nature portfolio

https://doi.org/10.1038/s41596-021-00640-z

Supplementary information

Integrative structural modeling of macromolecular complexes using Assembline

In the format provided by the authors and unedited

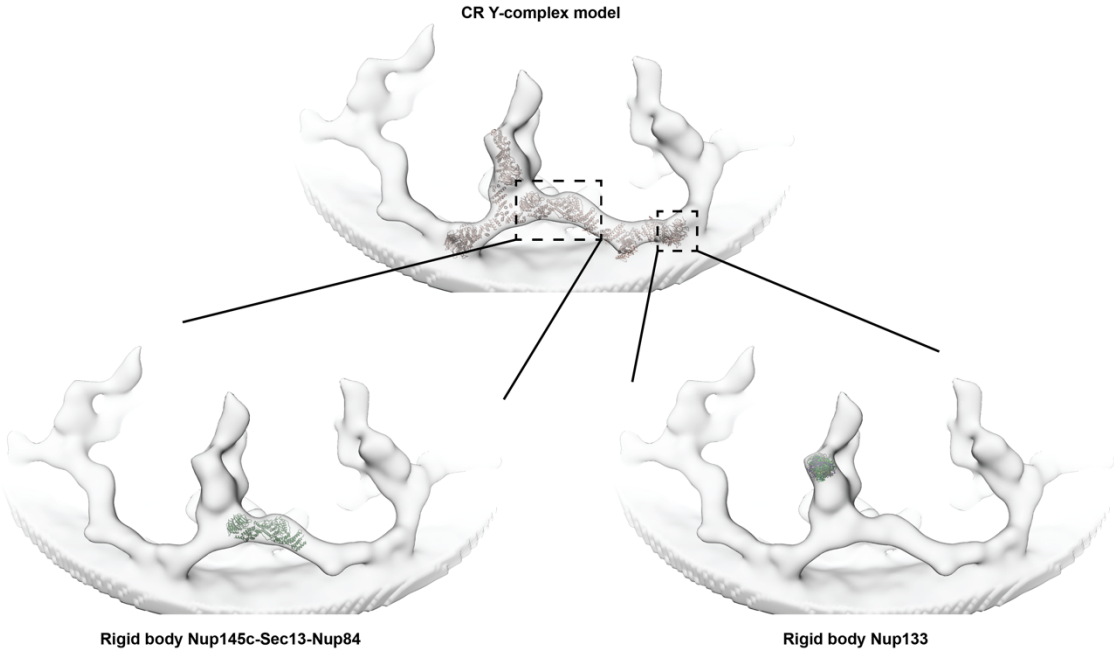
Supplementary information

Supplementary figures **1**

Supplementary Manual **5**
Assembline installation and usage manual. Detailed manual for Assembline protocol (online version: https://assembline.readthedocs.io/en/latest/#), which includes step-by-step explanations and suggestions on how to set up and run integrative modelling for any protein complex.

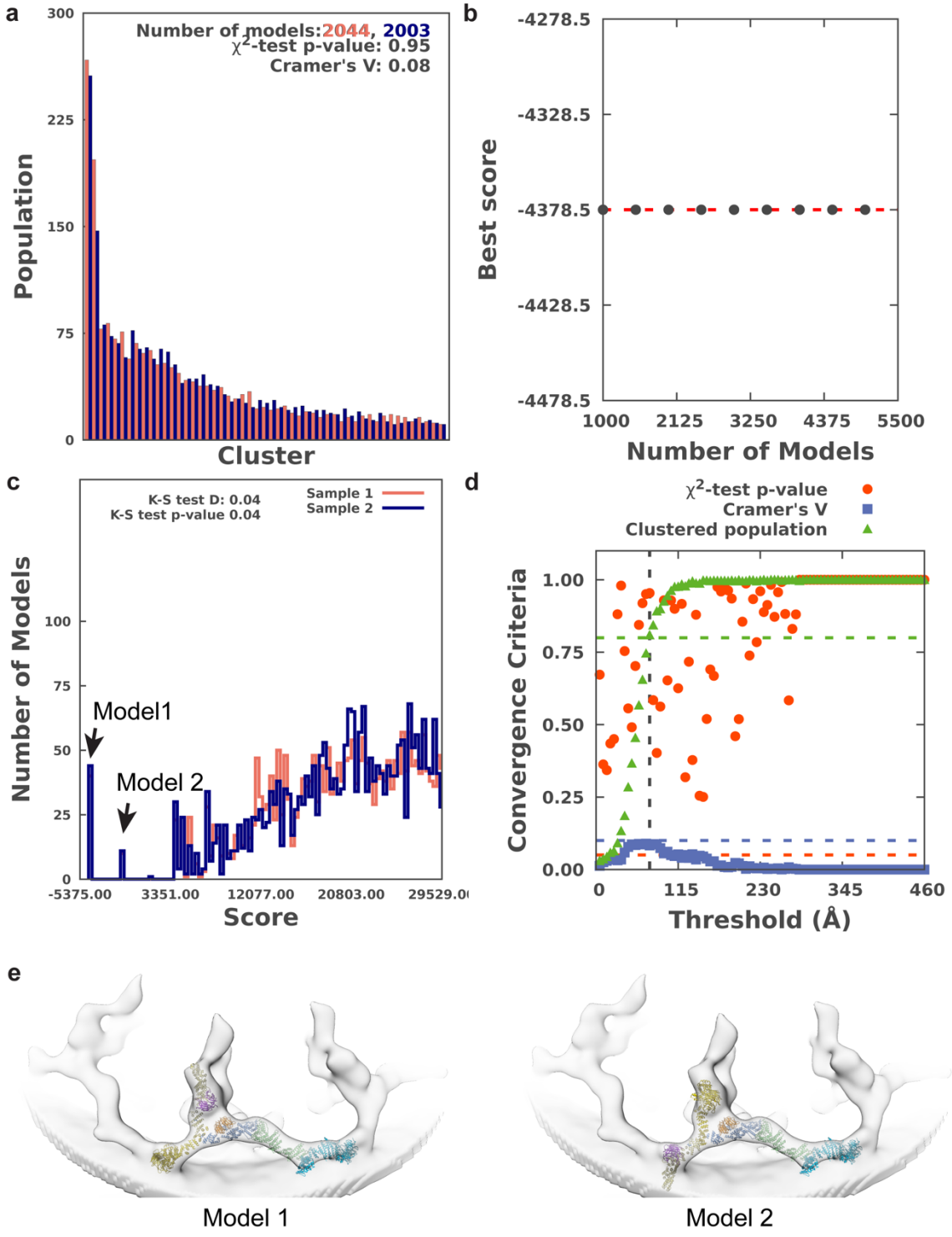
Supplementary Tutorial 1 **140**
S. cerevisiae nuclear pore complex modeling. Step-by-step tutorial including explanations for reproducing the ScNPC modelling based on in-cell cryo-ET maps as described in Allegretti et al.[6] (online version: https://scnpc-tutorial.readthedocs.io/en/latest/#).

Supplementary Tutorial 2 **218**
Elongator complex modeling. Step-by-step tutorial including explanations and suggestions for reproducing the Elongator complex modelling based on negative stain EM map and XL-MS data as described in Dauden et al.[28] (online version: https://elongator-tutorial.readthedocs.io/en/latest/index.html#).



Supplementary Figure 1

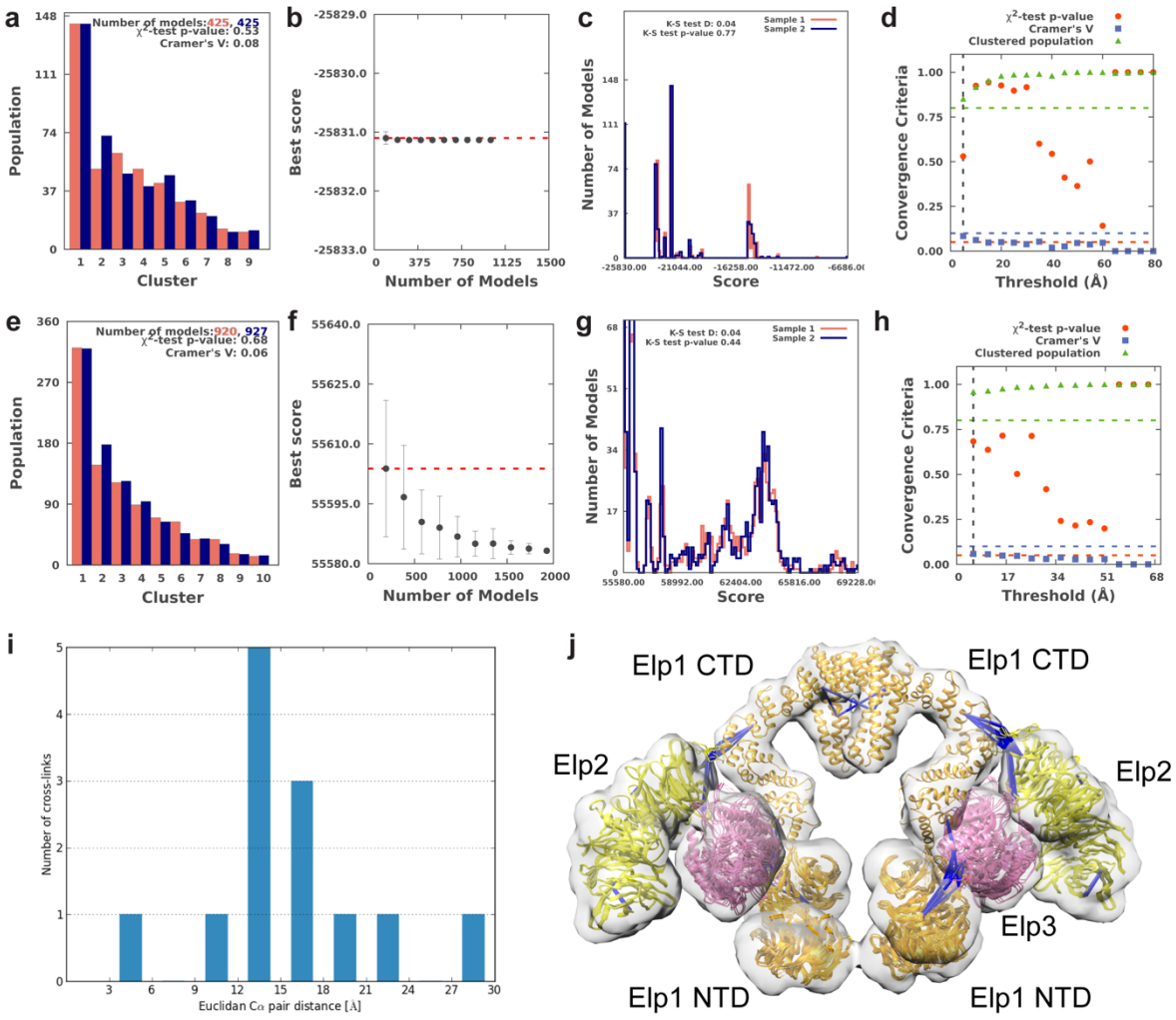
Examples of best-scoring fitted rigid bodies from fit libraries for ScNPC CR Y-complex. (Top) For comparison, the final integrative model for ScNPC CR Y-complex (brown cartoon) produced by Assembline is shown fitted in the ScNPC cryo-ET map (grey surface). (Bottom left) Representation of the best-scoring fit of the rigid body including domains of Nup145c-Sec13-Nup84 (green cartoon). This fit explains very well the local density features and is identical with the orientation of the equivalent rigid body that was used for the production of the final CR Y-complex model (top). (Bottom right) Representation of the five best-scoring fits of the rigid body including the N-terminal domain of Nup133 (multi-color cartoon). Although, the top fits were incorrectly localized at this stage, Assembline was still able to find the optimal location and orientation of this rigid body in the following integrative modelling steps.



