# BYC: theoretical background

## Build Your Complex (BYC) Application

by Marina Lleal Custey and Miquel Àngel Schikora Tamarit

### 1. WHAT IS BYC?

BYC is a command-line application that can be used for building biological macrocomplexes, which may include proteins, DNA or RNA molecules, starting from pairwise interactions between subunits of the complex to be modelled. BYC aims to be a helpful tool in structural analysis and offers several options to the user in order to either explore limitless possibilities or specify clearly detailed conditions.

BYC has been developed by Marina Lleal Custey and Miquel Àngel Schikora Tamarit as a project for both Introduction to Python and Structural Bioinformatics subjects of the MSc Bioinformatics for Health Sciences at UPF (Universitat Pompeu Fabra).

### 2. USAGE

BYC consists on two separate scripts as well as a module with implementations of different functions. The first script is intended to be used to build a complex, the second to refine it. Go to the README file for information on the usage of BYC.

### 3. BUILD THE COMPLEX

This section explains all the conceptual properties that BYC implements to solve the problem of re-building a complex.

#### 3.1. MATCHING FILES AND CHAINS

The BYC pipeline starts by storing information about unique chains found in the directory of the pairwise interaction files. For each chain found, it is checked whether it is already stored or otherwise a unique identifier is assigned to it (given by the user if a MULTIFASTA file is provided, or assigned randomly by the program). This way, the same chain (defined as having a sequence identity greater than 95%) found in different files can be labelled and recognised as the same every time it is found, enabling the matching of equal chains coming from different files.

#### 3.2. SUPERIMPOSITION, ROTATION AND ADDITION

The approach that BYC uses to build complexes from pairwise interactions is to take two pairwise interactions with a common chain (previously matched with the same ID), then superimpose only these two common chains by fixing one and moving the other, obtain the rotation matrix of that operation and then use it to copy and move the other chain from the moving structure to the fixed and add it to the model. Therefore, from a 2 chain current model, a 3 chain model can be obtained.

### 3.2.1. CORE ATOMS

We have considered as “core atoms” for each residue the alpha-carbon (‘CA’ is the atom id) for protein residues and the backbone phosphate (‘P’ is the atom id) for nucleic acids. These atoms are used for performing the superimpositions and/or calculations of distances in generate_pairwise_subunits_from_pdb. Any residue lacking these atoms is dropped from the analysis.

## 3.3. RECURSIVE METHOD OF BUILDING A COMPLEX

In order to be able to build large complexes, a recursive approach is needed so that in the first level of recursion, a pairwise interaction is chosen and all the pairs containing one of these two initial molecules in common are tried to be added to form a complex. Then, in the next level of recursion, the complex model has grown and the addition of more subunits is tried with the model at that point, and so on and so forth. Thus, starting from a file of a pairwise interaction, a recursive function called build_complex is called, which performs the previously explained superimposition, rotation and addition in a recursive way.

## 3.4. SAME CHAIN CRITERIA: RMSD cutoff

There are several situations that may happen during the superimposition process and need to be addressed. For instance, if the program may be superimposing two common chains (defined as having the same sequence) and moving one in the same place where another equal chain already is. Also, when comparing two structures and trying to determine if they are equal.

In order to address this issue, BYC uses a superimposition followed by a RMSD calculation and classification as same chain or not according to a cutoff.

Root Mean Square Deviation (RMSD) is the most commonly used quantitative measure of the similarity between two superimposed atomic coordinates and it can be calculated for any type and subset of atoms; for example, C-alpha atoms of the entire protein, C-alpha atoms of all residues in a specific subset, etc.

The distribution of the protein backbone RMSD for a large number of experimentally determined structure pairs of identical proteins in the PDB[1] demonstrates that for the majority of pairs, the RMSD ranges from 0 to 1.2 Å, due to inherent protein flexibility and experimental resolution limits. Then, the backbone RMSD values are distributed around 2.3 Å for easier homology modeling cases, and around 4.5 Å for distant homology modeling cases. Taking this data into account, and willing to allow for some structural flexibility between pairs but not excessive, BYC sets the RMSD cutoff at 3.0 Å in order to classify a structural superimposition of two molecules as the same or not. This is calculated between the “core atoms” of each of the residues.

## 3.5. DEFINING CLASHES BETWEEN CHAINS

A definition and calculations for clashes need to be established in order to build meaningful complexes which are plausible according to physicochemical rules and also open different branches to try different options while adding new chains to the complex.

The ideal approach to tackle this issue would be similar than the one used by UCSF Chimera software, which takes into account the specific VDW radii of each type of atoms[2].

