manual

Fabián Robledo

25 de marzo de 2020

Manual

Welcome to Promod's manual. Here you will find any information you need to run this software.

What is Promod?

Promod is python tool whose goal is to form macrocomplexes of molecules starting from the interacting pairs of chains that will form the complex. It is compatible with protein chains, single and double DNA strands and RNA strands. It has several parameters available to play with according to the users needs.

In essence, the builder is similar to a genomic assembler: seeks for similar chains in the different PDB files given as an input and overlaps them. After that, joins the chains in a single model. Pair by pair the builder puts all the possible chains inside a single model

Scientific Background

Introduction

Obtaining the tridimensional shape of a protein complex is a difficult and expensive process to obtain experimentally. Adding that there are thousands of different proteins in a single organism, with different splicings and mutations, which can vary even further in different spices, it's barely impossible to determine them all experimentally. Thus, finding computational ways to obtain the structure of a protein is advisable to know the structure of variations of a protein.

Here we present a computational tool to ensemble proteins and nucleic acids knowing their interactions. Currently, there are estimations that there are at least two hundred thousand different interaction between different chains of proteins that form every possible interaction in which a protein is present. Knowing those interactions, most of the complexes can be built easily knowing the stoichiometry of the protein.

Homology of the subunits

In order to build the model, we start with several files, each containing exclusively two chains (whether aminoacidical, DNA or RNA). By looking to similar proteins in two different pdbs, we can overlap them in the space to check if two chains of those pdbs are similar, and using this similarity to join them in the same model. This is known as protein superimposition, and the full process of joining is similar to a DNA sequence assembly: we have to recognise that two sequences are the same, and join them in the correct spot.

In order to consider two chains in diferent pdbs the same protein and overlap them, they must be homologous. To check whether two proteins or not are homologous, a pairwise alignment is made. This is a calculation of how much similar two sequences are, that takes both mutations and gaps into account to give a numeric score between 0 and 1. A score of 1 means that both sequences are identical, while high values indicate how similar

the sequences are. So, to consider them homologous, a high score must be achieved between two proteins. Only the homologous proteins will be overlaped and used to build the model

Superimposition of the 3D structure

Two homologous sequences will probably have similar structures. However, it's also possible that two proteins with lower homology score can have similar structures in space, as result of convergent evolution. Therefore, when two sequences are quite different but they may have similar structures, the may fullfill the same rol in our model and it's advisable to consider them: for example, two homologous proteins from distant species whose alignment doesn't pass our homologous threshold, but they still conserve the structure.

To consider them in our model, we need another measure to check if they really have similar structures, the root median square deviation (RMSD now onwars). To check this value, first we need to superimpose those two proteins; that's putting the atoms of both proteins in the same place with the same orientation to check how good or bad they overlape in space; similar structures will have atoms in nearly the same position, while different structures will differ. This similarity between the superimposed proteins can be used to calcule the RMSD, which is the average distance between the superimposed atoms. A value near zero indicates that both structures fits perfectly, and increases when the differences between the proteins structure start to grow.

This way, RMSD must also be a filter we must also filter those structures that are differents even if we consider them homologous.

Energy levels

Lastly, it's important to consider the final energy levels of the model. A good model of a complex should have minimum energy, as a functional, existing complex usually is, of every possible folding result of the complex, the one with minimum energy. Two atoms that are too close, hidrophobic residues in the external part of the model and several other considerations can increase the final energy of the complex, which must be corrected in a good model.

Two atoms too close can also indicate that two proteins are not correctly joined. For example, if several atoms of a chain might be to close of another chain (or even *inside*), those proteins should not be joined, even if they interact. Taking this into account avoids impossible protenis, such as those which are inside other chains but breaking several laws of thermodynamics and physics.

