PyInteraph2 user guide

Reference publication:

PyInteraph2 and PyInKnife2 to analyze networks in protein structural ensembles

Valentina Sora[1,2], Matteo Tiberti[1], Deniz Dogan[1], Shahriyar Mahdi Robbani[1], Joshua Rubin[1], Elena Papaleo[1,2]

[1] Computational Biology Laboratory, Danish Cancer Society Research Center, Copenhagen, Strandboulevarden 49, 2100, Copenhagen, Denmark
[2] Cancer Systems Biology, Section of Bioinformatics, Department of Health and Technology, Technical University of Denmark, 2800, Lyngby, Denmark

An early version of the pre-print available on biorxiv:

doi: https://doi.org/10.1101/2020.11.22.381616

1 Introduction

1.1 The software package

The *PyInteraph2* software suite is a package designed to analyze non-covalent interactions in structural ensembles of proteins and biomolecules. It also allows to analyze them using graph theory and the Protein Structure Network (PSN) framework. We refer to PSN when we use contacts between residue atoms or the center of masses of the residue side chains. On the contrary, when we refer to weak interactions (salt bridges, hydrophobic interactions, hydrogen bonds and, merged networks) the networks are defined as IIN (Intra/Intermolecular Interaction Networks), according to the first *PyInteraph* publication (Tiberti et al., 2014).
These ensembles can be derived either from simulations, such as Molecular Dynamics (MD) or Monte Carlo simulations or from experiments (such as NMR-based structural ensembles).
The package is the updated version of the *PyInteraph* package (Tiberti et al., 2014). It consists of a *Python* library with internal C loops, integrated with *Python* through a *Cython* abstraction layer, together with executable scripts.
The main features of the program are accessible through straightforward command-line scripts. In particular, the package allows the user to:

- calculate different classes of noncovalent interactions, i.e., salt bridges, hydrophobic interactions, and hydrogen bonds, and use this information to provide IINs;
- estimate contacts between all the residues of a protein-based center of masses corrected center of masses or atomic contacts to calculate cmPSN, ccmPSN, and acPSN, respectively;
- customize the analysis defining new moieties of interest, including non-protein biomolecules, and modifying some of the parameters related to the analysis;
- account for both intra- and intermolecular interactions;

- visualize these interactions and the networks through the *xPyder* plugin for *PyMOL* [ref];
- calculate interaction pseudo-energies between pairs of residues using the empirical force field HUNTER (Cohen et al., 2009, Potapov et al., 2010)
- IINs and PSNs can be constructed according to the occurrence of the interactions in the structural ensemble, as well as to combine them in a macro-IIN, which can, in turn, be weighted on the base of the pairwise pseudo-energy values or other metrics of interest
- Analyze the resulting macro-IIN or individual IINs and PSNs using different graph-theory based metrics (hubs, connected components, centrality measurements, shortest paths of communication, and metapaths)

The user-accessible features of PyInteraph are handled by a few front-end scripts:

- `pyinteraph`: to perform the calculation of the interaction class of interest on the input structural ensemble. It provides as output the interaction graphs, interaction maps, and interaction energy maps;
- `filter_graph`: to perform estimation of the threshold of significance for the interaction occurrence and to handle graph filtering according to the selected threshold;
- `graph_analysis`: to perform some basic network analyses (i.e. calculation of hubs, connected components) on the graph of interactions;
- `path_analysis`: to compute path-related metrics on the graph;
- `centrality_analysis`: to calculate centrality measures (i.e. closeness centrality, betweenness centrality, etc.) on the graph
- `dat2graphml`: to convert our adjacency matrix file format to the more widely used graphML format

1.2 Installation

PyInteraph can be obtained from the *Github* repository at https://github.com/ELELAB/pyinteraph2. It is installed as a standard Python package using setuptools. We also provide a Conda YAML file as well as a requirements.txt file for pip that list the required packages to install and use PyInteraph, and can be used to create environments ready to run it. Detailed instructions on how to do so are available in the INSTALL file on the GitHub repository.

1.3 Running unit tests

To run the unit tests for the *PyInteraph2* suite, you can simply run the following command in the root directory of the package:

```
$ pytest
```

This will run all tests in the $PYINTERAPH/tests directory, and print the outcome of such tests to the console. All tests should pass; if this isn't the case please let us know by posting an issue on the *PyInteraph2 GitHub* repository (.https://github.com/ELELAB/pyinteraph2/issues).

1.4 Classes of interactions and contacts covered by Pyinteraph2

The main *PyInteraph2* script (*pyinteraph*) takes care of calculating intramolecular interactions as defined in the next sections.
For each analysis, the main output of the program are
i) a **csv file** containing the list of interactions and the associated values that measures the extent of the interactions as calculated from the ensemble, i
ii) **a dat file**, including the graph adjacency matrix file that is used to define the interaction graph. The adjacency matrix encodes an undirected graph in which each node of the graph is a protein residue. An edge exists between pairs of nodes if a value larger than 0 is recorded in the corresponding matrix element in the adjacency matrix. In such cases, the corresponding matrix-encoded value is the edge weight (in all cases except acPSN, this is the quantification of the occurrence of the interaction in the ensemble).

Here on we use the term **occurrence** for the weights used in the majority PSN or IIN networks. Of note, we used the term 'persistence' in the first version of the software (Tiberti et al., 2014) and revised it in more recent applications of the tool. Occurrence is defined as how many models (or frames) in a given structural ensemble have a certain property (used for edge definition) over the total number of models belonging to the ensemble, in percentage. For instance, a hydrogen bond with an occurrence of 70% is present in 70% of the structures of the ensemble. This is used as a measure of the extent of the interactions in the ensemble and it is used in all cases except for acPSN (see below) or networks weighted on pseudo-energies. Some of our analyses are handled by configuration files. They allow defining which residues or atoms take part in the analysis and also allow to include non-protein residues, which are treated in the same way as protein residues. This is the case for salt-bridges, hydrogen bonds, hydrophobic clusters and cmPSN analysis. Other methods, such as acPSN and pseudo-energy maps, are only available for protein residues as they are just defined on those. Normalization factors for acPSN are also stored in a configuration file, which allows to add or change normalization factors for new residues if needed.