Supplementary Figure 2

Results of the analysis step for the CR Y-complex of ScNPC. Assessment of the sampling convergence and exhaustiveness and estimation of the sampling and model precision for the global optimization step for the CR Y-complex of ScNPC. **a,** Population numbers of the obtained clusters in two random samples of models (red and blue bars) are similar. **b,** Convergence of the top score indicates that the scores do not improve with more models. **c**, Score distributions in the two random samples are similar, although not at the statistically significant level. Two distribution peaks with two groups of identical models that score clearly

better than the remaining models are indicated with arrows. **d,** Three statistical criteria define the sampling precision. For explanations of the tests, see the original publication Viswanath et al.[19]. Although the precision is low (75 Å), the sampling converged (**b**) and two outstanding models can be identified. The low precision can happen at the global optimization stage in situations when, in addition to the good scoring models, the modeling results include many poor models that do not fall into big clusters. Nevertheless, the convergence and repeating of identical top scoring models are sufficient to proceed to the refinement step. **e**, The two top scoring models indicated with arrows in **c** displayed in cartoon representation in the EM map. The top-scoring model (Model 1, left) fits better to the EM density and agrees with the organization of the equivalent subunits in human NPC.



Supplementary Figure 3

Results of the analysis step for the Elongator complex. a-h, Assessment of the sampling convergence and exhaustiveness and estimation of the sampling and model precision for the global optimization (a-d) and refinement (e-h) steps. **a and e,** Population numbers of the obtained clusters in two random samples of models (red and blue bars) are similar. **b and f,** Convergence of the top score indicates that the scores do not improve with more models. **c and g**, Score distributions in the two random samples are similar. **d and h,** Three statistical criteria define the sampling precision. For explanations of the tests, see the original publication Viswanath et al.[19]. **i**, Quantification of crosslink restraint violation, plotted using Xlink Analyzer interface. **h**, An ensemble of five top-scoring models from the largest cluster shown within the localization probability densities depicted as gray transparent surface. Crosslinks are indicated as blue bars (all satisfying the expected distance of 30 Å).

Assembline

Release 1.0

Vasileios Rantos, Kai Karius, Jan Kosinski

Sep 02, 2021

INSTALLATION

1 System requirements **3**

2 Installation with Anaconda **5**
- 2.1 Install dependencies . . . 5
- 2.2 Install Anaconda . . . 5
- 2.3 Create virtual Anaconda environment . . . 5
- 2.4 Install Python dependencies . . . 5
- 2.5 Install R . . . 6
- 2.6 Install Assembline . . . 6

3 For developers **7**
- 3.1 Clone our repositories: . . . 7
- 3.2 Clone PDBX and PDB-TOOLS . . . 7
- 3.3 Setup before running . . . 7
- 3.4 Explanations: . . . 8

4 Organize data **9**

5 JSON configuration file **11**
- 5.1 Creating JSON file . . . 11
- 5.2 Series and copies . . . 19
- 5.3 Selectors . . . 19
- 5.4 Paths . . . 19

6 Parameter file **21**
- 6.1 Creating the parameter file . . . 21
- 6.2 Parameters . . . 27

7 About fit libraries **31**

8 Set up **33**

9 Run **37**

10 Analyze fits **39**
- 10.1 Check if run correctly . . . 39
- 10.2 Generating fitted PDBs . . . 40

11 Adding precomputed fitting libraries to JSON **43**

12 Input structures **45**

13 Rigid bodies **47**

14 Selectors **51**

15 Calculating symmetry **53**
15.1 Method 1 . . . 53
15.2 Method 2 . . . 53

16 Defining symmetry **57**
16.1 Using a symmetry axis information . . . 57
16.2 Using transformation matrix . . . 58

17 Applying symmetry **59**

18 Symmetry constraints **61**

19 Symmetry restraints **63**

20 Excluded volume (steric) restraints **65**

21 Connectivity restraints **67**

22 EM restraints **69**
22.1 Precomputed EM fits . . . 69
22.2 FitRestraint . . . 69
22.3 EnvelopePenetrationRestraint . . . 70
22.4 ExcludeMapRestraint1 . . . 71
22.5 CloseToEnvelopeRestraint . . . 72
22.6 EnvelopeFitRestraint . . . 73

23 Crosslink restraints **75**
23.1 Loading crosslink data . . . 75
23.2 Defining the restraints . . . 75
23.3 Parameters . . . 75

24 Symmetry restraints and constraints **77**

25 Interaction restraints **79**

26 Distance restraints **81**

27 Similar orientation restraints **83**

28 Elastic network restraints **85**

29 Parsimonious states restraints **87**

30 Custom restraints **89**

31 Run the modelling **91**
31.1 Introduction . . . 91
31.2 A single run . . . 91
31.3 Multiple runs . . . 92
31.4 Assembline.py options . . . 93
31.5 Checking if everyhting runs correctly . . . 93

32 1. Global optimization **95**

33 2. Recombinations **97**

34 3. Refinement **99**
34.1 Refine the top-scoring models from previous modelling modes/runs . . . 99
34.2 Refine a single model . . . 99
34.3 Refine multiple models . . . 100

35 Running more **103**

36 Modify and re-run **105**
36.1 Typical adjustments that need to be done: . . . 105
36.2 Was the calculation quick? . . . 105

37 Check the logs **107**

38 Extract scores **109**
38.1 Plot histograms of all scores . . . 109

39 Visualize models and trajectories **111**
39.1 Create CIF format file for top models . . . 111
39.2 Create CIF format file for a specific model . . . 111
39.3 Rebuilding flexible beads . . . 112
39.4 Create PDB for top models . . . 112
39.5 Create separate PDB files for rigid bodies . . . 112
39.6 Other options . . . 112
39.7 Visualize the top model(s) in Chimera and Xlink Analyzer . . . 113
39.8 Visualize the optimization trajectory of the top model(s) in Chimera and Xlink Analyzer . . . 113

40 Quick test of convergence **115**

41 Sampling exhaustiveness and precision **117**

42 Glossary **121**

43 Frequently asked questions **123**

44 Troubleshooting **125**

45 Tips and tricks **127**
45.1 Speeding up calculations . . . 127

Index **129**

Assembline - Assembly line of macromolecular assemblies!

CHAPTER

ONE

SYSTEM REQUIREMENTS

Currently only UNIX-based systems supported. Should work both on Linux and OS X.

We recommend using a computer cluster but Assembline can be also run on a standalone workstation.

Warning: Some scripts might not work on OS X. All tests were performed with bash (i.e. not tested in other shells).

CHAPTER

TWO

INSTALLATION WITH ANACONDA

Installation should take less than 1 hour.

2.1 Install dependencies

1. UCSF Chimera 1.14 (https://www.cgl.ucsf.edu/chimera/download.html)

   `chimera` command must be available in your command line.

2. Xlink Analyzer plugin to Chimera, version 1.1 and higher (https://www.embl-hamburg.de/XlinkAnalyzer/XlinkAnalyzer.html)

   It is only needed on your local workstation for input preparation and analysis.

2.2 Install Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, which aims to simplify package management and deployment.

We will use Anaconda to create a "virtual environment" in which Assembline will be installed.

Download and install version for Python 3 from https://www.anaconda.com/distribution/

2.3 Create virtual Anaconda environment

```
conda create --name Assembline
```

2.4 Install Python dependencies

1. Activate ("enter") the environment:

   ```
   source activate Assembline
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline
   ```

2. Install Integrative Modeling Platform (IMP) version at least 2.14:

```
conda install -c salilab imp
```

3. Install other dependencies:

```
conda install scipy numpy scikit-learn matplotlib pandas
conda install -c salilab pyrmsd
conda install -c conda-forge hdbscan
```

4. [optional] Install Modeller. Only needed for building full atom representation of loops modelled as flexible beads

```
conda install -c salilab modeller
```

Follow the displayed instruction to edit `lib/modeller-9.25/modlib/modeller/config.py` inserting your Modeller license key

2.5 Install R

The easiest is to install R in your Anaconda environment:

```
conda install r-base r-fdrtool r-psych r-ggplot2 r-tidyr r-data.table
```

If for any reason you want to install R separately, install R from https://www.r-project.org/ and make sure:

- `Rscript` is available in your command line.
- The following R packages are installed:
  - fdrtool
  - psych
  - ggplot2
  - tidyr
  - data.table

2.6 Install Assembline

```
conda install -c kosinskilab assembline
```

For your information - this will also install some external dependencies:

- PDBX https://github.com/soedinglab/pdbx
- PDB_TOOLS https://github.com/haddocking/pdb-tools

Warning: Always activate the environment before using the software by:

```
source activate Assembline
```

or depending on your computer setup:

```
conda activate Assembline
```

CHAPTER

THREE

FOR DEVELOPERS

Create the environment and install all dependencies, but do not install Assembline from Anaconda.