After that, it's possible that the position of the atoms inside the model may not be the most adequate. It's possible that little variations in some atoms' positions that conseve the protein structure

Input

The input is a folder with 2 or more interactions betweem 2 proteins, a protein and DNA/RNA strand or 2 single DNA strands (which may or may not form a double strand). Two structures are considered to be interacting where the minimum distance between them is less than 10 Armstrong. In lesser distances, forces between atoms are strong enough to produce changes between the chains and force them to adopt a different conformation, an thus, a realistic model must keep the residues at a real distance to consider it correct

Output

The output is mainly one pdb file with all the possible chains joined in the selected folder. However, if the optimized option was selected, several pdb files will be created in the same directory. These are different approaches made by modeller to optimize the energies of the model and the files it used. The model will be saved as final_model.pdb

Tutorial

Installing dependencies

In order to make Promod work, there are some dependencies that must be installed before executing the software: python3 and Biopython

If you're using Windows, you can download python3 from its website. If you got Linux, there's high chance that you already got it installed.

Independently of the operating sistem you got, you can install biopython using pip.

```
pip3 install biopython
```

Instalation

You can install Promod easily by downloading it from the github repository and running the next command in the downloaded folder. This is, for now, the recommended install way

```
pip3 install .
```

Alternatively, you can also run the setup script installation commands.

```
python3 setup.py install
```

Hands on: how to use promod

Promod has a command line interface which explains briefly how to use it before executing it.

```
python3 main.py -h
```

```
## usage: Build a macromolecular complex using interacting subcomponents
##        [-h] -i INPUT_FOLDER -o OUTPUT_FOLDER -f FASTA [-v] [-s STOICHIOMETRY]
##        [-d DISTANCE] [-t THRESHOLD] [-optimize] [-start START]
##
## optional arguments:
##   -h, --help            show this help message and exit
##   -i INPUT_FOLDER, --input-folder INPUT_FOLDER
##   -o OUTPUT_FOLDER, --output-folder OUTPUT_FOLDER
##   -f FASTA, --fasta FASTA
##   -v, --verbose
##   -s STOICHIOMETRY, --stoichiometry STOICHIOMETRY
##                         The stoichiometry of the final molecule the builder
##                         will try to forme
##   -d DISTANCE, --distance DISTANCE
##                         Maximun distance to consider that two atoms clash, in
##                         Armstrongs. By default, 0.1 armstrong
##   -t THRESHOLD, --threshold THRESHOLD
##                         Minimum score to consider two sequences homologus, and
##                         thus, the same to be build. Default 0.95
##   -optimize, --optimize
##                         Whether the model will be optimized with MODELLER
##                         after building or not. Default False
##   -start START, --start START
##                         Indicate the initial pdb from which the protein will
##                         be assembled
```

There are 3 mandatory commands while the rest are optional. The mandatory ones are related to the input files and output folder, necessary for the program to work, and they are *-i, the input folder; -o, the output folder; and -f, the fasta file of the sequences.* All of them will be covered in this manual.

Paramenters

Input folder The input folder should contain, at least, 2 pdb files. This folder may contain other files or subfolders; however, the program will ignore other files and will not check subfolders to find more pdb files. If you want to include some pdb, include it in the folder before running the program.

Beyond that last thing, no more things are necessary to know about the input folder. However, it's a mandatory argument, and should be indicated with -i or –input-folder tags

Output folder The output folder is the folder where the output files are written. This is mandatory, and no special folder is required. Just a warning, as, for the moment, the model file is written as 'final_model.pdb' and thus any other file with that filename will be overwritten.

The output folder is indicated with the -o or –output-folder tags

Fasta file The fasta file contains the sequences of the chains and the id for the stoichiometry. And it's a critical file, as the sequences must be homologous for the chain in the pdb. We can hold up to 5% of differences (by default, using -t parameter can modify this threshold). However, if one of the chains (if no stoichiometry indicated) or one of the chains in the indicated stoichiometry differs too much, the program will exit.

Note: That means the sequences in the fasta should be of a similar length than equivelent chain in the pdb. For example, the fasta file and the pdbs directly downloaded from Protein Data Bank may differ in the initial and ending residues, as they are quite difficult to model, and usually are removed from PDBs.