1.4.1 Salt bridges

PyInteraph calculates salt bridges between oppositely-charged residues as follows. It considers the atoms belonging to charged groups of charged residues as defined in the charged groups configuration files. For any oppositely-charged charged residue pair, a salt bridge between them will be identified if at least two atoms belonging to the two groups are closer than the chosen cut-off. This calculation is performed for every model in the ensemble and the final occurrence value is written in the output files, if the occurrence is larger than a predefined cut-off, which can be changed with the option –sb-perco. In the case of the salt-bridge analysis, the output adjacency matrix will contain the same occurrence values as the csv file. The distance cut-off for salt bridge identification can be modified by using option --sb-co.

1.4.2 Hydrogen bonds

Hydrogen bonds are calculated between pairs of atoms that can act as either acceptor, donor, or both. These lists are defined in the hydrogen bonds configuration file as lists of atom names

(see below) and can be customized at will. PyInteraph identifies hydrogen bonds based on the distance between the donor and acceptor atoms as well as the donor-hydrogen-acceptor angle, using the hydrogen bonds calculation module from MDAnalysis. The cut-offs for these parameters can be changed by means of the `--hb-co` and `--hb-angle` command-line options, respectively. As a default, `--hb-co` is set to 3.5 Å and `--hb-angle` at 120º.

It is possible to further specify which parts of the protein(s) should be subject to the hydrogen bonds analysis by means of the `–hb-class` option (as a default "all") as detailed in the `pyinteraph` help text (here "sc" means side-chains and "mc" means main-chains - see the output of `pyinteraph -h`). Using option "custom" here will also allow the user to select more specific groups by supplying the `--hb-custom-group-1` and `--hb-custom-group-2` options. These options accepts as arguments MDAnalysis selection strings (see https://docs.mdanalysis.org/stable/documentation_pages/selections.html). For instance, the argument "resid 1:500" would select residues with number 1 to 500. The hydrogen bond analysis outputs all the identified hydrogen-bond interactions in the output csv file which also includes which atoms were involved in each hydrogen bond, after filtering for the occurrence cut-off (`--hb-perco`, as a default this is set to 0 since we recommend to use `graph_analysis` to estimate the occurrence cut-off to be applied). Similar flags are available for generating the csv files of the other classes of interactions described below. The occurrence values in the output graph are calculated in a slightly different manner, i.e., a hydrogen bond between a pair of residues is considered to be present if at least one hydrogen bond exists between them for each model of the ensemble. The number of frames in which this happens is counted towards the calculation of the occurrence values in the output graph files.

1.4.3 Hydrophobic clusters

This analysis identifies the interactions between hydrophobic residues in the protein. It does so by calculating the distance between the center of mass of side-chains of pairs of residues. The center of mass of a group of N atoms with masses $m_1, m_2, \ldots, m_n$, a total mass M, and coordinates $x_1, x_2, \ldots, x_n$ is defined to be the mass-weighted average position of an atom in that residue:

$$CoM = \frac{\sum_{i=1}^{N} m_i x_i}{M}$$

An interaction between two residues is then identified if the distance between the centers of mass of their respective side chains lies below a specified cut-off, selectable with the --hc-co option. The occurrence cut-off can be selected by means of the `--hc-perco` option. The occurrence values are also used as weights in the graph adjacency matrix output.

The list of residues this analysis consider can be specified on the command-line as a comma-separated string using the `--hc-residues` (for example, `--hc-residues` ALA,VAL,ILE)

1.4.4 The inter/intramolecular interaction network (IIN)

A IIN is a network that represents all the important interaction types identified in the previous sections (1.4.1 to 1.4.3), joined together in a single network. It is generated starting from the previously introduced interaction graphs and has it's simply the logical OR of all those networks. Similarly, to what was described so far, nodes of the IIN are protein residues, and an edge exists if the corresponding edge exists in at least one of the graphs used to build it. The network is by default unweighted, with a weight of 1 for every edge. Weights can be however added to the network as explained in the tutorial.

1.4.5 KBP network

This network is based on the HUNTER knowledge-based potential. We refer you to the original HUNTER publications (Cohen et al., 2009, Potapov et al., 2010) and the previous PyInteraph publication (Tiberti et al. 2014) for a thorough description. Briefly, it is a energy potential designed on distances between two pairs of atoms, independently selected for each pair residue type. The final network has the same shape as the previously described networks with one

1.4.6 cmPSN and ccmPSN

The centers of mass PSN analysis (cmPSN) is a generalization of the hydrophobic clusters analysis. It is designed to be a generic measure of the interaction between residues disregarding their type or the type of interaction they partake in. The calculation works exactly in the same way as the hydrophobic clusters analysis (see section 1.4.3) with corresponding command-line options `--cmpsn-co`, `--cmpsn-perco`, `--cmpsn-residues`. The residue list includes by default all the natural amino acids except glycine.

Distance between centers of mass is a common approximation of inter-residue distance, but it implicitly represents residues as point masses. We know however that different residue types have different side-chain composition and size, which means that the center of mass will be more or less buried (i.e. distant from the surface) for different residue types. This can introduce a bias when using the centers of mass to identify if residues are in contact, as the centers of mass of two larger residues in contact will likely be further away than the centers of mass of two smaller residues in contact. In order to account for this effect we have developed a corrected cmPSN (ccmPSN) analysis that refines the model of inter-residue distances by incorporating information gained from the radius of gyration of the residues. The radius of gyration (R_g) is calculated as the mass-weighted root mean square distance of atoms to the center of mass of the residue side-chains:

$$R_g = \sqrt{\sum_{i=1}^{N} m_i(x_i - CoM)^2/M}$$

Where N is the number of atoms, m_i and x_i are the position and mass of atom i, CoM is the position of the center of mass and $M = \sum_{i=1}^{N} m_i$. When running the ccmPSN analysis, the final corrected distance between side-chains center of mass is calculated as:

$$c_{AB} = d_{AB} - R_{gA} - R_{gB}$$

Where c_{AB} is the corrected distance between residues A and B, R_{gX} is the radius of gyration of residue X and d_{AB} is the distance between the centers of mass of residues A and B. c_{AB} is then compared with the distance cut-off to determine if residues are in contact or not. The correction to cmPSN can be turned on by means of the `--cmpsn-correction` option.