Instead:

3.1 Clone our repositories:

In your `software_dir`:

```
git clone git@git.embl.de:kosinski/efitter.git
git clone git@git.embl.de:kosinski/pyxlinks.git
git clone git@git.embl.de:kosinski/pdb_utils.git
mkdir Assembline
cd Assembline
git clone git@git.embl.de:kosinski/imp_utils1.git
git clone git@git.embl.de:kosinski/SuperConfig.git
```

3.2 Clone PDBX and PDB-TOOLS

In your `software_dir`:

```
git clone https://github.com/soedinglab/pdbx.git
git clone https://github.com/haddocking/pdb-tools.git
```

3.3 Setup before running

After installation run sth like the below every time before using the pipeline:

```
module load Anaconda3
source activate Assembline
export PYTHONPATH=<your path>Assembline/SuperConfig:$PYTHONPATH
export PYTHONPATH=<your path>Assembline/imp_utils1:$PYTHONPATH
export PYTHONPATH=<your path>pyxlinks:$PYTHONPATH
export PYTHONPATH=<your path>situs_utils:$PYTHONPATH
export PYTHONPATH=<your path>pdb_utils:$PYTHONPATH
export PYTHONPATH=<your path>/pdb-tools/:$PYTHONPATH
export PYTHONPATH=<your path>/pdbx:$PYTHONPATH
```

of course replacing the path with your path.

And, use the full paths to the scripts:

- Assembline scripts are located in `Assembline/imp_utils1/scripts`
- Efitter scripts (those used for generating and analyzing fit libraries) are in `efitter/scripts`

3.4 Explanations:

- `git clone` - creates local copies of software in so-called "git repositories" e.g. at git.embl.de

CHAPTER

FOUR

ORGANIZE DATA

1. Create a new directory for your project

   Note: Note that for Assembline your files can be located in arbitrary locations on your disk but it is more convenient to have them in a single directory and use relative paths in the JSON configuration file.

2. Collect sequences of your subunits

   Prepare a single FASTA-formatted file with all sequences of your subunits. A subunit is a protein of your complex.

   For example:

```
>subunit1
MVEHDKSGSKRQELRSNMRNLITLNKGKFKPTASTAEGDEDDLSFTLLDSVFDTLSDSI
ISWRGDCDYFAVSSVEEVPDEDDETKSIKRRAFRVFSREGQLDSASEPVTGMEHQLSWK
EMKKGKHPSIVCEFPKSEFTSEVDSLRQVAFINDSIVGVLLDTDNLSRIALLDIQDITQ
RYKEAFIVCRTHRINLDILHDYAPELFIENLEVFINQIGRVDYLNLFISCLSEDDVTKT
HGLALYRYDSEKQNVIYNIYAKHLSSNQMYTDAAVAYEMLGKLKEAMGAYQSAKRWREA
RLIERLNQTKPDAVRVVEGLCRR
>subunit2
MVECITPEAIFIGANKQTQVSDIHKVKKIVAFGAGKTIALWDPIEPNNKGVYATLKGHE
LLSNKQYKFQIDDELRVGINFEALIMGHDDWISSLQWHESRLQLLAATADTSLMVWEPD
GEDDANEDDEEEEGGNKETPDITDPLSLLECPPMEDQLQRHLLWPEVEKLYGHGFEITC
RLRWSHLKRNGKLFLGVGSSDLSTRIYSLAYE
```

3. Collect structures of your subunits

   Create a directory with structures of your subunits in PDB format.

   The subunit chains can be organized in the PDB files in any way: a PDB file can contain multiple subunits, extra proteins not used in modeling, domains of a subunit can be separate or together. The JSON configuration file will take care of reading only what is needed.

   Warning: Make sure that the protein sequence and residue numbering in the PDB files correspond to the sequences in the FASTA file.

4. [OPTIONALLY] To simplify definition of rigid bodies later: Prepare your PDB file such that each PDB file corresponds to an anticipated rigid body (i.e. there is 1-to-1 mapping between the PDB files and your anticipated rigid bodies).

Read more about rigid bodies and alternative ways of preparing them here: *Rigid bodies*

5. Collect EM maps and put them into a subdirectory if you want to use EM restraints (read more *EM restraints*)
6. Collect CSV files with crosslinks in xQuest or XlinkAnalyzer format, if you want to use crosslink restraints (read more *Crosslink restraints*)
7. You may have the following directory structure at this point:

```
complexX/
    X_sequences.fasta
    in_pdbs/
        pdb1.pdb
        pdb2.pdb
        ...
    xlinks/
        some_name.csv
        ...
    EM/
        map1.mrc
        map2.mrc
        ...
```

CHAPTER

FIVE

JSON CONFIGURATION FILE

5.1 Creating JSON file

1. Create XlinkAnalyzer project file for your complex

   Note: XlinkAnalyzer is used here as a graphical interface for input preparation in Assembline.

   Does not matter if you do not have crosslinks - we will use XlinkAnalyzer to prepare the input file for modeling.

   1. Add all subunits using Xlink Analyzer graphical interface
   2. Assign unique chain ID and color to every subunit
   3. Optionally, define domains within subunits. You could then refer to these domains in the *Selectors*
   4. Add sequences using Setup panel in Xlink Analyzer, map the sequences to names of subunits using the Map button
   5. Add crosslinks if available, map the crosslinked protein names to names of subunits using the Map button

2. Make a copy of the Xlinkanalyzer project file. This copy will be next manually edited to add modeling directives. E.g. Copy `XlinkAnalyzer_project.json` as `X_config.json`
3. Open `X_config.json` in a text editor

   Note: The project file is in so-called JSON format

   While it may look difficult to edit at the first time, it is actually quite OK with a proper editor (and a bit of practice ;-)

   We recommend to use a good editor such as:

   - SublimeText
   - Atom

   At this point, the JSON has the following format:

```
{
    "data": [
        {
            "some xlink definition 1"
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

4. [Optionally] Define `series`

   Serie is a group of related subunit copies e.g. related by symmetry.

   In Assembline configuration, series are used to:

   - create multiple copies of the same subunit
   - define symmetry acting on the subunits within the serie
   - define symmetry between two different series, if any

   Series can be specified in the top-level `series` block, at the same level as `data`, `subunits` etc.:

```
{
    "series": [

        "series specifications go here"

    ],

    "data": [
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

   Specification of a single serie:

```
{
    "name": "Name of the serie",
    "subunit": "Subunit instance for this Serie",
    "mode": "[optional] 'auto' or 'input', default: 'input'
             a parameter used in imp_utils1 to define whether PMI copies (
↪'input') or clones ('auto') should be created",
    "cell_count": "[optional] How many elements in the Series, default: 1",
    "tr3d": "[optional] Name of the transformation relating Subunits in␣
↪this Serie, as defined in the symmetry config below",
    "inipos": "[optional] 'input' or name of a symmetrical transformation␣
↪in config",
    "ref_serie": "[optional] Reference Serie. If ini_pos is a␣
↪transformation, imp_utils1 will transform
                  each copy in this serie relatively to the ref_serie by␣
↪that transformation",
    "states": "[optional] List of state indices this serie should act on.␣
↪Not sure if it's working."
}
```

Example:

```
{
    "series": [
        {
            "name": "2fold",
            "subunit": "Elp1",
            "mode": "input",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        },
        {
            "name": "2fold",
            "subunit": "Elp2",
            "mode": "auto",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        },
        {
            "name": "2fold",
            "subunit": "Elp3",
            "mode": "auto",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        }
    ]
}
```

defines three series, one for each of the three subunits Elp1, Elp2, Elp3, two copies per subunit as defined by `cell_count` (six molecules total).

`"mode": "input"` and `"mode": "auto"` define how molecules will be created behind the scenes.

`auto` would be typically used if you have structure for one subunit copy and you want the system to auto-generate the remaining copies a `clones` of the first. If unsure, keep `input`.

Users familiar with PMI: `auto` would build clones, `input` would build copies.

Here, we keep `input` for Elp1 subunit, as one of the input PDB structures is a dimer comprising fragments of both copies of Elp1. We keep `auto` for Elp2 and Elp3, as for them, we have structures of one copy only and we let the system to generate clones of the the first copy.

`tr3d` points to a name we gave to a transformation matrix for the 2-fold symmetry, which we define below.

`"inipos": "input"` defines the initial position. If `input` the positions will be as in the input PDB files. `inipos` can be set to a name of another `series` (see the example below why)

- Examples:

Note: In some examples provided in the tutorials or our published work you may see "series" defined in a different place, under the "symmetry" block. That is an old way, both are supported for backward compatibility.

---

5. [If applicable] Add symmetry information

   Symmetry can be specified in the top-level `symmetry` block, at the same level as `symmetries`, `data`, `subunits` etc.:

```
{

    "symmetry": {

        "sym_tr3ds": "<specification of transformation matrices>",

        "apply_symmetry": "<specification of subunits or regions of␣
↪subunits on which the symmetry is acting>"

    },

    "series": [

        "series specifications go here"

    ],

    "data": [
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
```

(continues on next page)

(continued from previous page)

```
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

Within the symmetry block you specify:

- `sym_tr3ds` - transformation matrices for each symmetry axis within the complex, and give each transformation a unique name. You can specify multiple symmetries if applicable.

  Read *Defining symmetry* for instructions.

- `apply_symmetry` - subunits or regions of subunits on which the symmetry is acting. This means you may have one symmetry specification for one domain of a subunit, and another symmetry or no symmetry at all for another domain.

  Read *Applying symmetry* for instructions.

6. Define input PDB files and rigid bodies

   as specified here: *Input structures*

   and add it to the `data` block as below:

```
{

    "symmetry": {

        "sym_tr3ds": "<specification of transformation matrices>",

        "apply_symmetry": "<specification of subunits or regions of␣
↪subunits on which the symmetry is acting>"

    },

    "series": [

        "series specifications go here"

    ],

    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [

                    "PDB file specifications"
            ]
        },

        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

7. [Optionally] Define custom rigid bodies.

   By default, rigid bodies are created based on blocks in the `pdb_files` specification above.