About the stoichiometry An additiona file with the stoichiometry can be given to the program to make sure that the final protein will contain no more than the indicated chains

Uninstalling

You can uninstall everything by using

```
pip3 unistall pro-mod-FC
```

Examples

Example 1 (3e0d)

3e0d is a small complex formed by 2 protein chains and 2 double DNA strands. It's a subunit of the eubacterial DNA polimerase but it's perfect for an initial testing of our project, so we can test how the program works, the time it takes and if the result is similar to the original one. In this cases, we have 6 different interactions, as each DNA strand is counted as a single one interacting with another one and binded to the protein chain by one of them. In fact, this complex can be thought as a dimer: two monomers formed by a protein and a double strand DNA, which interact by some residues in their protein chains.

So, the first test to our program consists on joining the different pdb files into one single structure, without further information about the stoichiometry. Therefore, the program will try to build the protein with only the information provided by the pdb files and the fasta file. The command to build it is the next one:

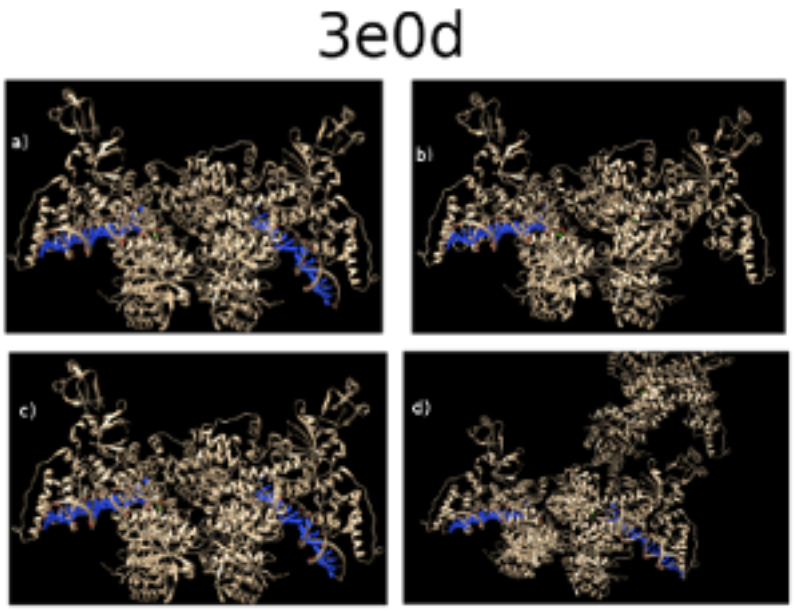

Figure 1: 3e0d structure obtained from: a) rcsb.com b) promod without any extra info c) promod with starting point and selected stoichiometry d) without stoichiometry control

```
python3 main.py -i examples/example_1/chains/pairs -o examples/example_1/results  \
        -f examples/example_1/3e0d.fa
```

We just need to give the folder with the input pdbs (-i), the desired output folder (-o) and the fasta file with the sequences of the proteins. The program will take the different pdbs on the folder (By alphabetical order, to make it reproducible) And the result it's incomplete:

There we can see that there is a missing double strand DNA. So, let's see how we can complete the model. We can see that there is a missing protein, let's force the program to include it. We have two different ways of indicating this: The easiest one is selecting the starting complex of our model (which contains those missing parts). This results that different starting points can result in different models, so, it's important to select a good one. By default, the pdbs filenames are sortered alphabetically and the first one is chosen as the starting point. In fact, this is an important choice: the differences between not chosing one (Figure 1.b) chosing an adequate one (Figure 1.c) or chosing another one (Figure 1.d) can result in models with extra chains or with less chains than expected

The other not-so-difficult way is to indicate the stoichiometry of the complex. In this case we wil focus in selecting the starting pdb, so the command would be.

```
python3 main.py -i examples/example_1/chains/pairs -o examples/example_1/results \
        -f examples/example_1/3e0d.fa -start examples/example_1/chains/pairs/3e0d_ZG.pdb
```

Yay! This is fare more similar to the original structure, very close to the original model. However, this is a very simple protein, and bigger complexes may are harder to build. However, from this example, we leart that:

- The program can build a model without any indication. However, it might be incomplete o have extra chains.
- The starting pdb is a important choice that can influence the final model. Thus it is important to remember which starting point gave which result. The same starting pdb will gave the same output.