1.4.7 acPSN

In PyInteraph2, we also included an implementation of the atomic contacts-based PSN (hereafter named acPSN) first devised by Kannan and Vishveshwara (Kannan and Vishveshwara, 1999). acPSN was originally designed for static protein structures but has been subsequently applied to conformational ensembles as well (Seeber et al., 2011). This approach considers residues as nodes of the PSN and relies on the calculation of atom pairs in contact between two residues to identify the edges. Only pairs of residues whose sequence distance is higher than a preset threshold (prox_cut) are considered for edge calculation. In detail, for each pair of residues, the number of atom pairs within a certain distance cut-off is computed. Then, the result is normalized by the square root of the product of two pure numbers, called “normalization factors”. A normalization factor is defined for each residue type and acts as a proxy for the residue's propensity to form contacts. They were first calculated for the canonical amino acids by Kannan and Vishveshwara (Kannan and Vishveshwara, 1999). For each pair of residues, the normalized result represents the “interaction strength” between the two residues. If the interaction strength exceeds a pre-defined cut-off (i_min), an edge is drawn between the two residues with weight equal to the interaction strength. In the case of a conformational ensemble, an acPSN is calculated for each structure and only edges present in a minimum pre-determined percentage of structures in the ensemble (p_min) are kept in the acPSN representing the whole ensemble. This percentage represents the “occurrence” of the edge in the structural ensemble. The interaction strength associated with each edge in the final acPSN is the average interaction strength for that edge over all the structures where the edge was present. The edges in the final acPSN can be weighted either on their average interaction strength or on their occurrence.
The calculation of acPSN is accessible through the pyinteraph executable. Several options are available to fine-tune the parameters of the acPSN construction. For example, you can set the distance cut-off for a pair of atoms to be considered in contact with --acpsn-co. The default distance cut-off is 4.5 Å, in agreement with the cut-off used to calculate the normalization factors performed by Kannan and Vishveshwara. On the other hand, an interaction strength cut-off different from the default (3.0) can be set using the `--acpsn-imin` option. Users can also set an occurrence cut-off on acPSN’s edges (--acpsn-perco), defaulting to 0.0 (no cut-off). Edge weighting can be selected via the `--acpsn-ew` option, with “strength” (interaction strength) being the default. Customized normalization factors files can be passed through the `--acpsn-nf` option. Unless the `--acpsn-permissive` option is active, residue types found in the topology or reference file which do not have a normalization factor associated will cause pyinteraph to exit with an error. However, users can set a default value for residue types with no normalization factor via the `--acpsn-nf-default` option when running in permissive mode.

1.5 Customizing the PyInteraph analysis

The analyses performed by PyInteraph can be customized by defining the groups and atoms that are used to perform the calculation or other aspects, such as the normalization factors for acPSN. This is done by modifying the default configuration files and supplying them to PyInteraph and supplying them to the program by using the corresponding options (--sb-cg-file for salt bridges, --hb-ad-file for hydrogen bonds, --acpsn-nf-file for acPSN normalization factors. This is explained in section 3.3.2 of the tutorial.

1.5 The main output file formats

1.5.1 CSV files

In PyInteraph2, CSV files are used to store lists of edges (i.e. residue-residue contacts) found in a PSN. These CSV files are produced by pyinteraph when constructing a PSN, and are generated if the user passes one (or more) of the following options:

- `--cmpsn-csv` if the PSN built is a cmPSN (-m, --cmpsn option used when running).
- `--acpsn-csv` if the PSN built is an acPSN (-a, --acpsn option used when running).
- `--hc-csv` if the PSN built is network of hydrophobic contacts (-f, --hydrophobic option used when running).
- `--sb-csv` if the PSN built is network of salt bridges (-b, --salt-bridges option used when running).
- `--hb-csv` if the PSN built is network of hydrogen bonds (-y, --hydrogen-bonds option used when running).

1.5.2 DAT files

Differently from CSV files, DAT files in PyInteraph2 are used to store matrices representing the PSNs. They are simple ASCII text files which encode for a symmetric square matrix, which is the graph adjacency matrix. Each row and column of the file represent a specific residue and each matrix element represents the interaction between them (i.e. a graph edge). A value of 0 indicates that the edge between those two residues doesn't exist, a different value represents the weight of that interaction (or 1 for unweighted graphs). They are used as inputs/outputs in `pyinteraph`, `filter_graph`, `graph_analysis`, `path_analysis`, `centrality_analysis`.

1.5.3 GRAPHML files

Finally, Pyinteraph2 generates graphml formatted files (GFFs) to extend the utilization of already generated PSN matrices. GFFs might be given as input to any graphml-supported graph analysis and/or visualization tools, e.g. Cytoscape. The generation of GFFs is operated through the conversion of PSN-stored dat files via `dat2graphml`.

2 Graph analysis of PSNs

2.1 Network filtering and determination of significance threshold

Several interactions will be at very low occurrence in the `pyinteraph` output files, due to the intrinsically dynamic nature of proteins and the large conformational variability embedded in protein structural ensembles. The lower occurrence interactions are therefore likely not to represent an important interaction for protein structure and dynamics and should be discarded from further analyses. Furthermore, many network properties that are calculated by PyInteraph2 do not consider network weights but just network topology. It is therefore advisable to filter the network to remove very infrequent interactions from the graphs, such as those below a certain weight (most often occurrence) value, also called p_{crit}. This procedure is referred to as filtering here.

As in many cases, finding the right threshold for this is not trivial. The PyInteraph2 suite provides a method to define a significance threshold for the interaction occurrence values, called p_{min}, previously used in other applications of graph theory to the analysis of protein structures (Brinda and Vishveshwara, 2005). This is based on analyzing the node size of the largest connected component (an isolated portion of the graph - see below) at different filtering thresholds. In protein-based networks this value has been suggested to have a roughly sigmoidal trend with increasing p_{min} and that the inflection point of such sigmoid would strike a good balance between having a very connected network with non-relevant edges and a too disconnected network.

