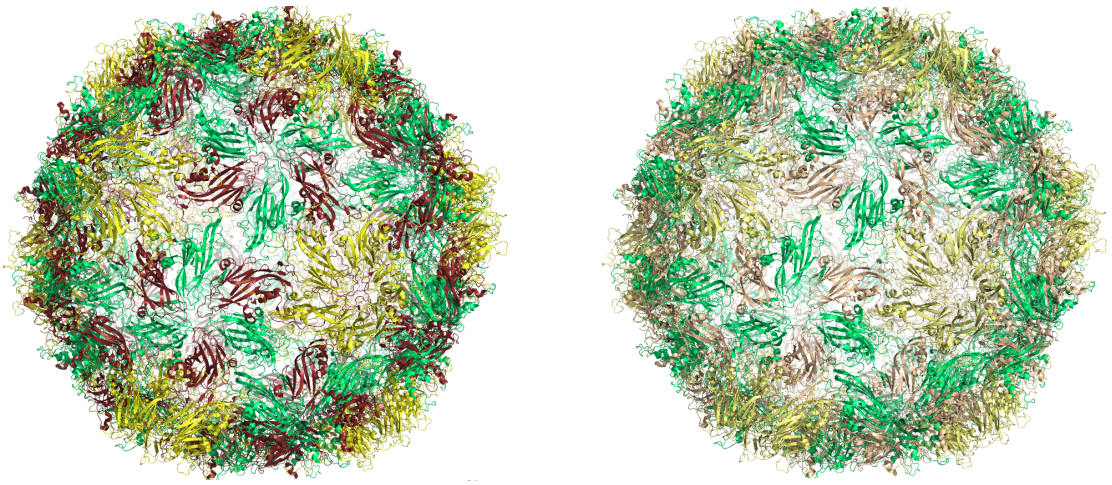

Fig 5. *Left: complex constructed by MacroPy. Right: PDB 3j23.*

Decreasing the maximum number of steps of the energy minimization to around half of the steps that have been necessary for convergence (50) and overwriting the output folder:

```
$ python3 Macro/Macro.py -i examples/3j23/ -o output_3j23 -f -min 50
```

We conclude that, independently of the number of steps, the algorithm takes roughly the same time when we use the *-min* argument than when we limit the maximum number of steps with *-min X*, but that the complex is formed and optimized correctly anyway.

6.6. Phosphate dehydratase: PDB 2f1d

The structure of phosphate dehydratase is composed of 24 identical proteins, arising from the *in vivo* assembly of 8 inactive homotrimers. With this last example, we want to compare the default output in MMCIF format and in PDB format, obtained using the *-pdb* argument. In addition, we also want to demonstrate the capabilities of the stoichiometry implementation, as the program will be able to reconstruct the complex when we specify a stoichiometry of A:24 (the 24 proteins of the complex are the same), even when the chain IDs of the pairwise interactions input files are A, B, C, D, H and F (the program recognizes they are the same and puts them in a single ModelChain) and there are only 4 input files.

```
$ python3 Macro/Macro.py -i examples/2f1d/ -o output_2f1d -s examples/stoichiometry_files/2f1d_stoichiometry.txt -pdb
```

As we expected, there is not any difference between the complex in MMCIF format and in PDB format, since this complex only has 24 chains and the *-pdb* argument outputs a PDB file with up to 25 chains.

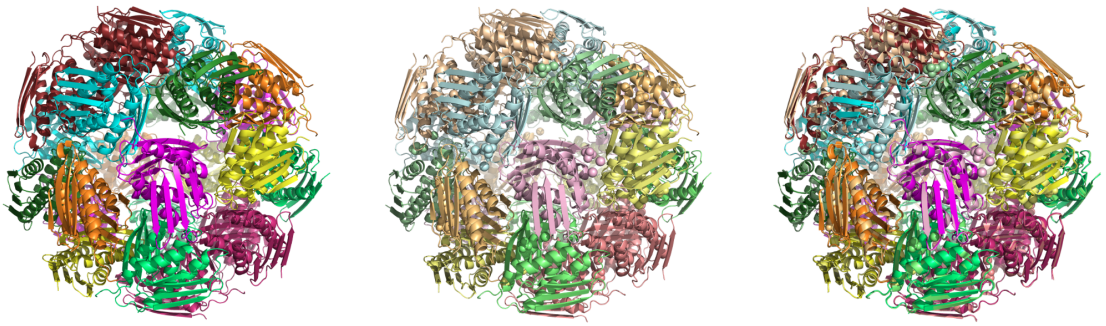

Fig 6. *Left: complex constructed by MacroPy. Center: PDB 2f1d. Right: superimposition.*

Apart from being able to process the chains and the stoichiometry to reconstruct the complex successfully, this example also shows that our program is able to handle “heteroatoms” and the output complexes also contain these molecules and ions.