   If you want to define rigid bodies differently (there are some special situation you may want to do it) you can define custom rigid bodies

   as specified here: *Rigid bodies*

   and add it to the `data` block as below:

```
{

    "rigid_bodies": {
        "Rigid bodies file specifications"
    },

    "symmetry": {

        "sym_tr3ds": "<specification of transformation matrices>",

        "apply_symmetry": "<specification of subunits or regions of␣
↪subunits on which the symmetry is acting>"

    },

    "series": [

        "series specifications go here"

    ],

    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [

                    "PDB file specifications"
            ]
        },

        {
```

(continues on next page)

(continued from previous page)

```
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

8. For *1. Global optimization* step, compute fitting libraries and add them to the JSON file as specified here:

   About fit libraries

9. Add other restraints:

   Restraints are specified as blocks in the `data` block:

```
{

    "rigid_bodies": {
        "Rigid bodies file specifications"
    },

    "symmetry": {

        "sym_tr3ds": "<specification of transformation matrices>",

        "apply_symmetry": "<specification of subunits or regions of␣
↪subunits on which the symmetry is acting>"

    },

    "series": [

        "series specifications go here"

    ],

    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [

                    "PDB file specifications"
            ]
        },
```

(continues on next page)

(continued from previous page)

```
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        },

        {
            "a restraint"
        },

        {
            "another restraint"
        },

        {
            "and so on"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "1.1.1"
}
```

You can define the following restraints

EM restraints

Excluded volume (steric) restraints

Connectivity restraints

Symmetry restraints and constraints

Interaction restraints

Distance restraints

Similar orientation restraints

Elastic network restraints

Parsimonious states restraints

Custom restraints (e.g. original IMP restraints and your own implementations)

5.2 Series and copies

As shown above, **Series** is a group of related subunit copies e.g. related by symmetry.

In Assembline configuration, series are used to:

- create multiple copies of the same subunit
- define symmetry acting on the subunits within the series
- define symmetry between two different series, if any

Series have names and can be referred to by this name within selectors.

Each series is a group of subunit copies: **Copies** of subunits, numbered 0, 1, 2. For example, you may have a subunit of the nuclear pore complex in 16 copies arranged in two rings, 8 copies each. You could then define two series, for each ring, ring1 and ring2, each with copies 0, 1, 2, 3, 4, 5, 6, 7. Then, the first copy of ring1 would selected by ring1 and copy index 0. The first copy of the ring2 would be ring2 and, again, copy 0.

5.3 Selectors

Selectors are collections of keywords used to "select" parts of the system and act on them. For example, you can select a part of the system and define as rigid body, add to an EM restraint, or other restraints.

See here how to define *Selectors*

5.4 Paths

You can use either absolute or relative paths in your JSON project file. Both have advantages and disadvantages. The relative paths allow to move your project between different computers but cause some trouble when working on the output (not a big deal, but you need to use some extra option for scripts to point to the original directory with the data). The absolute paths do not need the extra options and just work, but you cannot the move the projects between computers without modification.

CHAPTER

SIX

PARAMETER FILE

The **parameter file** file defines modeling protocol, scoring functions, output parameters and some restraints.

The parameter file is written in Python programming language, but really, no programming required!

6.1 Creating the parameter file

1. Define the protocol:

   For example, for the *1. Global optimization*

   ```
   protocol = 'denovo_MC-SA'
   ```

   See parameters below for available protocols and guidelines for *1. Global optimization*, *2. Recombinations*, *3. Refinement*

2. Define Monte Carlo Simulated Annealing schedule:

   For example:

   ```
   SA_schedule = [
       (50000,   10000),
       (10000,   10000),
       (5000,    10000),
       (1000,    50000),
       (100,     50000)
   ]
   ```

   SA_schedule is a list of pairs. Each pair defines a Monte Carlo temperature and number of steps.

   The Monte Carlo Simulated Annealing with run the specified number of steps for the first temperature, then switch to the second temperature in the list and so on.

   Note: How to select the temperatures and number of steps?

   Unfortunately, there is no absolute way. For this, one usually first runs a small number of runs, and evaluates the output to select.

   The **temperature** would be selected to be similar in order to the obtained scores (e.g. if scores are in the 100,000 range, the temperature could be 10,000-100,000 range too) and based on the acceptance rates of accepted moves (listed in the output log files). If the acceptance rates are low, you may want to increase the temperatures.

You can always calculate the Metropolis probability of accepting a move given your score scales. For example, if you score moved up from 100,000 to 110,000 (difference 10,000) you can calculate the probability of accepting this move by this equation:

$$p = e^{-\Delta Score/kT}$$

substituting `k=1`, `T=<your temperature>`, `deltaScore=10,000`

The **number of steps** can be adjusted after evaluating the convergence of the initial runs. Also, the number of steps is always a compromise between the time needed for convergence and available computational resources.

---

3. Optionally, define an initial optimization:

```
do_ini_opt = True
ini_opt_SA_schedule = [
    (1000, 1000)
]
```

The initial optimization can be used for example to equilibrate the system with different restraints than in the main optimization.

4. Optionally, adjust log files and frequency of saving logs and frames:

```
traj_frame_period = 10
print_frame_period = traj_frame_period

print_log_scores_to_files = True
print_log_scores_to_files_frame_period = 10

print_total_score_to_files = True
print_total_score_to_files_frame_period = 10
```

5. Adjust representation resolutions for your molecules:

   For example, to define two resolutions, 1- and 10-residues per bead:

```
struct_resolutions = [1,10]
```

6. Decide whether to define rigid bodies based in the *Input structures* definition in the *JSON configuration file*:

```
add_rbs_from_pdbs = True
```

7. Set weight for *Connectivity restraints* and whether they should be applied only to the first copy of each series (should be True for symmetry **constrained** complexes):

```
connectivity_restraint_weight = 1.0
conn_first_copy_only = True
```

8. Add symmetry constraints and/or restraints?:

   For symmetry **constraints**:

```
add_symmetry_constraints = True
```

For symmetry **constraints**:

```
add_symmetry_restraints = True
```

9. Optionally, set parameters for cross-link restraints:

```
add_xlink_restraints = True
x_min_ld_score = 25
x_weight = 100.0
x_xlink_distance = 30
x_xlink_score_type='HarmonicUpperBound'
```

10. For *1. Global optimization* and *2. Recombinations*, set weight for "discrete restraints":

    The descrete restraints is just a name used for the restraint derived from the *Adding precomputed fitting libraries to JSON*

```
discrete_restraints_weight=10000 # weight for the restraint derived from␣
↪the fit libraries
```

11. Define excluded volume (steric) restraints:

```
ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    },
    {
        'name': 'ev_restraints_highres',
        'weight': 10,
        'repr_resolution': 1
    }
]
```

    In this block you define a list of possible excluded volume restraints. Each gets a custom name, weight and resolution of the representation it is assigned. You decide which will be used later in the scoring functions.

    The representation resolution will match the closest existing, e.g. if you have mixed representation resolution, 10 and 1 for rigid bodies and 1 for flexible beads, and specify `repr_resolution` of 10, it will be matched to the resolution 10 of rigid bodies and 1 of flexible beads.

12. Define scoring functions:

    In the example below, two scoring functions are defined, `score_func_ini_opt` and `score_func_lowres`. These are arbitrary names, choose them to your liking. Each scoring function has a list of `restraints`, listing names defined above and any other restraints defined in the JSON.

    Names of some restraints are predefined, such as:

    - discrete_restraints - restraints derived from the pre-defined positions, e.g. precomputed libraries of fits
    - conn_restraints - connectivity restraints
    - xlink_restraints - cross-link restraints

Other restraints bare names defined above (e.g. `ev_restraints_lowres` and `ev_restraints_highres` defined above) or in the :doc:json.

```
scoring_functions = {
    'score_func_lowres': {
        'restraints': [
            'discrete_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
    },
    'score_func_highres': {
        'restraints': [
            'discrete_restraints',
            'xlink_restraints',
            'conn_restraints',
            'ev_restraints_highres'
        ]
    }
}
```

In the example above, two scoring functions are defined. One uses the low resolution excluded volume restraint and does not use crosslinks. The second uses high resolution representation and turns on crosslinks.

13. Define which scoring function should be used for each modeling stage:

```
score_func_ini_opt = 'score_func_lowres'
score_func = 'score_func_highres'
```

The `score_func_ini_opt` parameter defines which of the scoring functions defined above should be used for the **initial** optimization.

The `score_func` parameter defines which of the scoring functions defined above should be used for the **main** optimization.

14. Finally, define running parameters for modeling (multiprocessing, cluster):

For example, for multiprocessor workstation of 8 cores, just define:

```
ntasks = 8
```

For cluster using Slurm queuing system:

```
cluster_submission_command = 'sbatch'
from string import Template
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=00:30:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
```

(continues on next page)

(continued from previous page)

```
srun hostname

$cmd

wait
""")
```

In summary, the above parameter file looks like this:

```
protocol = 'denovo_MC-SA'

SA_schedule = [
    (50000,   10000),
    (10000,   10000),
    (5000,    10000),
    (1000,    50000),
    (100,     50000)
]

do_ini_opt = True
ini_opt_SA_schedule = [
    (1000, 1000)
]

traj_frame_period = 10
print_frame_period = traj_frame_period

print_log_scores_to_files = True
print_log_scores_to_files_frame_period = 10

print_total_score_to_files = True
print_total_score_to_files_frame_period = 10

struct_resolutions = [1,10]

add_rbs_from_pdbs = True

connectivity_restraint_weight = 1.0
conn_first_copy_only = True

add_symmetry_constraints = True

add_xlink_restraints = True
x_min_ld_score = 25
x_weight = 100.0
x_xlink_distance = 30
x_xlink_score_type='HarmonicUpperBound'

discrete_restraints_weight=10000 # weight for the restraint derived from the␣
↪fit libraries

ev_restraints = [
```

(continues on next page)

(continued from previous page)

```
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    },
    {
        'name': 'ev_restraints_highres',
        'weight': 10,
        'repr_resolution': 1
    }
]


scoring_functions = {
    'score_func_lowres': {
        'restraints': [
            'discrete_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
    },
    'score_func_highres': {
        'restraints': [
            'discrete_restraints',
            'xlink_restraints',
            'conn_restraints',
            'ev_restraints_highres'
        ]
    }
}

score_func_ini_opt = 'score_func_lowres'
score_func = 'score_func_highres'

cluster_submission_command = 'sbatch'
from string import Template
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=00:30:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
srun hostname