PyInteraph allows to calculate the size of the largest component at varying p_{min} as well as to attempt the automatic identification of p_{min} by fitting a sigmoidal function to the identified. The function has the following form:

$$f(x) = \frac{m}{1 + e^{k(x - x_0)}} + n$$

where m, n, k and x_0 are the parameters modified by the fit.

2.2 Basic network properties

2.2.1 Connected components
Connected components are subsets of nodes such that a path exists between any arbitrary pair of nodes within that subset, while no path exists between any arbitrary pair of nodes between different of such subsets. They are, in fact, isolated portions of the network. These usually represent the more interconnected parts of the protein.

2.2.2. Hubs
Hubs are graph nodes (i.e. residues) connected by a relatively high number of edges in the graph (i.e. they have a large node degree). Hubs are generally considered in graph analysis of protein structures as residues characterized by more than 3 or 4 edges in the graph (Fanelli and Felline, 2011; Vishveshwara et al, 2009). In a PSN a hub residue is typically in contact

with many others, and so is considered to be important for protein stability as well as for the transmission of structural information.

2.2.3 Paths

A path between two nodes is an ordered list of edges that joins a sequence of nodes that are all distinct. In practical terms, a path is a list of distinct nodes chosen so that each consecutive pair is connected by an edge and that it is possible to travel from the first to the last node through edges following the list of nodes. Many paths may connect any pair of given residues, but we are especially interested in shortest paths, i.e. paths connecting source and target residues through the lowest possible number of nodes, as they are likely to be the most direct route in a PSN between different regions of a protein.

2.3 Metapaths

Metapaths provide a global picture of the structural communication that occurs in a PSN (Fanelli et al., 2013). It consists of the set of the most common edges and nodes in the graph found in the whole set of shortest paths between pairs of residues. In order to calculate the metapath, the shortest paths are calculated between all pairs of residues that have a sequence distance greater than a predefined threshold. Next, for every node and edge in the graph, the frequency of that node or edge in the pool of shortest paths is calculated. This is known as the recurrence value of the edge or node. Finally, both nodes and edges must be filtered according to their recurrence value so that only the most recurrent nodes and edges remain. The output of the metapath analysis is a subgraph of the original graph that outlines the most common edges found among the set of shortest paths, outlining the most important global routes for structural communication in the protein structure.

2.4 Centrality measures

Centrality measures are network properties calculated for each node and edge. They are designed to give an idea of how important a residue is for the mantainment of the network topology, even though their exact interpretation differs from case to case. PyInteraph allows to calculate several centrality measure, most of them as implemented in the Networkx Python package. Centrality measures can be calculated for nodes or edges; in the following section, they are defined for nodes unless when noted otherwise.

2.4.1 Degree centrality

The degree centrality for a node is its degree (i.e. the number of nodes it is connected to) divided by the number of other nodes in the graph, i.e. the fraction of nodes it is connected to:

$$C_d = k_i/(N-1)$$

Where k_i is the degree of node i and N is the total number of nodes in the graph

2.4.2 Betweenness centrality

The betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, σ(s,t) is the number of shortest (s,t)-paths, and σ(s,t|v) is the number of those paths passing through some node v other than s,t. If s=t, σ(s,t)=1, and if v∈s,t, σ(s,t|v)=0. It is conceptually similar to metapaths, and gives an idea of how crucial that node of the network is for the communication between all the others. Edge betweenness centrality of an edge e is similarly calculated as the sum of the fraction of all-pairs shortest paths that pass through e:

$$c_B(e) = \sum_{s,t \in V} \frac{\sigma(s,t|e)}{\sigma(s,t)}$$

2.4.3 Closeness centrality

Closeness centrality: The closeness centrality of a node u is the reciprocal of the average shortest path distance to u over all n-1 reachable nodes scaled by the number of reachable nodes:

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

where d(v, u) is the shortest-path distance between v and u, n is the number of nodes that can reach u and N is the number of nodes in the graph. It gives an idea on (the inverse of) how long is on average the shortest path that is needed to reach the considered node from any other.

2.4.4 Eigenvector centrality

Eigenvector centrality: The eigenvector centrality computes the centrality for a node based on the centrality of its neighbors. The eigenvector centrality for node i is the i-th element of the eigenvector x defined by the equation:

$$Ax = \lambda x$$

where A is the adjacency matrix of the graph G with eigenvalue λ, which is the largest eigenvalue. This centrality score gives higher weight to nodes connected to many nodes that also have higher weight. This means that it is likely to give higher scores to groups of nodes (= residues) that are very interconnected between each other.

2.4.5 Current flow closeness centrality (for nodes and edges)

This measure is similar to shortest path closeness centrality but uses random walks instead of shortest path. It allows us to consider not just shortest paths but longer paths as well.

Centrality values can be calculated using the -c option followed by the names of the requested centrality measures. Instead of individually listing the centrality measures, the user can also use “node” to calculate all node based centrality measures, “edge” for all edge based centrality measures and “all” for both node and edge based centrality measures

3 A tutorial on a structural ensemble from CypA MD simulations

3.1 Content of the tutorial package

This tutorial provides a step by step guide to the usage of the different tools implemented in PyInteraph2. We used as a case study one microsecond MD trajectory of the Cyclophilin A wild-type enzyme, previously published (Salamanca Viloria et al., 2017). In particular, we here show examples of analyses on a skipped version of the trajectory including 10000 frames.
This tutorial is based on a trajectory that has been resolved for periodic boundary conditions (PBC) and keeping the protein atoms only in our skipped trajectory. The pre-processing for PBC is always a step required before using `pyinteraph` on an MD ensemble.
After that, it is recommended to use PyInKnife2 (see https://github.com/ELELAB/PyInKnife2) to explore the data and define the cutoffs for the final PSN or IIN calculation. You can do so by running the PyInKnife2 tutorial before this tutorial. In alternative, if you aim only to understand the usage of Pyinteraph2 you can use the cutoffs defined in this tutorial and skip the PyInKnife2 tutorial for the moment.