7. LIMITATIONS AND FUTURE PERSPECTIVES

7.1. Construct many possible complexes

At present, our program only provides one possible complex as output. We could introduce heuristics or randomness at various steps of our approach, so that many possible complexes could be built and then ranked to be able choose the best one, for example using an energy-based analysis.

First, instead of starting to create the *ModelChain* objects using the chains with the longest sequences, we could do this process randomly, although this approach could lead to the problem we have stated earlier (see Theoretical framework, Create ModelChains starting from the longest chains).

After that, we could attempt the building of a different complex starting from each *ModelChain* (we consider the chains inside each *ModelChain* to be basically the same, so it probably wouldn't make a difference which the approach we used) instead of just picking one (see Theoretical framework, First chain of the complex), so that we could end up with many possible different models starting from these.

Moreover, when it is time to add the interactors of a chain, we could try to add them in all the possible orders they can be arranged into. For example, if two interactors are located in the same position of the surface of a chain, placing one or the other in the complex may make a difference in the rest of the construction, so the exploration of all these possibilities would most probably yield many different models too.

However, this would mean that we would be constructing many complexes at the same time, and thus increasing the computational cost of our program, as there would be many more clash checkings, alignments, superimpositions… to be performed. Still, we have already tried to speed up some calculations (see Theoretical framework, Check clashes).

7.2. Whole sequences to model missing residues

It would be more convenient to work with complete PDB files instead of files that only contain the lines about the atomic coordinates of the residues, since there are missing residues of the real protein sequence without these. This forces us to use sequences that we extract from the list of residues that do have atomic coordinates, and we can't retrieve the information about the missing residues this way. This is especially critical if we use the minimization argument, as Modeller tries to complete the missing positions, but without the information of the complete sequences it is not able to model these missing parts correctly (see Theoretical framework, Filling of missing atoms and minimization by Modeller).

For this reason, we propose that the PDB files used as input contain the appropriate entry with the whole sequence, as well as other possible strategies to retrieve these sequences.

For instance, with the sequence that we extract from the structures, we can perform a BLAST search (e.g., in the Uniprot database) in order to get the whole sequence that is the most similar to ours, and provide it to Modeller to model the missing parts adequately.

Alternatively, we could implement an additional argument to let the user specify a FASTA file containing the whole sequences of the proteins and provide them to Modeller.

8. REFERENCES

1. Nooren, Irene MA, and Janet M. Thornton. "Diversity of protein–protein interactions." *The EMBO journal* 22.14 (2003): 3486-3492.

2. Thiel, P., Kaiser, M., & Ottmann, C. (2012). Small-Molecule Stabilization of Protein–Protein Interactions: An Underestimated Concept in Drug Discovery?. *Angewandte Chemie International Edition*, *51*(9), 2012-2018.

3. Braun, P., & Gingras, A. C. (2012). History of protein–protein interactions: From egg-white to complex networks. *Proteomics*, *12*(10), 1478-1498.

4. Luck, K., Kim, D. K., Lambourne, L., Spirohn, K., Begg, B. E., Bian, W., ... & Calderwood, M. A. (2020). A reference map of the human binary protein interactome. *Nature*, *580*(7803), 402-408.

5. Browne, F., Zheng, H., Wang, H., & Azuaje, F. (2010). From experimental approaches to computational techniques: a review on the prediction of protein-protein interactions. *Advances in Artificial Intelligence (16877470)*.

6. Rao, V. S., Srinivas, K., Sujini, G. N., & Kumar, G. N. (2014). Protein-protein interaction detection: methods and analysis. *International journal of proteomics*, *2014*.

7. Vangaveti, S., Vreven, T., Zhang, Y., & Weng, Z. (2020). Integrating ab initio and template-based algorithms for protein–protein complex structure prediction. *Bioinformatics*, *36*(3), 751-757.

8. Halperin, I., Wolfson, H., & Nussinov, R. (2004). Protein-protein interactions: coupling of structurally conserved residues and of hot spots across interfaces. Implications for docking. *Structure*, *12*(6), 1027-1038.

9. Webb, B., & Sali, A. (2016). Comparative protein structure modeling using MODELLER. *Current protocols in bioinformatics, 54*(1), 5-6.

10. Baster, Z. (2013). The identification and the elimination of clashes in the structure of an early-stage intermediate in the protein folding process. *Bio-Algorithms and Med-Systems*, *9*(4), 199-207.