Example 2 (6gmh)

6gmh is a very big complex formed by 24 different chains, with a double strand of DNA. There are several chains that interact between them, but there are not repeated sequences: each monomer is unique.

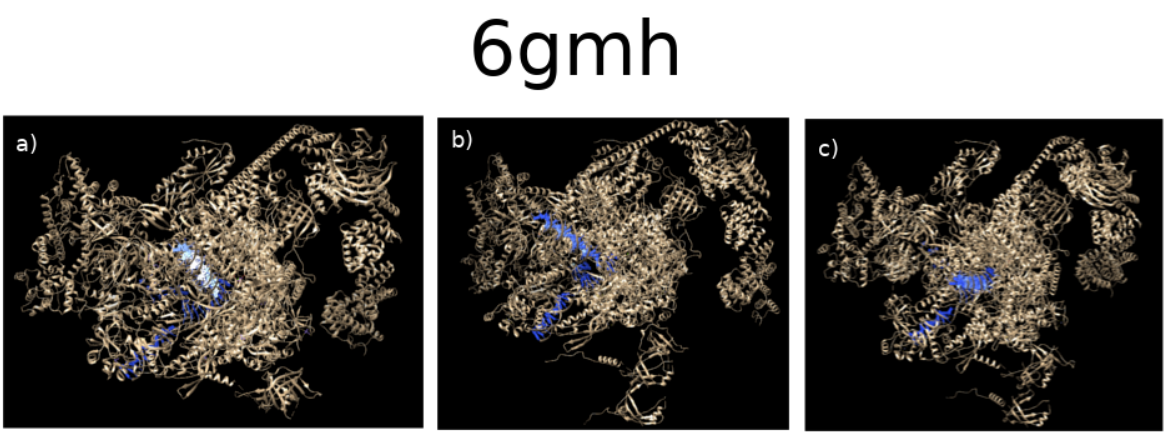

Figure 2: 6gmh structure obtained from a) rcsb.com b) builded without assistance c) selecting starting point and stoichiometry

Thus, lets try to build it:

```
python3 main.py -i examples/example_2/chains/pairs -o examples/example_2/results  \
        -f examples/example_2/6gmh.fa
```

In this case, there are several chains that hasn't been added to the model, and some are repeated. So, let's select a different starting point and limit the chains with the stoichiometry, so at maximun there is 1 copy of each chain.

```
python3 main.py -i examples/example_2/chains/pairs -o examples/example_2/results  \
        -f examples/example_2/6gmh.fa -start examples/example_2/chains/pairs/6ghm_VA.pdb \
        -s examples/example_2/6gmh.stoic
```

The 6gmh.stoic file contains the stoichiometry of the protein. In essence, is a csv file, without header, in which each line follows the same structure: ,. Each chain_name must be in the fasta as a sequence

This starting pdb was not in the model, and with this we force it to be included in the model, and start growing from there. And this way, the result contains far more chains in the model and it's far more similar to the original one.

Example 4 (5nss)

Here we want to show how to use two parameters we haven't shown yet: distance and threshold. Distance is a value that considers if two atoms collide, which is not allowed, as two atoms very close would have very high energy (if the separation between them is less than distance, it's considered invalid. However, to avoid some errors, up to 10 atoms colliding are allowed to consider a valid interaction). Threshold, however, makes reference to the homology percentage between the pdb sequences between themselves and between the sequences in the fasta. This allows for certain flexibility in sequences with some mutations that may alter the conformation of the chain.