The tutorial package is available at the PyInteraph website (https://github.com/ELELAB/pyinteraph2/tree/master/tutorial) and includes, the following folders:

1.index_files 2.filt_tprs 3.filt_trjs 4.frames 5.psn

To run the tutorial is sufficient to start from the folder 5.psn. The other folders (1-4) have been used to prepare the needed index files, trajectories or topology for the analyses.

```
cd 5.psn
```

3.2 Required software

The tutorial assumes that the user has completed the installation of PyInteraph2. Please refer to the INSTALL documentation file for more information (https://github.com/ELELAB/pyinteraph2/blob/master/INSTALL).

3.3 Preparation of trajectories and configuration files

3.3.1 Preparation of trajectory

As stated above, the molecules in the trajectory need to made whole before the analysis (i.e., PyInteraph2 won't solve PBCs). We also recommend to remove all molecules that are not involved in the analysis, including solvent. We here provide a xtc file **traj_prot_dt1000.xtc** where PBC have been already solved.
The file is in the folder **3.filt_trjs**

3.3.2 Preparation of custom configuration files for salt-bridges and hydrogen bonds analysis

Two configuration files are available to customize the PyInteraph2 analysis of hydrogen bonds and salt bridges. Defaults for these files are provided in the installation package and used here.
However, the user can supply custom configuration files to the pyinteraph script through command-line options if need be. We suggest using the original files as templates for further modifications.
It should be noted that changing these files allows supporting unusual atom types and topologies, including potentially non-protein residues and coarse-grained systems.

The configuration files are called:

- *hydrogen-bonds.ini (https://github.com/ELELAB/pyinteraph2/blob/master/pyinteraph/hydrogen_bonds.ini)*

This file controls which atoms (as in atom names of any residues) should be considered as acceptor or donor atoms for hydrogen bonds, in two comma and space-separated lists. Of course, the same atom name can appear in both lists. A custom hydrogen bonds configuration file can be supplied using the `--hb-ad-file` option.

- *charged_groups.ini (https://github.com/ELELAB/pyinteraph2/blob/master/pyinteraph/charged_groups.ini)*

This file describes the charged groups recognized by PyInteraph2 in the context of salt-bridges analysis, for each residue name.
The CHARGED_GROUPS section describes all the recognized charge groups. Their names are arbitrary strings, that must end with 'p' (lowercase p) or 'n' (lowercase n) in case the group is respectively, positively or negatively charged. Otherwise, the group names are not case-sensitive. For instance, AAAp, AaAp and aaap are the same group. AaAP is not a valid name (uppercase final P).
Each group is defined by a list of atom names. All the atoms must be present in each group for them to be recognized as such. However, atom names preponed by "!" must NOT be present in the group for it to be recognized in the topology. The atom names are defined as comma and space separated lists.
The default_charged_groups is a special entry in this section, and includes charged groups that might be present in all the residues, listed as a comma and space separated list
In the [RESIDUES] section, each residue name is associated with one more or charged groups (other than the ones in default_charged groups), again in comma-separated lists.

- *normalization_factors.ini* (https://github.com/ELELAB/pyinteraph2/blob/master/pyinteraph/normalization_factors.ini)

This files incorporate the default normalization factors to be used for the calculation of acPSNs. Here, residues types are identified by their three-letters names. Therefore, any additional residue type must have a unique identifier as well. Please note that these names must match those used in the topology/reference files to identify the different residue types.

3.4 The PyInteraph2 workflow

3.4.1 Calculation of individual interaction classes

PyInteraph2 calculates intra-or inter-molecular interactions on structural ensembles. It is capable of identifying three main types of noncovalent interactions: salt bridges, hydrophobic contacts, and hydrogen bonds. The user can also build a side chain contact map including all residues with a side chain using the program option for hydrophobic interactions and using a list including all the protein residues (except for GLY) as a residue reference list. This leads to the construction of a center-of-mass PSN (cmPSN), in which a pair of residues are said to be in contact for a frame of the trajectory if the distance between their respective centers of mass falls below the distance cutoff (5.0 angstroms by default). In addition, the user has the option to use the experimental ccmPSN (corrected cmPSN) which incorporates corrections to account for differences in residue type. Please be aware that although we are making this feature available to the PSN community, it has not yet been properly benchmarked. We are currently working on its benchmarking on a dataset of proteins with different folds and size. We also provide a atomic-contact PSN (acPSN).

We will start by calculating the three main interaction types present in the structural ensemble defined by the MD trajectory of CYPA. The program we are using is the `pyinteraph` main script and it requires two mandatory input
files:

- A topology file (GRO, PDB, ...)
- A trajectory file (XTC, DCD, PDB, ...)

The topology file contains the definition of the structure(s) whose atomic coordinates are specified in the trajectory file. Topology and trajectory files are required to have the same number of atoms, which should also appear in the same order. The reference file (PDB) is an optional input, and is the structure that is taken as a reference when writing output files. This file is especially important to supply information, which is not explicitly declared in the topology file, such as chain definition (which for instance are not present in GRO files) and residue numbering. The reference file must have the same order and residue sequence as the topology file, although the number of atoms may differ (i.e. it can omit hydrogen atoms). The reference file must also include heteroatoms and ligands, if present, in the same order as in the topology and trajectory files. In the example provided below, only canonical amino acids are present and therefore the configuration files do not require any modification. The pyinteraph script is conceived with a flag system to specify the kind of analyses to be carried out on the trajectory file and related parameters. The main flags are:

- -b, –-salt-bridges : to analyze salt bridges.
- -f, –-hydrophobic: to analyze hydrophobic interactions.
- -y, –-hydrogen-bonds: to analyze hydrogen bonds.
- -p, –-potential : to calculate the pairwise knowledge-based potential pseudo-energy.
- -m, --cmpsn: to calculate the PSN based on side-chain centers of mass
- -a, --acpsn: to calculate the PSN based on atomic contacts

With the following command line, you can run the analyses of the three interaction classes on CYPA MD trajectory. Or you can use the bash scripts in the folder `5.psn/pyinteraph2/interactions`

To calculate salt bridges:

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r pdbmovie_1.pdb --sb-co 5 -b --sb-graph sb-graph.dat --ff-masses charmm27 -v --sb-cg-file charged_groups.ini
```

In this example the choice of the distance cutoff of 5 Å comes from the analyses of the PyInKnife2 plots from the aforementioned tutorial.

The outputs are `salt-bridges_all.csv` and `sb-graph_all.dat`

To calculate hydrogen bonds (in this example sidechain-sidechain but other classes are supported, see `pyinteraph -h` for more information):

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r pdbmovie_1.pdb -f --hb-graph hb-graph.dat --ff-masses charmm27 -v --hb-ad-file hydrogen_bonds.ini
```

The calculation of hydrogen bond is the most time-consuming of the PyInteraph2 tools, it could take up to 15 minutes for this trajectory. If it is too slow we recommend to use even less frames for the tutorial.