Nevertheless, in order to speed things up and because of time and computational resources’ limitations, BYC uses a simpler approach. When a new chain is added to the complex, we use the module NeighborSearch to ask how many atoms of this chain are located closer than a threshold distance to those in the current structure. This threshold distance has been selected to be 2.5 according to the minimal possible distance between two VDW radii of two atoms interacting through a hydrogen bond[3].

Our method is subjected to superimposition error, which may explain some of the clashes. To solve this we define as “clash between chains” the situation in which more than 20 atoms are actually clashing. The idea is that the optimization process may correct them, so that we have to accept a certain (low) number of clashes when building the complex.

## 3.6. CREATING BRANCHES: PRODUCING SEVERAL COMPLEXES FROM THE SAME INPUT STRUCTURES

There might be some cases in which it is possible to generate different complexes with the same input files, because the complex may have changing architecture and composition across conditions. If this is the case, our hypothesis is that it can be detected by clashing events when building the complex. If a set of pairwise interactions can only build one complex the only clashes that occur while building it result from including new chains at the exact same place where an equal chain was. Conversely, if a chain clashes against the complex while being added (not because of repeating an addition operation of a subunit that was already there) it indicates that an alternative complex (and an alternative complex-building path) may be possible. Our program includes this possibility, in a way that opens an alternative building of the complex (opens a branch) through recursion.

When such a branch-opening clash is detected, the pipeline opens a new branch removing the clashing subunit (or subunits) that was already there and replacing it by the new one and also keeping the previous model. After that, the information of these two clashing chains' IDs is stored in a list called non_brancheable_clashes (argument of the build_complex function) in a way that if this clash is found again between the same two chains, a new branch will not be opened, as these two options have already been consider before and it would be redundant.

## 3.7. CHECKING FOR UNIQUE MODELS: SUPERIMPOSING WHOLE STRUCTURES

After generating a model, it is compared to the already saved models to make sure it is not equal to any of them. This is performed with the function structure_in_created_structures. This function checks if these structures have the same chains (if not the new structure is saved) and then performs a superimposition of the common atoms of the two whole structures to determine if the newly built structure is already saved.

## 3.8. OUTPUT: LARGE COMPLEXES CANNOT HAVE SINGLE-MODEL ALPHANUMERIC CHAINS

The number of unique single-character chain identifiers is 62 using an alphanumeric code, which is insufficient for large complexes. As many programs, such as Chimera, have problems displaying chains with non single-character identifiers we decided to split the created complex in models of 62 chains, so that each of them has a unique identifier within several models.

## 3.9. HOMODIMERS AS INPUTS HAVE TO BE HANDLED SPECIFICALLY

As written above, our pipeline includes new chains when building a complex through a process of superimposition in which a given chain acts as "common" (see 3.2.). This represents a problem when two equal molecules interact with each other within the complex (which appear as homodimers in the input of the program), as it is not clear which have to be the "added" or "common" chain and there would be two possibilities of performing the superimposition.

We have implemented a solution in the build_complex function that assumes that both chains in the homodimer could potentially act as "common" and/or "added", so we try to include them both.

## 4. REFINE THE MODEL

After obtaining the model by a series of superimpositions, an optimization process needs to be run in order to minimise the energy of the final model and turn it into a “relaxed state”. It should be noted that during all the process of building the complex, some superimposition errors may happen and then will not be solved at any point. Moreover, the fact of using pairwise interactions to model complexes assuming that these pairwise interactions will remain exactly the same with the presence of more molecules around, can be also a source of error. So then, this optimization can be done with BYC itself by running refine_model.py, which uses PyRosetta4 package[4] to refine the structure. This is a classical relax optimization pipeline.

## 5. BYC PERFORMANCE ANALYSIS

In order to check the performance of BYC, we used it for rebuilding several complexes (containing DNA, RNA and proteins) that were previously crystallised as a whole. These complexes were splitted in the unique pairwise interacting subunits and BYC was ran. Afterwards, the complex built was compared to the original one. To mention some examples, BYC can completely rebuild a nucleosome, a proteasome, a phosphate dehydratase and a microtubule.

### 5.1. A TESTING MODE TO CHECK ANY EXISTING COMPLEX

An issue of our application is that it is hard to find valid template files to run and test the functionality. We provide an option that allows the user to build any existing complex (in the form of an input .pdb or a .cif file) from the unique pairwise-interacting subunits.

The implementation is performed through the generate_pairwise_subunits_from_pdb function, which takes a pdb, splits it into pairwise interacting chains (defined as chains that have at least one residue at less than 7 Å of distance between each other), and applies a rotation/translation operation to them before saving. In addition, the function checks if the pairwise subunit is unique, avoiding redundancies between input files.