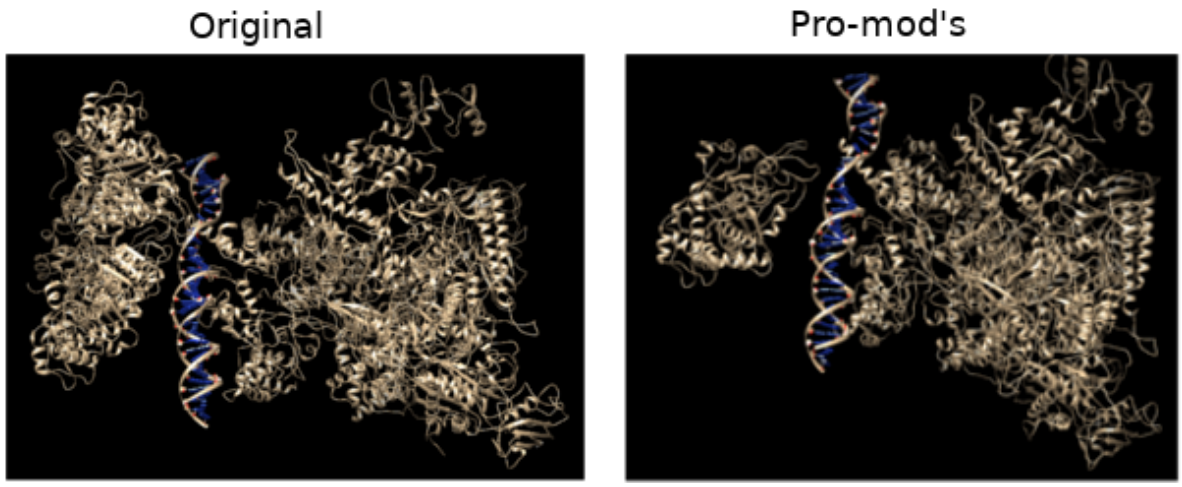

Figure 3: 5nss structure obtained from rcsb.org vs composed one

```
python3 main.py -i examples/example_4/chains/pairs -o examples/example_4/results  \
        -f examples/example_4/5nss.fa -d 2 -t 0.9 -start examples/example_4/chains/pairs/5nss_NG.pdb
```

In this case, the result is quite good, but let's focus on the speed of the program. As the number of interactions grew, the time it takes for the program also increases. However, there are a few things to comment about it.

1.- Adding a new chain that passes all the checks is the most computationally expensive moment. The number of checks grows as the number of chains of the model increases.

2.- It takes less time to discard an interaction that has no homologous already in the model than adding a new chain. During the first steps of the program, until the model has grown enough, this is the most common case. The number of alignments done is relatively low as there are a few chains in the model, but it can be an issue

3.- Once the model has a considerable number of chains, and particularly if there are several copies of a monomer, it's also noticeable that discarding an homologous protein because doesn't fit with the current parameters is a very time-consuming processes. The program tries to introduce the chain using all the possible alignments whose score is higher than the threshold. This is an important issue for proteins with a lot of copies of several monomers.

4.- The stoichiometry must be carefully selected to reach the desired structure. It's possible that the program cannot construct a model with the desire stoichiometry, but it's also possible that the stoichiometry wasn't what the user meant

Example 5 (6om3)

In this last example, let's talk about how the software behaves about the number of input pdbs and stoichiometry related stuff.

```
python3 main.py -i examples/example_5/chains/pairs -o examples/example_5/results  \
        -f examples/example_5/6om3.fa -d 0.3 -t 0.95 \
        -s examples/example_5/stoic.csv
```

In this last example, let's talk about how the software behaves about the number of input pdbs and stoichiometry related stuff.