The outputs are `hydrogen-bonds_all.csv` and `hb-graph_all.dat`

To calculate hydrophobic interactions:

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r pdbmovie_1.pdb --hc-co 5 -f --hc-graph hb-graph.dat --ff-masses charmm27 -v --hc-residues ALA,VAL,LEU,ILE,PHE,PRO,MET,TRP
```

The outputs are

```
Hydrophobic-clusters_all.csv and hc-grap_all.dat
```

The csv files that can be also used for further data mining if the user is not interested in network analyses but only in the estimate of the interactions or contacts in the protein and their occurrence during the simulation.
In the csv files there is one line for each detected interaction between pair of residues, and the following columns: first residue chain name, first residue residue number, first residue name, first residue interaction group, second residue chain name, second residue residue number, second residue name, second residue interaction group and occurrence percentage of identified interaction.

For example with salt bridges

```
#CHAIN,RES1_ID,RES1_NAME,RES1_CHARGEDGROUP,RES2_ID,RES2_NAME,RES2_CHARGE_GROUP,OCCURRENCE_
PERC.
SYSTEM,28,LYS,sc.lys.NZp,SYSTEM,27,ASP,sc.asp.COOn,13.886113886113884
SYSTEM,34,GLU,sc.glu.COOn,SYSTEM,31,LYS,sc.lys.NZp,0.0999000999000999
SYSTEM,44,LYS,sc.lys.NZp,SYSTEM,34,GLU,sc.glu.COOn,29.67032967032967
SYSTEM,44,LYS,sc.lys.NZp,SYSTEM,43,GLU,sc.glu.COOn,64.03596403596403
SYSTEM,81,GLU,sc.glu.COOn,SYSTEM,31,LYS,sc.lys.NZp,29.47052947052947
SYSTEM,81,GLU,sc.glu.COOn,SYSTEM,76,LYS,sc.lys.NZp,5.594405594405594
SYSTEM,82,LYS,sc.lys.NZp,SYSTEM,81,GLU,sc.glu.COOn,17.582417582417584
```

Notice that if more than one flag is used in the same command line, it is possible to perform two or more analyses together, for example providing both -y and -f.

The sb-graph.dat, hb-graph.dat and hc-graph.dat files are the so-called IIN files. They are ASCII-encoded numerical matrices, which represent adjacency matrices for weighted graphs, as detailed above and in the publications.

The --ff-masses option controls the force field masses associated with residue atoms, and it is therefore especially important for the analysis of hydrophobic interactions (refer to `pyinteraph -h` to have a full list of the force fields currently available). This option was mainly introduced to improve the support for united-atom force fields, for which atomic masses of united atoms significantly differ from those of all-atom force fields. Indeed, the information about force-field masses is saved in several files under the ff_masses directory, which are written in the standard JSON format. Support for new molecules and force fields can be easily added by generating the mass file from the standard GROMACS force field files, using the provided script parse_masses .

The dat files from the calculations above can be provided to filter_graph and graph_analysis for network analysis, with command lines similar to the ones we will describe below. They can be used alone or in combination to generate a macroIIN (see below).

3.4.2 - cmPSN and ccmPSN

To collect a cmPSN without correction, as showed in the PyInKnife2 tutorial and our previous publication (Salamanca Viloria et al. 2017), in this example we will use a distance cutoff of 5 Å.

```
cd 5.psn/cmPSN
```

To calculate it:

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r pdbmovie_1.pdb --cmpsn --cmpsn-co 5 --cmpsn-csv cmpsn.csv --cmpsn-graph cmpsn.dat --cmpsn-correction null --ff-masses charmm27
```

The outputs (similarly to what obtained for the interactions above) are cmpsn_all.csv and cmpsn_all.dat

In the case of correction (ccmPSN)

```
cd 5.psn/ccmPSN
```

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r pdbmovie_1.pdb --cmpsn --cmpsn-co 2.5 --cmpsn-csv cmpsn.csv --cmpsn-graph cmpsn.dat --cmpsn-correction rg --ff-masses charmm27
```

The output file have the same names as above.
The distance cutoff in this case does not have the same meaning of when we think about contacts between atoms or center of mass in a protein due to the correction that we have applied and thus it should be a small value. We are in the process of benchmarking this PSN and in the tests carried out so far for CYPA a good trade-off seems to be in the range of 2.25-2.5 Å.

A note, we here used the PDB file for topology and reference. One could use for example a PDB file as a reference and a GRO file as a topology (in the case of simulations from GROMACS). This is especially useful if the residue numbering in the topology and the reference files is different. In this case if you provide a reference PDB file, the numbering of the reference will be used in the outputs. It is also an essential requirement if the topology file contains more than one protein chain, because the GRO file does not contain explicit information about chain definitions.

3.4.3 Determination of the significance threshold

The biggest connected component size value calculated for each occurrence value is written in the `filter_graph` output file specified by –c flag, whereas a plot in PDF format of these data is available by passing the -p flag:

filter_graph −d cmpsn_`all.dat` −c clusters_sizes.dat −p clusters_plot.pdf

In several application of PyInteraph a occurrence percentage of 20% is used as a threshold (which seems also a reasonable value in this example). It is still useful to try to validate the choice of the threshold using the filter_graph step with the command line above since it could be system-dependent.

3.4.4 Graph filtering

`filter_graph` can also be used to filter graph files according to the specified occurrence threshold (see previous section). The input adjacency matrix is filtered so that edges with weights below the specified threshold are discarded. This can be easily performed in this way:

```
filter_graph -d cmpsn_all.dat -o cmpsn_all_filtered.dat -t 20.0
```

The flag -t indicates the selected threshold of interaction occurrence values. Both threshold estimation and filtering need to be performed on the individual graphs of each interaction class. Once you have performed this operation on the three graphs for the interactions (salt-bridges, hydrogen-bonds and hydrophobic interactions) and obtained three filtered graphs, you can merge them to obtain a so-called macro-IIN (see below).

3.4.5 Calculation of the interaction energy map

```
cd 5.psn/pyinteraph2/energy_net
```

The interaction energy map can be calculated as follows:

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r
pdbmovie_1.pdb -p --kbp-kbt 1 --ff-masses charmm27 -v
```

Interaction pseudo-energy maps are calculated using the knowledge-based potential HUNTER, as explained in the paper. It is calculated between pairs of residues. More negative energy values indicate stronger interactions between residues and vice versa. The output is available in two distinct formats. The first one (kb-potential.dat) lists every residue pair for which the calculated average interaction energy over the ensemble is nonzero. The same data are also stored in an adjacency matrix format (kbp-graph.dat), which is compatible with the xPyder plugin (Pasi et al., 2012). Finally, the –kbp-kbt option is available to specify the kb*T value in the reverse Boltzmann equation. It is useful in case the user would like to provide a specific unit and temperature for the calculated energy values. The default value is 1.0.

3.4.6 Generating the macro-IIN

```
cd 5.psn/pyinteraph2/macroIIN
```

The graphs needs first to be filtered

```
filter_graph -d cmpsn_all.dat -o cmpsn_all_filtered.dat -t 20.0
```

The calculation of the macro-IIN can be performed by merging the three single filtered IIN graphs. Once again, the process is performed by `filter_graph`.

```
filter_graph -d hb-graph-filtered.dat -d hc-graph-filtered.dat -d sb-graph-filtered.dat
-o macro_IIN.dat
```

The macro-IIN.dat is unweighted (i.e.,all weights are equal to 1) by default. It should be used as unweighted matrix of data just for qualitative purposes (i.e. to have a map visualization of all the more persistent interactions in the protein and their reciprocal location). Nevertheless, it is also possible to assign weights to edges of the obtained graphs by supplying a weighted adjacency matrix file, with the command line option `-w`. In this way the edges that are present in the macro-IIN graph are weighted according to the corresponding value in the weighted adjacency matrix provided with `-w`. For instance, one could use the interaction pseudo-energy map, which PyInteraph2 itself provides, as the weighted matrix. In the obtained output graph, only significant interactions would be considered and the respective edges would be weighted according to the side-chain interaction energy. Nonetheless, the script accepts any matrix with the same shape as the adjacency matrix of the original graphs as weight, making it possible to use any value as weight.

```
filter_graph -d hb-graph-filtered.dat -d hc-graph-filtered.dat -d sb-graph-filtered.dat
-o macro_IIN_w.dat -w kbp-graph.dat
```

The interaction pseudo-energy map is calculated as detailed in section 3.4.5

3.4.7 acPSN

In the example to calculate acPSN we will use the cutoffs suggested by the authors of the method (Kannan and Vishveshwara, 1999), i.e. a I_crit of 3.0 and a distance cutoff of 4.5 Å. We also discarded from the analysis, residues that contiguous in the sequence (setting `--acpsn-proxco 2`) as suggested in previous applications of the method.

```
pyinteraph -s pdbmovie_1.pdb -t ../../../3.filt_trjs/traj_prot_dt1000.xtc -r
pdbmovie_1.pdb -a --acpsn-co 4.5 --acpsn-imin 3 --ff-masses charmm27 --acpsn-proxco
2 --acpsn-nf-file normalization_factors.ini --acpsn-graph acpsn-graph.dat
```

3.5 Network analysis

Finally, network analysis on the graphs (be they individual IIN graphs, pseudo-energy graphs, or weighted macro-IINs) can be performed using the `graph_analysis`, `path_analysis` and `centrality_analysis` tools, according to the type of analyses requested. These tools are able to analyze any output graphs in the adjacency matrix format obtained by PyInteraph2 or even other graphs provided in the same format for which the user would like to carry out network analysis, such as residue-based contact maps or matrices of correlated motions. In the examples below we refer to each of these files as $file.dat The graph file contains no information about residue type or residue number; a reference PDB file can be supplied with the –r flag to provide such information. In this case, the graph nodes are labelled after the residue names and numbers found in the PDB file, in consecutive order (i.e. the first graph node corresponds to the first PDB residues and so on). If a PDB file is not provided, simple numerical indices will be used as node labels. It is advisable to keep the graph labels as a reference before starting the analysis, because they will be used in the output of the program, and expected as inputs in the path_analysis script.

3.5.1 Basic network analysis with graph_analysis

The example are provided for the cmPSN but can be extended to any PSNs or IINs.

```
cd 5.psn/pyinteraph2/cmPSN
```

`graph_analysis` allows the user to:

- Print out the graph node labels. If a reference structure is supplied with the –r flag, node names strictly follow residue names. In particular, they are referred to as 'X-Ya' (where X, Y and a are the chain ID, residue number and residue type, respectively). If the reference PDB file does not include a chain ID, the X will be replaced by 'SYSTEM'. This helps to easily identify residues in the output files and to provide residue labels for the input flags (i.e. for –s and –t in the path analysis procedure, for example). If no reference file is supplied, numbers are used instead. In our example the list of graph nodes can be provided by the following command line:

  graph_analysis −a $file.dat −r $reference.pdb

- Identify connected components. The general analysis for the connected components can be performed with -c flag. It provides as standard output the list of identified connected components along with the residues belonging to each one. Moreover, if the user provides an additional -cb flag together with an input reference structure (-r flag) an output PDB file will be written identical to the reference PDB, in which the B-factor column is replaced by the ID of the connected component to which the residue belongs. For instance, all residues having "1" as B-factor belong to the largest connected component (that has ID "1"), all the residues having B-factor "2" belong to the second-largest connected component, and so on. The reference command line for the connected component analysis is:

  ```
  graph_analysis -a cmpsn_all_filtered.dat -r pdbmovie_1.pdb -c -cb ccs.pdb
  ```