$cmd

wait
""")
```

6.2 Parameters

protocol Modeling protocol. Options:

- denovo_MC-SA - Monte Carlo Simulated Annealing global optimization moving rigid bodies according to pre-computed position libraries, if they are defined for the rigid bodies, otherwise moving with random rotations and translations. Flexible beads are moved through Monte Carlo with random rotations and translations.
- denovo_MC-SA-CG - as denovo_MC-SA, but flexible beads are moved using Conjugate Gradient optimization
- refine - both rigid bodies and flexible beads are moved with random rotations and translations, pre-computed libraries are ignored. Flexible beads are moved using Conjugate Gradient optimization.
- all_combinations - generates all combinations of positions from pre-computed position libraries
- custom - a function with a custom protocol based on a user-defined function custom_protocol() defined in params.

, default: **denovo_MC-SA**

do_ini_opt Perform initial optimization using the score_func_ini_opt scoring function?, default: **False**

SA_schedule Simulated Annealing schedule. List of (temperature, number of steps) pairs., default: **[(30000, 1000), (2000, 1000), (1000, 1000)]**

before_opt_fn A Python function with code that should be executed before optimization, default: **None**

number_of_cg_steps_for_flex_beads Number of Conjugate Gradient steps per round in denovo_MC-SA-CG and refine protocols, default: **100**

stop_on_convergence Whether to stop the current stage of Simulated Annealing when converged, working well only for optimizations with high number of steps., default: **False**

no_frames_for_convergence Number of frames for evaluating convergence when stop_on_convergence is set to True, default: **1000**

print_frame_period Print every Nth frame ID in progress reporting, default: **10**

traj_frame_period Save every Nth frame to the trajectory output file, default: **10**

print_total_score_to_files Save frame total scores to log files?, default: **False**

print_total_score_to_files_frame_period Print total score to log files every Nth frame, default: **10**

print_log_scores_to_files Save frame individual restraint scores to log files?, default: **False**

print_log_scores_to_files_frame_period Print individual restraint scores to log files every Nth frame, default: **10**

struct_resolutions Resolutions of bead representations., default: **[1, 10]**

add_missing Add missing regions as flexible beads? Either False or a list of selectors., default: **False**

missing_resolution Representation resolution for the missing regions (if add_missing is specified), default: **1**

add_rbs_from_pdbs Define rigid bodies automatically based on pdb_files specification in JSON?, default: **True**

ca_only Read only Calpha from PDB structures?, default: **True**

add_connectivity_restraints Add domain and bead connectivity restraints?, default: **True**

connectivity_restraint_weight Connectivity restraint weight, default: **1.0**

max_conn_gap Number of missing residues of the missing atomic region above which the restraint will not be added, default: **None**

connectivity_restraint_k Connectivity restraint spring constant k., default: **10.0**

conn_reweight_fn A function that accepts the following parameters: mol - PMI molecule object next_resi_idx - residue of the next rigid body or bead prev_resi_idx - residue of the previous rigid body or bead connectivity_restraint_weight - default weight passed to add_connectivity_restraints, default: **None**

conn_first_copy_only Whether to add the restraints only for the first copy of the molecule (useful for symmetrical assemblies), default: **False**

ca_ca_connectivity_scale Scale the average CA-CA distance of 3.8 to account for that it is unlikely that the linker is fully stretched, default: 0.526 (count 2A per peptide bond), default: **0.526**

ev_restraints Specification of excluded volume (clash score) restraint. Parameters: 'name': a custom name 'weight': weight 'repr_resolution': which representation resolution to use for this restraint 'copies': which molecule copies are included in this restraint 'distance_cutoff': distance cutoff for non-bonded list 'slack': slack for non-bonded lists Read more about distance_cutoff and slack: https://integrativemodeling.org/2.14.0/doc/ref/classIMP_1_1container_1_1ClosePairContainer.html#aa7b183795bd28ab268e5e84a5ad0cd99, default: **[{'name': 'ev_restraints', 'weight': 1.0, 'repr_resolution': 10, 'copies': None, 'distance_cutoff': 0.0, 'slack': 5.0}]**

add_xlink_restraints Add crosslink restraints?, default: **False**

x_xlink_score_type A type of crosslink restraint.

Options:

'HarmonicUpperBound' - 0 below x_xlink_distance, harmonic above, distance calculated between centers of the beads

'HarmonicUpperBoundSphereDistancePairScore' - 0 below x_xlink_distance, harmonic above, distance calculated between surfaces of the beads

'XlinkScore' - 0 below x_xlink_distance, 1 above

'LogHarmonic' - A log harmonic potential with the maximum at x_xlink_distance

'CombinedHarmonic': - harmonic above distance of 35 A and log harmonic with the maximum at x_xlink_distance below 35 A

, default: **HarmonicUpperBound**

x_min_ld_score Only crosslinks with the confidence score above this threshold will be used as restraints. The score must be defined in "score" column of crosslink CSV files (see also Xlink Analyzer documentation), default: **30.0**

x_weight Weight of crosslink restraints, default: **1.0**

x_xlink_distance Target or maximual crosslink distance (depending on the implementation), default: **30.0**

x_k Spring constant for the harmonic potential, default: **1.0**

x_inter_xlink_scale Multiply the weight of inter-molecule crosslinks by this value, default: **1.0**

x_first_copy_only Apply crosslinks only to the first molecule copy of each series, default: **False**

x_skip_pair_fn A Python function to skip specific crosslinks. Arguments: p1, p2, component1, component2, xlink Return True to skip the crosslink, default: **None**

x_log_filename Optional log file name for printing more information about added and skipped crosslinks, default: **None**

x_score_weighting Scale crosslink weights by their score, default: **False**

xlink_reweight_fn A custom Python function to scale crosslink weights. Arguments: xlink, weight Return final weight (float), default: **None**

x_random_sample_fraction Take this random fraction of crosslinks for modeling, default: **1.0**

add_symmetry_constraints Add symmetry constraints?, default: **False**

add_symmetry_restraints Add symmetry restraints?, default: **False**

symmetry_restraints_weight Weight of symmetry restraints, default: **1.0**

add_parsimonious_states_restraints Add parsimonious states restraints?, default: **False**

parsimonious_states_weight Weight, default: **1.0**

parsimonious_states_distance_threshold Distance threshold for elastic network restraining the states, default: **0.0**

parsimonious_states_exclude_rbs Python function to exclude selected rigid bodies from the restraint. Arguments: IMP's rigid body object Return: True if the rigid body should be excluded., default: Some default function

parsimonious_states_representation_resolution Representation resolution for this restraint, default: **10**

parsimonious_states_restrain_rb_transformations Restraint rigid body transformations instead of using elastic network, default: **True**

create_custom_restraints Python function to create custom restraints. Arguments: imp_utils1.MultiRepresentation class. Return dictionary mapping restraint names to list of restraints, default: **None**

discrete_restraints_weight Weight for restraints derived from the pre-defined positions, e.g. precomputed libraries of fits., default: **1.0**

discrete_mover_weight_score_fn help message, default: **None**

scoring_functions A collection of scoring functions, default: **{}**

score_func Name of the scoring functions in the scoring_functions collection to be used for the main optimization, default: **None**

score_func_for_CG Name of the scoring functions in the scoring_functions collection to be used for conjugate gradient steps. Only restraints that have implemented derivative calculation can be used for Conjugate Gradient optimization., default: **None**

score_func_ini_opt Name of the scoring functions in the scoring_functions collection to be used for the initial optimization, default: **None**

score_func_preconditioned_mc Name of the scoring functions in the scoring_functions collection to be used for the preconditioned Monte Carlo, default: **None**

add_custom_movers A Python function to add custom Monte Carlo movers, default: **None**

custom_preprocessing A Python function with code that should be executed before modeling protocols are initiated and after adding and setting all restraints. Arguments: imp_utils1.MultiRepresentation class., default: **None**

rb_max_rot Maximal rotation of a rigid body in a single Monte Carlo move, in radians, default: **0.2**

rb_max_trans Maximal translation of a rigid body in a single Monte Carlo move, in Angstroms, default: **2**

beads_max_trans Maximal translation of a flexible bead in a single Monte Carlo move, in Angstroms, default: **1**

randomize_initial_positions Randomize initial positions of all particles?, default: **False**

randomize_initial_positions_remove_clashes Randomize initial positions of all particles and run short optimization to try removing steric clashes?, default: **False**

get_movers_for_main_opt Python function defining movers for the main optimization, override for custom implementations, default: Some default function

get_movers_for_ini_opt Python function defining movers for the initial optimization, override for custom implementations, default: Some default function

get_movers_for_refine Python function defining movers for the refinement, override for custom implementations, default: Some default function

debug Print debug messages?, default: **False**

ntasks Number of tasks to run for multiprocessor runs, default: **1**

cluster_submission_command Command to run the cluster submission script, default: **None**

run_script_templ A template for running the jobs on a computer cluster. Make sure your template includes $ntasks, ${prefix}, $outdir, and $cmd, default:

```
"""

    #!/bin/bash
    #
    #SBATCH --ntasks=$ntasks
    #SBATCH --mem-per-cpu=2000
    #SBATCH --job-name=${prefix}fit
    #SBATCH --time=5-00:00:00
    #SBATCH -e $outdir/logs/${prefix}_err.txt
    #SBATCH -o $outdir/logs/${prefix}_out.txt

    echo "Running on:"
    srun hostname

    $cmd

    wait #necessary when ntasks > 1 and cmd are run in the background, otherwise␣
↪job may end before the background processes end

"""
```

CHAPTER

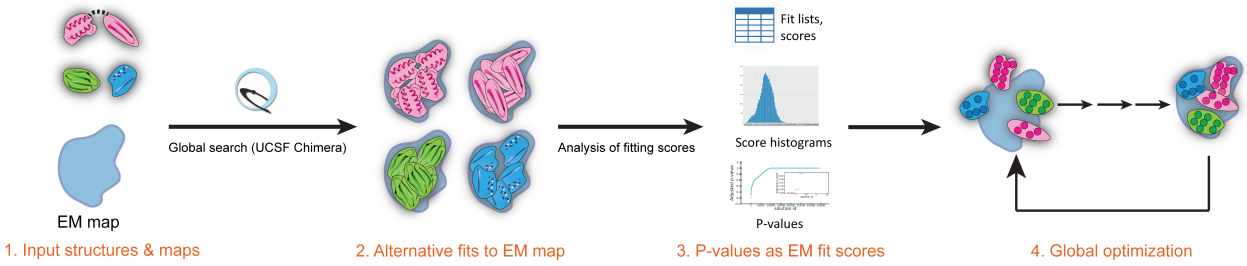
SEVEN

ABOUT FIT LIBRARIES

Fit libraries are lists of possible positions (non-redundant fits) of your input structures in the EM map. The positions are used in the *1. Global optimization* step to generate a large number of combinations of the fits through a Monte Carlo integrative modelling procedure. Each position (fit) has its associated score and p-value, which are used as EM restraints during the modelling.

The fit libraries are typically generated by fitting every input structure into the EM map separately, a procedure handled by the `Efitter` pipeline of Assembline.

Note: This part of the Assembline pipeline (`efitter`) will generate all requested libraries per input structure in a single run (automated) according to user's predefined parameters (see *Set up*, and following figure).



CHAPTER

EIGHT

SET UP

To run the calculation of fit libraries (Assembline sub-pipeline called efitter) you need a parameter file in Python language format that specifies:

- input structures to fit
- EM map (or EM maps)
- fitting parameters
- optionally, options for execution on a computer cluster

Two template parameter files are provided in the `Assembline/doc/templates` directory which can be found and inspected from the git repo Assembline. Explanations for the different fitting parameters are provided below and in the two template parameter files named:

- `fit_params_template_cluster.py` - for a computer cluster (Slurm-based cluster)
- `fit_params_template_multicore.py` - for a workstation, preferably multicore

Example parameter file with explanations:

```
#Some import lines required
from efitter import Map
from string import Template


method='chimera' # Fitting method, currently only 'chimera' supported
dry_run = False # Dry run would only print commands it is going to run
run_crashed_only = False # Only run the jobs that have not delivered output
master_outdir = 'path_to_output_directory' # relative path to the output, it will be␣
↪created in the running directory

# For experimental maps specify: the paths to the maps, the density threshold at which␣
↪the fitting to the experimental map should be performed, the resolution of the␣
↪experimental map in Angstrom
MAPS = [
    Map('/path_to_your_maps/map1.mrc', threshold=0.01, resolution=5),
    Map('/path_to_your_maps/map2.mrc', threshold=0.02, resolution=5),

]

models_dir = '/path_to_your_structure(s)' # directory with the structure files

#the actual structure files in the above directory
```

(continues on next page)

(continued from previous page)

```
PDB_FILES = [
    'pdb_file_name1.pdb',
    'pdb_file_name2.pdb',
    'pdb_file_name3.pdb'
]

CA_only = False # Calculate the fitting scores using Calpha atoms only?
backbone_only = False # Calculate the fitting scores using backbone atoms only?
move_to_center = True # Move the PDB structure to the center of the map prior to fitting?

# Each element of fitmap_args is a dictionary specifying parameters for a run
# If multiple dictionaries are specified,
# the script will run a separate run for each dictionary for each map-structure␣
↪combination.
# E.g. if two maps, three structures, and two parameters dictionaries are specified,
# the script will run 2 x 3 x 2 = 12 runs.
fitmap_args = [
    # We suggest to run a small test run with search 100 first
    {
        'template': Template("""
            map $map
            map_threshold $threshold
            fitmap_args resolution $resolution metric cam envelope true search 100␣
↪placement sr clusterAngle 3 clusterShift 3.0 radius 200 inside .30
            saveFiles False
            """),
        'config_prefix': 'test' # some name informative of the fitmap_args parameters
        },
    # Paramters for main runs (https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/
↪fitmap.html)
    {
        'template': Template("""
            map $map
            map_threshold $threshold
            fitmap_args resolution $resolution metric cam envelope true search 100000␣
↪placement sr clusterAngle 3 clusterShift 3.0 radius 200 inside .30
            saveFiles False
            """),
        'config_prefix': 'search100000_metric_cam_inside0.3' # some name informative of␣
↪the fitmap_args parameters
        },
    # Run with alternative "inside" parameter
    {
        'template': Template("""
            map $map
            map_threshold $threshold
            fitmap_args resolution $resolution metric cam envelope true search 100000␣
↪placement sr clusterAngle 3 clusterShift 3.0 radius 200 inside .60
            saveFiles False
            """),
        'config_prefix': 'search100000_metric_cam_inside0.6' # some name informative of␣
↪the fitmap_args parameters
```

(continues on next page)

(continued from previous page)

```
        }
]

# If necessary, edit the below template of the script that will run individual fitting␣
↪runs

run_script_templ = Template("""#!/bin/bash
#
echo $job_name
$cmd &>$pdb_outdir/log&
""")
```

The provided template for a cluster includes an example configuration for the Slurm queuing engine. If your cluster uses a different queuing engine, modify the respective parameters in `fit_params_template_multicore.py`:

```
cluster_submission_command = 'sbatch'
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --job-name=$job_name
#SBATCH --time=1-00:00:00
#SBATCH --error $pdb_outdir/log_err.txt
#SBATCH --output $pdb_outdir/log_out.txt
#SBATCH --mem=1000

$cmd
""")
```

Note: It is important that your edited template includes `$job_name`, `$pdb_outdir`, and `$cmd`. They will be auto-filled by the program.

CHAPTER

NINE

RUN

1. Run the fitting

```
fit.py efitter_params.py
```

The fitting runs will execute in the background and take from a few dozens of minutes to several hours depending on the case.

In the output, you should have got the following directory structure:

```
master_outdir/ # Directory specified with "master_outdir" parameter in␣
↪params.py
    config_prefix1/ # named after config_prefix specifications in fitmap_
↪args in params.py
        map1.mrc/   # named after the filenames of the EM maps used
            map1.mrc  # symbolic link to the reference map
            pdb_file_name1.pdb/     # named after the pdb file names used␣
↪for fitting
                solutions.csv   # the list of solutions and their scores
                log_err.txt     # standard error log
                log_out.txt     # standeard output log
                run.sh          # sbatch script used for running the job
                ori_pdb.pdb     # symbolic link to the original query file
                map1.mrc        # symbolic link to the reference map
            pdb_file_name2.pdb/
            pdb_file_name3.pdb/
            config.txt          # A config file for fitting, saved FYI.
        map2.mrc/
    config_prefix2/
    config_prefix3/
```

Note: The fitting is complete when each of the `.pdb` directories contains `solutions.csv` file.

Inspect the `log_out.txt` files for status and `log_err.txt` for error messages.

2. Upon completion, calculate p-values:

```
genpval.py <fitting directory/master_outdir>
```

This should create additional files in each `.pdb` directory:

```
Rplots.pdf
solutions_pvalues.csv
```

The `solutions_pvalues.csv` is crucial for the global optimization step.

CHAPTER

TEN

ANALYZE FITS

10.1 Check if run correctly

In the output, you should have got the following directory structure:

```
master_outdir/ # Directory specified with "master_outdir" parameter in params.py
    config_prefix1/ # named after config_prefix specifications in fitmap_args in params.
↪py
        map1.mrc/   # named after the filenames of the EM maps used
            map1.mrc  # symbolic link to the reference map
            pdb_file_name1.pdb/     # named after the pdb file names used for fitting
                solutions.csv   # the list of solutions and their scores
                solutions_pvalues.csv # the list of solutions and their scores including␣
↪pvalues THIS IS THE FILE NEEDED FOR THE NEXT STEP
                log_err.txt     # standard error log
                log_out.txt     # standeard output log
                run.sh          # sbatch script used for running the job
                ori_pdb.pdb     # symbolic link to the original query file
                map1.mrc        # symbolic link to the reference map
                Rplots.pdf      # some statistics from the pvalue calculation
            pdb_file_name2.pdb/
            pdb_file_name3.pdb/
            config.txt          # A config file for fitting, saved FYI.
        map2.mrc/
    config_prefix2/
    config_prefix3/
```

Check if you obtained these files, in particular the `solutions_pvalues.csv` file.

Note that ori_pdb.pdb and map1.mrc files are symbolic links to the original files. If for any reason those links are broken or do not exist, you can re-generate them by running the `fit.py` script with ``–update_links `` option:

```
fit.py --update_links efitter_params.py
```

10.2 Generating fitted PDBs

Although not necessary for the subsequent modelling, you can generate top fits as PDB files visualization.

You may for example see that for some structures you obtain significant p-values in `solutions_pvalues.csv` file, and upon visual inspection, decide that you want to restrict the fits to these significant fits using `max_positions` parameter when *Adding precomputed fitting libraries to JSON*.

10.2.1 Method 1

Enter the results directory for the given map:

```
cd master_outdir/config_prefix1/map1.mrc/
```

Generate PDBs for multiple structures and/or maps into a single directory:

```
cd fit_cam_inside0.3_fa_10000
genPDBs_many.py [options] outdir <solutions_list>
```

Example case-scenario following:

Generate fits for all maps: Enter the "parameters-set" fit directory like `search100000_metric_cam_inside0.3_radius500/` and run:

```
genPDBs_many.py -n5 top5 */*/solutions.csv
```

This will generate a directory `top5` with subdirectories for each map, and top 5 fits for each map.

Generate fits for a specific map: Enter the directory for specific map like `search100000_metric_cam_inside0.3_radius500/P_negstain_01.mrc` and run:

```
genPDBs_many.py -n5 top5 */solutions.csv
```

This will generate a directory `top5` with top 5 fits for each map

10.2.2 Method 2

A bash command like the following will iterate through all output directories and generate 10 fits there:

```
for f in `ls fits_chimera/fit_cam_inside0.3_Big/nr_8_norm_m3i_filt.no_membrane.
↪mrc/* | grep ":" | perl -p -e "s/\://"`;
    do
        cd $f;
        genPDBs.py -n 10 solutions.csv ori_pdb.pdb
        cd /g/kosinski/kosinski/NPC/fitting/Chlamy
    done
```

10.2.3 Method 3 (“Manually”)

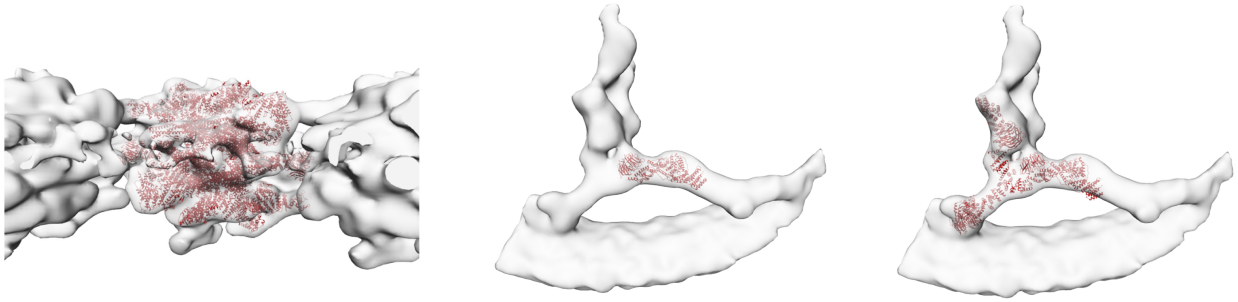
Enter the directory for the given run:

```
cd master_outdir/config_prefix1/map1.mrc/pdb_file_name1.pdb/
```

Generate the PDBs for a specified number of top fits:

```
genPDBs.py -n 5 solutions.csv <path to the pdb files>/pdb_file_name1.pdb
```

Visualize the pdbs with your map in Chimera (example figure following)



Examples of best-scoring fits (S. cerevisiae nuclear pore complex) displayed in respective EM maps

CHAPTER

ELEVEN

ADDING PRECOMPUTED FITTING LIBRARIES TO JSON

1. Fit every structure using our Efitter pipeline, calculate `solutions_pvalues.csv` files using `genpval`, as described in the previous steps.
2. For each structure, add the following "positions" lines to the `pdb_files` (if `add_rigid_bodies_from_pdb` set to `True`) or `rigid_bodies` specifications (see *Input structures* and *Rigid bodies*)

```
{
    "components": [
        {
        "name": "required, keep same as subunit name",
        "subunit": "required, keep same as subunit name",
        "chain_id": "required, which chain from the PDB file should be read␣
↪as input for this subunit",
        "filename": "yourpath.pdb"
        }
    ],
    "positions": "<path>/solutions_pvalues.csv",
    "positions_type": "chimera - optional, chimera is default",
    "max_positions": "optional, by default all fits are read",
    "positions_score": "log_BH_adjusted_pvalues_one_tailed is default",
}
```

For example:

```
{
    "components": [
        {
        "name": "<subunit_name1>",
        "subunit": "<subunit_name1>",
        "chain_id": "E",
        "filename": "yourpath.pdb"
        }
    ],
    "positions": "<path>/solutions_pvalues.csv",
    "positions_type": "chimera",
    "max_positions": 10000,
    "positions_score": "log_BH_adjusted_pvalues_one_tailed"
}
```

> **Warning:** If you do have the `rigid_body` block specification and `add_rigid_bodies_from_pdb` set to `False` in `params.py`, the positions MUST be added there, not in `pdb_files`, otherwise they won't have an effect.

3. If there is 1-to-1 mapping between PDB files and rigid bodies AND you want to take the same number of fits for every structure:
    1. Fit every structure using our fitting pipeline, calculate `solutions_pvalues.csv` files (as above)
    2. Add the following "alternative_positions" definition to the JSON in the data section:

```
{
    "active": true,
    "type": "alternative_positions",
    "positions_type": "chimera",
    "add_from_dir": "<fitting master dir>/<parameter set fitting dir>/
↪<map dir>",
    "max": 10000,
    "score": "log_BH_adjusted_pvalues_one_tailed"
}
```

CHAPTER

TWELVE

INPUT STRUCTURES

- PDB files are defined by appropriately formatted objects (`{components: ...}`, see below) that define which chain of the given PDB file maps to which subunit
- The PDB file specification is added to `data` object in the *JSON configuration file*
- A PDB file can contain parts of different subunits, for example, in the example below, `yourpath.pdb` contains two subunits: `<subunit_name1>` and `< subunit_name2>` with chains `E` and `F`, respectively.
- Different parts of subunits can be present in different PDB files and under different chain ids, for example, in the example below, `<subunit_name2>` has one part in `yourpath.pdb` under chain `F`, and another part in `yourpath2.pdb` under chain `B`
- by default, the rigid bodies are created based on this PDB file specification. All PDB fragments grouped into the same `components` list will be grouped into a rigid body. In the example below, chain `X` and `Y` of `yourpath3.pdb` will be in the same rigid body and all other chains will be seperate rigid bodies.
- Specification:

```
{
    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [
                        {
                            "components": [
                                {
                                "name": "<subunit_name1>",
                                "subunit": "<subunit_name1>",
                                "chain_id": "E",
                                "filename": "yourpath.pdb"
                                }
                            ]
                        },
                        {
                            "components": [
                                {
                                "name": "<subunit_name2>",
                                "subunit": "<subunit_name2>",
                                "chain_id": "F",
                                "filename": "yourpath.pdb"
                                }
                            ]
```

(continues on next page)

(continued from previous page)

```
                        },
                        {
                            "components": [
                                {
                                "name": "<subunit_name2>",
                                "subunit": "<subunit_name2>",
                                "chain_id": "B",
                                "filename": "yourpath2.pdb"
                                }
                            ]
                        },
                        {
                            "components": [
                                {
                                "name": "<subunit_name3>",
                                "subunit": "<subunit_name3>",
                                "chain_id": "X",
                                "filename": "yourpath3.pdb"
                                },
                                {
                                "name": "<subunit_name4>",
                                "subunit": "<subunit_name4>",
                                "chain_id": "Y",
                                "filename": "yourpath3.pdb"
                                }
                            ]
                        }
                    ]
            }
        ]
}
```

CHAPTER

THIRTEEN

RIGID BODIES

By default, rigid bodies will be defined automatically based on the specifications in the PDB files *Input structures*. If you want to define rigid bodies differently to your PDB files, you can do it in a separate `rigid_bodies` block at the top level of the *JSON configuration file*.

Warning: For `rigid_bodies` to have an effect, you have to set `add_rigid_bodies = False` in the *Parameter file*

The specification is the same as for PDB files:

```
{
    "rigid_bodies": [
        {
            "components": [
                "list of selectors"
            ]
        },
        {
            "components": [
                "list of selectors"
            ]
        },

        "and so on"
    ]
}
```

You can also use the `foreach_serie` and `foreach_copy` keywords here:

```
{
    "rigid_bodies": [
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                "list of selectors"
            ]
        },
        {
            "foreach_serie": true,
```

(continues on next page)

(continued from previous page)

```
            "foreach_copy": true,
            "components": [
                "list of selectors"
            ]
        },

        "and so on"
    ]
}
```

An example:

```
{
    "rigid_bodies": [
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp1_1", "subunit": "Elp1", "domain": "propeller1"}
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp1_1", "subunit": "Elp1", "domain": "propeller2"}
            ]
        },
        {
            "foreach_serie": true,
            "components": [
                { "name": "Elp1_1", "subunit": "Elp1", "domain": "CTD", "copies
↪": [0, 1]}
            ],
            "freeze": true
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp2_1", "subunit": "Elp2",
                    "resi_ranges": [
                        [1, 239],
                        [257, 788]
                    ]
                }
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
```

(continues on next page)

(continued from previous page)

```
                { "name": "Elp3_4", "subunit": "Elp3", "domain": "helix4"}
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp3_5", "subunit": "Elp3", "domain": "struct"}
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp3_6", "subunit": "Elp3", "domain": "flex_helix"}
            ]
        }
    ]
}
```

CHAPTER

FOURTEEN

SELECTORS

Selectors are collections of keywords used to "select" parts of the system and act on them. For example, you can select a part of the system and define as rigid body, add to an EM restraint, or other restraints.

Syntax / list of allowed keywords:

```
{
    "subunit": "....",
    "state": "....",
    "states": "....",
    "domain": "....",
    "serie": "....",
    "copies": "....",
    "chain_id": "....",
    "chainIds": "....",
    "filename": "....",
    "resi_ranges": "....",
}
```

Additionally, *Input structures* and *Rigid bodies* specifications allow using the special `foreach_serie` and `foreach_copie` modifiers. They allow to apply selectors to all series in the sytem and/or to all copies within the matching series.

Examples:

Residue 925 of the Nup120 subunit in the series named NR_1

```
{"subunit": "Nup120", "serie": "NR_1", "resi_ranges": [925]}
```

The membrane_binding domain of the Copy 0 of the Nup155 subunit in the IR_cyt_outer ring (one of the four sub-rings of the inner ring of the nuclear pore complex)

```
{
    "name": "Nup155",
    "subunit": "Nup155",
    "domain": "membrane_binding",
    "copies": [
        0
    ],
    "serie": "IR_cyt_outer"
}
```

All copies of Nup107 subunit in the CR series

```
{
    "name": "Nup107",
    "subunit": "Nup107",
    "serie": "CR"
}
```

Define the same PDB file for each copy of Elp1 in all series

```
{
    "foreach_serie": true,
    "foreach_copy": true,
    "components": [
        { "name": "Elp1",
        "subunit": "Elp1",
        "domain": "propeller1",
        "filename": "in_pdbs/Elp1_NTD_1st_propeller.pdb"
        }
    ]
}
```

Define a rigid body for propeller1 domain for each copy of Elp1 in all series

```
{
    "rigid_bodies": [
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp1_1", "subunit": "Elp1", "domain": "propeller1"}
            ]
        }
    ]
}
```

CHAPTER

FIFTEEN

CALCULATING SYMMETRY

For symmetry restraints and constraints, symmetry axes or transformation matrices need to be specified in *JSON configuration file*.

If you don't have the symmetry axes or matrices yet, here are some alternative methods for obtaining that information using UCSF Chimera based on an EM map. Other EM processing software should be also able to report that information.

15.1 Method 1

If you know the symmetry axis goes along x, y or z axis, you still need the "symmetry point", i.e. the point through which the axis passes through. It might go through (0,0,0) but not necessarily. To detect the symmetry and find the center you can do:

1. Open the EM map in Chimera.
2. Open command line: Menu -> General Controls -> Command Line
3. In the command line, execute `measure symmetry #0` command.

   The software should print sth like "Symmetry map_name.mrc: C2, center 112 112 113".

   The center values are in "grid coordinates" so multiply each value by map voxel size to obtain the coordinates of the center in Angstrom. Enter the resulting values in *Defining symmetry*

15.2 Method 2

The manual way. A lot of fun.

1. Open the EM map in Chimera.
2. Open the same map again.
3. Open a IDLE window where the information about transformations will be printed: Menu Tools -> General Controls -> IDLE
4. Open command line: Menu -> General Controls -> Command Line
5. Open Model Panel (Menu Favorites -> Model Panel).
6. In the Model Panel, de-activate the first map from motion using the check box in the A column
7. Using a mouse, rotate the second EM map around the symmetry axis for around one rotation step, so the maps roughly overlap.

   E.g. for a 2-fold symmetry, rotate 180 degrees, for a 8-fold symmetry rotate 45 degrees.

8. Activate the first model back for motion by clicking the check box in in the A column
9. Fit the rotated map to the first map using *Fit in Map tool <https://www.cgl.ucsf.edu/chimera/docs/ContributedSoftware/fitmaps/fitma* (Tools -> Volume Data -> Fit in Map)
10. In the command line, execute `measure rotation #1 #0` command.
11. Switch to the IDLE window. A log that looks like the following should appear:

```
Position of elp123_150616_cl3dK4nofirstround3class1symC2_reformat.mrc (#0)␣
↪relative to elp123_150616_cl3dK4nofirstround3class1symC2_reformat.mrc (
↪#1) coordinates:
Matrix rotation and translation
  -0.99999999  -0.00009462  -0.00011784 492.84569581
   0.00009464  -0.99999999  -0.00014030 492.81640318
  -0.00011782  -0.00014031   0.99999998   0.06655213
Axis  -0.00005892  -0.00007015   1.00000000
Axis point 246.41119075 246.41986373   0.00000000
Rotation angle (degrees) 179.99457796
Shift along axis   0.00294274
```

12. Use the values define the rotation matrix in the JSON in *Defining symmetry*:
    1. Use the values in the "Axis" and "Axis point" parts to define the symmetry in the JSON:

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "2fold",
                "axis": [0, 0, 1],
                "center": [246.41119075, 246.41986373, 0.00000000],
                "type": "C2"

            }

        ]
    }
}
```

    2. Use the values in the "Matrix rotation and translation" part to define the rotation matrix in the JSON.

       The first three columns correspond to rotation and the last column to the translation. Copy the two rows of the rotation columns as a single row of 9 elements in the JSON configuration. Copy the column of the translation transposing to a row of 3 elements in the JSON configuration.