This simulates a real situation in which you would input pairwise interacting subunits of different sources. The user can use this “testing mode” to evaluate the functionality of our pipeline.

### 5.2. ANALYSIS OF EXAMPLES

The performance of BYC has been analysed on different complexes. Below is a description of the particularities of each example, which are also examples of the limitations of our approach.

**Nucleosome (PDB id: 3kuy)**

The reconstruction of the nucleosome complexed with DNA was successful. However, there is an important consideration when building DNA-protein complexes like the nucleosome, where the interaction of the proteins with the DNA is non-specific. This is a case in which the DNA can be used as a ‘common chain’ and two DNA chains can be superimposed but this would not be extrapolable in all cases.

**Phosphate dehydratase (PDB id: 2f1d)**

The phosphate dehydratase could be successfully reconstructed using only the asymmetric unit and obtaining the biological assembly as a final model. Thus, from the minimal set of interactions and subunits, a whole complex could be built, validating the ability of our pipeline to address a variety of situations.

**Proteasome (PDB id: 4r3o):**

The proteasome could also be successfully reconstructed.

**ATP synthase (PDB id: 5dn6)**

The reconstruction of the ATP synthase was mostly successful, with some concerns. First, it needs to be taken into account that the ATP-synthase available in the PDB includes a few chains composed by unknown residues (UNK), which were dropped for the rebuilding, as it was not possible to align them. Therefore, these chains could not be matched among pairs from the input files and this is why it some chains were not found in the generated model.

Also, due to the greediness of our approach, there is a repeated chain which is only present once in the initial PDB but several times in our model. So, if the final desired model needs to have the same stoichiometry as the initial, this issue could be fixed by determining the stoichiometry and building several models. However, it is also interesting to note that if these interactions appear in the model built by BYC is that they are possible and that the initial PDB structure or the pairwise interactions given by the user could be incomplete.

**Ribosome (PDB id: 4v4a):**

The reconstruction of the ribosome was almost successful also. We noticed that a few chains were missing (likely because of the superimposition error), and some others were added in a different place than in the original complex, likely because of the greedyness of the approach, as in the ATP synthase.

**Virus capsid (PDB id: 1e57):**

The virus capsid was the most challenging complex to rebuild. It is composed by a repetition of the same molecule, and the 3D arrangement of the final capsid is hard to achieve because of the greediness of the pipeline. To simulate a real situation, we tried to rebuild it from the pairwise interactions resulting from joining two asymmetric units of the PDB Biological assembly (provided as virus_2_asymmetric_units_1e57.pdb).

We can observe how the capsid was built, but there was a point in which new subunits were added surrounding it, and this is likely to be because the pairwise interactions chosen are arranged in a way that they can be added to the already completely formed capsid. This is also likely to be a result of the imprecision in superimpositions. This can be solved by adjusting the -n_chains parameter of the pipeline to achieve the precise number of subunits for closing the capsid. It is also likely that the simulated pairwise interactions are not the necessary ones for rebuilding the capsid.

### 5.3. BYC LIMITATIONS

The approach of BYC shows great performance in many situations already discussed, but also several limitations that are commented in this section.

The main limitation of the algorithm implemented is the fact that it uses superposition, resulting in a certain error each time a new molecule is added to the complex. This propagates with complex size, and results in errors in the final complex building, such as the virus. A better approach would be to refine the structure after each chain adding, but this would be very expensive in terms of computational cost.

In addition, it has been difficult to test the pipeline on real cases, specially when testing on structures that are actually models (such as the virus). The main problem is to input the exactly necessary input interactions to rebuild the complex. In addition, it is difficult to find examples in which a complex can adopt multiple conformations, so that the -exh and -n_models options have been difficult to test.

## REFERENCES:

[1] Kufareva I, Abagyan R. Methods of protein structure comparison. Methods Mol Biol. 2012;857:231-57. doi: 10.1007/978-1-61779-588-6_10.

[2] Find Clashes/Contacts. UCSF Computer Graphics Laboratory / September 2014. Available at: https://www.cgl.ucsf.edu/chimera/docs/ContributedSoftware/findclash/findclash.html

[3] Tsai J, Taylor R, Chothia C, Gerstein M. The packing density in proteins: standard radii and volumes. J Mol Biol. 1999 Jul 2;290(1):253-66. https://doi.org/10.1006/jmbi.1999.2829.

[4] P. Bradley, K. M. S. Misura & D. Baker, Toward high-resolution de novo structure prediction for small proteins. Science 309, 1868-1871 (2005).