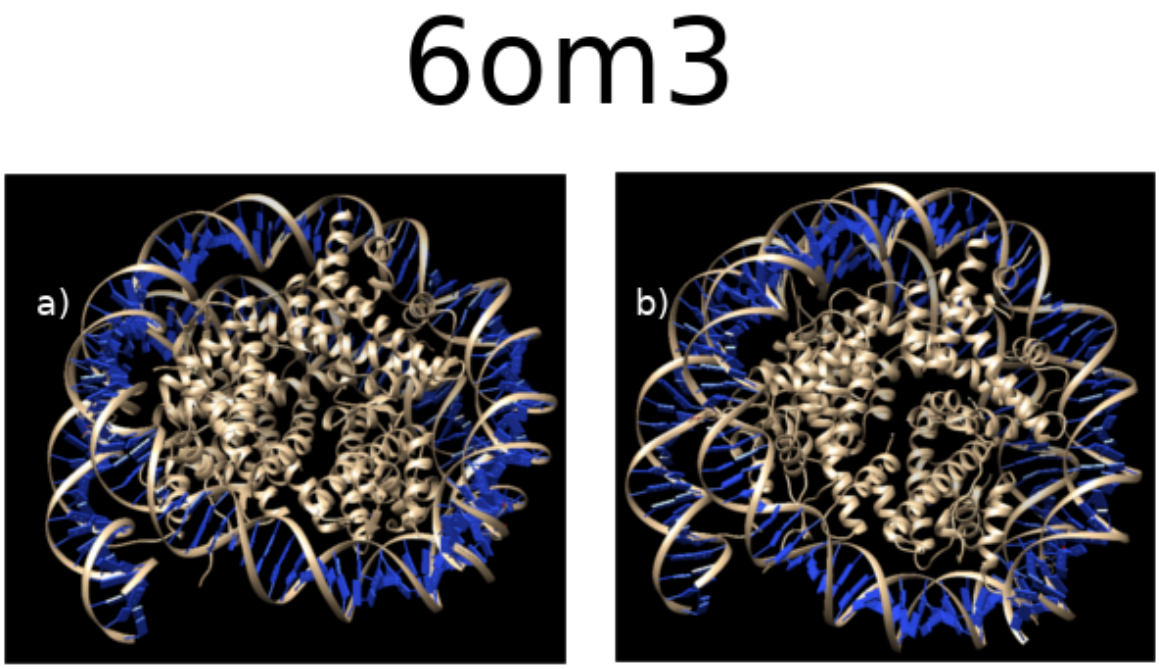

Figure 4: 6om3 builded from: a) rcsb.com b) promod with stoichiometry

There are som chains left in the protein builded with promod, which can be added used custom values of distance or an apropiate starting point, but let's ignore that and focus on topics still not covered in the manual.

If you indicate a stoichiometry file but it doesn't exist, it exits. We could have changed it to a non-stoichiometry builder but we think it's better to do things explicitly as you may or may not notice this error.

An important thing about the stoichiometry is that is critical to make sure that all the model is builded as you want. For example, without any assistance, the work heres is quite good, but with a selected stoichiometry, the result is missing some chains. Here we want to say that is important to check the stoichiometry file to make sure the chains are correctly selected and the number of them is adequate to our purpose.

Limitations

1. The sequences in the pdb and the sequences in the fasta must be similar. The program admits a certain degree of tolerance (which can be selected by the user using the -t parameter) but using sequences in the fasta which are fairly different from the pdb will not allow the assembly of the protein. This is particularly difficult for pdb in which protein tails are not well modelled and they're not present in the pdb but they're in the fasta. The pairwise alignment with the tail will drop the score of the alignment under the threshold, forcing to decrease it or even mistake to which sequence in the fasta corresponds to jus by chance. Thus, our recommendation is to prepare the fasta file with the sequence as similar as possible to avoid mistakes. In case you needed, in this package there is a script included, pdbsplit.py, which can give you the sequences of the chains inside the pdb file, avoiding this kind of errors up to ceratin degree. So, if you're looking for the effect of certain mutations in the desired protein, might be worthy prepare the fasta file according to the pdb sequences.
2. The running time of the program is proportional to the number of pdbs selected and is affected by the

order of the structures and the starting point. The most critical steps: Reading all the pdbs sequences (However, the higher number of interactions, more chains might be added). This is a tradeoff between number of interactions and speed.

3. Selecting different starting points can produce different models. It's difficult to say which is the most adequate starting point, as may vary according to the goal of the assembly.

About Promod