- Identify hubs. Calculation of hubs is performed with the -u flag. It allows calculating the number of edges for each node, and the nodes connected to at least *k* other nodes are included in the output. The value of *k* can be set with the command line option –k (we recommend setting this value equal to 3 or larger). If a reference input structure is used (-r flag), the option -ub can be added to save as output a PDB file identical to the reference one except for the B-factor column, which will contain the number of edges for each identified hub node, and 0 in all the other cases (nodes with less than *k* edges). The command line for the analysis of hubs is:

  ```
  graph_analysis -a cmpsn_all_filtered.dat -r pdbmovie_1.pdb -u -ub hubs.pdb -
  k 3
  ```

3.5.2 Analysis of paths of communication with path_analysis

path_analysis allows the user to:

- Calculate the shortest paths between any pair of nodes in the graph. Since multiple equally long shortest paths may exist for a given pair of nodes, all of them will be reported by `path_analysis`.
- We also support the ability to calculate all paths up to an arbitrary length. The maximum length is set with option -l.
- Calculate the metapath of the graph. The option -m is used to calculate the metapath, while options -e and -n are used to set the filtering thresholds for nodes and edges, respectively. By default, both filtering thresholds have a value of 0.1. The -g option can be used to select the minimum sequence distance. The -d option can be used to set the name of the metapath output file. This results in the creation of a DAT file containing an adjacency matrix representation of the metapath and a PDF file with a plot of the metaapath. The DAT file can be used for plotting the metapath in xPyder. An example command using filtering thresholds of 0.1 and a residue spacing of 3 to calculate the metapath is:

  ```
  path_analysis -i cmpsn_all_filtered.dat -r pdbmovie_1.pdb -m -g 3 -e 0.1 -n 0.1 -d metapath_hb
  ```

3.5.3 Centrality measures with centrality_analysis

`centrality_analysis` allows the user to:

- Calculate the centrality values for the following node based centrality measures:
  - The node and edge based centralities are written to different files. By default, the node based centralities are saved in a file called “centrality.txt” while edge based centralities are outputted in a file called “centrality_edge.txt”. These can be changed with the -o option.
- The node centrality file can be sorted based on a centrality value using the -b option followed by the centrality name and the edge centralities can similarly be sorted using the -d option.
- The -p option allows the user to additionally create pdb files for each node centrality measure where the B factor column is replaced with the centrality value. This can be used for plotting in PyMOL. The -m option allows the user to create .dat files containing the adjacency matrix of all the edge centrality values. This can be visualized using xPyder
- The -n option is used to normalize all centrality values to the range 0 to 1 where applicable. This is set to True by default. The -e option allows the use of endpoints when calculating shortest paths in closeness and betweenness centrality. This is set to False by default. The -x and -t options are used for the maximum number of iterations and tolerance values when calculating eigenvector centrality. By default they are set to 1e-06 and 100 respectively.
- An example command to calculate all node and edge based centrality measures:

```
centrality_analysis -i cmpsn_all_filtered.dat -r pdbmovie_1.pdb -c all -o centrality.csv
```

3.6 Visualizing and analyzing graphs in Cytoscape

Also this example is carried out on the cmPSN above.

```
dat2graphml -a cmpsn_all_filtered.dat -r pdbmovie_1.pdb -o  graph.graphml
```

- The <-a dat_file> is to give any previously generated PSN-stored dat file.
- The [-r reference_structure_file] allows to give a reference structure PDB file. This option is used to set node names. If it is not provided, auto-generated numbers are used to label node names.
- The [-o output_name] is used to define the output name of the graphml file to be generated. Its default value is set to “graph.graphml”.

The file `graph.graphml` can then be open with Cytoscape for further analyses.

References

(Brinda and Vishveshwara, 2005) Brinda, K. V., and Saraswathi Vishveshwara. "A network representation of protein structures: implications for protein stability." Biophysical journal 89.6 (2005): 4159-4170.

(Cohen et al., 2009) Cohen, Mati, Vladimir Potapov, and Gideon Schreiber. "Four distances between pairs of amino acids provide a precise description of their interaction." PLoS computational biology 5.8 (2009): e1000470.

(Fanelli and Felline, 2011) Fanelli, Francesca, and Angelo Felline. "Dimerization and ligand binding affect the structure network of A2A adenosine receptor." Biochimica Et Biophysica Acta (BBA)-Biomembranes 1808.5 (2011): 1256-1266.

(Fanelli et al., 2013) Fanelli, Francesca, Angelo Felline, and Francesco Raimondi. "Network analysis to uncover the structural communication in GPCRs." Methods in Cell Biology. Vol. 117. Academic Press, (2013): 43-61.

(Kannan and Vishveshwara, 1999) Kannan, N., and S. Vishveshwara. "Identification of side-chain clusters in protein structures by a graph spectral method." Journal of molecular biology 292.2 (1999): 441-464.

(Papaleo et al., 2014) Papaleo, Elena, et al. "Conformational changes and free energies in a proline isomerase." Journal of Chemical Theory and Computation 10.9 (2014): 4169-4174.

(Pasi et al., 2012) Pasi, Marco, et al. "xPyder: a PyMOL plugin to analyze coupled residues and their networks in protein structures." Journal of chemical information and modeling 52.7 (2012): 1865-1874.

(Potapov et al., 2010) Potapov, Vladimir, et al. "Protein structure modelling and evaluation based on a 4-distance description of side-chain interactions." Bmc Bioinformatics 11.1 (2010): 1-17.

(Seeber et al., 2011) Seeber, Michele, et al. "Wordom: a user-friendly program for the analysis of molecular structures, trajectories, and free energy surfaces." Journal of computational chemistry 32.6 (2011): 1183-1194.

(Tiberti et al., 2014) Tiberti, Matteo, et al. "PyInteraph: a framework for the analysis of interaction networks in structural ensembles of proteins." Journal of chemical information and modeling 54.5 (2014): 1537-1551.

(Viloria et al., 2017) Viloria, Juan Salamanca, et al. "An optimal distance cutoff for contact-based Protein Structure Networks using side-chain centers of mass." Scientific reports 7.1 (2017): 1-11.

(Vishveshwara et al., 2009) Vishveshwara, Saraswathi, Amit Ghosh, and Priti Hansia. "Intra and inter-molecular communications through protein structure network." Current Protein and Peptide Science 10.2 (2009): 146-160.