       For the transformation above, you should end up with sth like this

```
{
  "symmetry": {
      "sym_tr3ds": [

          {
              "name": "8-fold",
              "rot": [-0.99999999, -0.00009462, -0.00011784,0.
↪00009464, -0.99999999, -0.00014030, -0.00011782, -0.00014031, 0.
↪99999998],
```

(continues on next page)

(continued from previous page)

```
            "trans": [492.84569581, 492.81640318, 0.06655213]
        }

    ]
  }
}
```

Insert this code in the JSON file, as specified in *JSON configuration file*.

CHAPTER

SIXTEEN

DEFINING SYMMETRY

Symmetry can be defined in "symmetry" blocks at the top level of the *JSON configuration file* in lists of "sym_tr3ds". Multiple symmetries can be defined and used simultaneously.

16.1 Using a symmetry axis information

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "2fold",
                "axis": "vector of the symmetry axis",
                "center": "symmetry point (through which the above symmetry axis is␣
↪passing through)",
                "type": "cyclic symmetry type like C2, C3, C8 and so on"

            }

        ]
    }
}
```

For example:

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "2fold",
                "axis": [0, 0, -1],
                "center": [246.39112398, 246.41114644, 248.600000],
                "type": "C2"

            }

        ]
```

(continues on next page)

(continued from previous page)

```
    }
}
```

How to get this transformation information? Follow *Calculating symmetry* instruction how to do it based on an EM map.

16.2 Using transformation matrix

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "a unique custom name",
                "rot": "list of 9 values for rotation, 3 for each row of the symmetry␣
↪matrix",
                "trans": "list of 3 values for translation"
            }

        ]
    }
}
```

For example:

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "2-fold",
                "rot": [0.70712087, 0.70709269, 0.00003928, -0.70709269, 0.70712087, 0.
↪00000394, 0.00003057, -0.00002499, 1.00000000],
                "trans": [-411.56570152, 993.5285317, 0.09136987]
            }

        ]
    }
}
```

How to get this transformation matrices? Follow *Calculating symmetry* instruction how to do it based on an EM map.

CHAPTER

SEVENTEEN

APPLYING SYMMETRY

Note: This is not necessary if you want to apply symmetry to entire subunits. In that case it is enough you specify `tr3d` arguments in the series, setting them to the names of transformation matrices defined above.

Specification of `apply_symmetry`:

```
"apply_symmetry": [
    {
        "sym": // name of the symmetry transformation specified in "symmetry",
        "restraint_type": "symmetry_restraint" or "symmetry_constraint",
        "selectors": [
            // list of selectors of regions that should be restrained.
            // the number of selectors must equal the order of the symmetry
            // so for 2-fold - two selectors, 8-fold - eight selectors
        ]
    }
]
```

For example:

```
"apply_symmetry": [
    {
        "sym": "2fold",
        "restraint_type": "symmetry_restraint",
        "selectors": [
            {"subunit": "EccB", "serie": "2fold", "resi_ranges": [[85, 260],␣
↪[345,400], [408,490]], "copies": [0]},
            {"subunit": "EccB", "serie": "2fold", "resi_ranges": [[85, 260],␣
↪[345,400], [408,490]], "copies": [1]}
        ]
    }
]
```

will restrain only specified residue ranges of the copy 0 with the copy 1 of EccB subunit.

CHAPTER

EIGHTEEN

SYMMETRY CONSTRAINTS

Symmetry constraints can be used to enforce symmetry of a complex. In contrast to symmetry restraints, the constraints cannot be violated and enforce perfect symmetry. The modelling protocols comprising Assembline will also use constraints to speed up calculations - only the reference particle from each constraint set will be moved and the remaining particles will be automatically set to follow the symmetry. Constraints are also faster to calculate than restraints.

So, whenever you are fine with enforcing perfect symmetry - use constraints.

To use constraints:

1. Set `add_symmetry_constraints = True` in `params.py`
2. Define symmetry axes transformations as in *Defining symmetry*
3. Define which components (subunits or specific regions of a subunit) should be constrained as in the *Applying symmetry*

Note: You can use both symmetry constraints and restraints, applying constraints to some subunits or regions, and restraints to others!

CHAPTER

NINETEEN

SYMMETRY RESTRAINTS

Symmetry restraints can be used to favor symmetry of a complex. In contrast to symmetry constraints, the restraints can be violated and only favor symmetry, with "strength" dependent on their user-defined weight.

They can be used when expect some deviations from a perfect symmetry.

To use restraints:

1. Set `add_symmetry_restraints = True` in `params.py`
2. Define symmetry axes transformations as in *Defining symmetry*
3. Define which components (subunits or specific regions of a subunit) should be restrained as in the *Applying symmetry*
4. Add `sym_restraints` to the scoring function in `params.py`, for example:

```
scoring_functions = {
    'score_func_lowres': {
        'restraints': [
            'xlink_restraints',
            'conn_restraints',
            'ev_restraints_lowres',
            'FitRestraint',
            'sym_restraints'
        ]
    }
}
```

Note: You can use both symmetry constraints and restraints, applying constraints to some subunits or regions, and restraints to others!

CHAPTER

TWENTY

EXCLUDED VOLUME (STERIC) RESTRAINTS

Excluded volume restraints, or steric restraints, prevent overlap between molecules.

The excluded volume restraints are defined in the *Parameter file* through blocks like this:

```
ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    },
    {
        'name': 'ev_restraints_highres',
        'weight': 10,
        'repr_resolution': 1
    }
]
```

In this block you define a list of possible excluded volume restraints. Each gets a custom name, weight and resolution of the representation it is assigned. You decide which will be used later in the definition of scoring functions (see *Parameter file*).

The representation resolution will match the closest existing, e.g. if you have mixed representation resolution, 10 and 1 for rigid bodies and 1 for flexible beads, and specify `repr_resolution` of 10, it will be matched to the resolution 10 of rigid bodies and 1 of flexible beads.

Parameters:

name a custom name

weight weight

repr_resolution which representation resolution to use for this restraint

copies which molecule copies are included in this restraint

distance_cutoff distance cutoff for non-bonded list

slack slack for non-bonded lists

Read more about distance_cutoff and slack: https://integrativemodeling.org/2.14.0/doc/ref/classIMP_1_1container_1_1ClosePairContainer.html#aa7b183795bd28ab268e5e84a5ad0cd99,

CHAPTER

TWENTYONE

CONNECTIVITY RESTRAINTS

Connectivity restraints are used to restrain distance between:

- consecutive single-residue beads (e.g. when representation resolution of 1 is used)
- two consecutive domains of a single protein separated by a linker of non-modelled/missing residues

The restraint is implemented as a harmonic distance restraint with a target distance equal to:

- 3.8 Angstrom for single-residue beads
- distance proportional to the number of missing residues between connected consecutive domains

The connectivity restraints are defined in the *Parameter file* automatically. If you want to use them, add `conn_restraints` to the scoring functions (see *Parameter file*).

Parameters (to be defined in the *Parameter file*):

connectivity_restraint_weight Connectivity restraint weight, default: **1.0**

max_conn_gap Number of missing residues of the missing atomic region above which the restraint will not be added, default: **None**

connectivity_restraint_k Connectivity restraint spring constant k., default: **10.0**

conn_reweight_fn A function that accepts the following parameters: mol - PMI molecule object next_resi_idx - residue of the next rigid body or bead prev_resi_idx - residue of the previous rigid body or bead connectivity_restraint_weight - default weight passed to add_connectivity_restraints, default: **None**

conn_first_copy_only Whether to add the restraints only for the first copy of the molecule (useful for symmetrical assemblies), default: **False**

ca_ca_connectivity_scale Scale the average CA-CA distance of 3.8 to account for that it is unlikely that the linker is fully stretched, default: 0.526 (count 2A per peptide bond), default: **0.526** Use 1.0 for no scaling at all.

CHAPTER

TWENTYTWO

EM RESTRAINTS

22.1 Precomputed EM fits

The restraints calculated from *pre-calculated fit libraries*.

They are created automatically if the fit libraries are defined as *positions* in the *JSON configuration file* and named `discrete_restraints`.

If you want to use them, add `discrete_restraints` to your scoring functions.

Parameters:

discrete_restraints_weight Weight for restraints derived from the pre-defined positions, e.g. precomputed libraries of fits., default: **1.0**

22.2 FitRestraint

The standard cross-correlation-based EM restraints implemented using FitRestraint from IMP

It can be defined in the *JSON configuration file* using blocks like this:

```
{
    "active": true,
    "type": "em_map",
    "name": "FitRestraint",
    "em_restraint_type": "FitRestraint",
    "filename": "EM_data/emd_4151_binned.mrc",
    "voxel_size": 6.6,
    "resolution": 25,
    "weight": 1000,
    "first_copy_only": false,
    "repr_resolution": 10,
    "optimized_components": [
        {"name": "Elp1", "subunit": "Elp1"},
        {"name": "Elp2", "subunit": "Elp2"},
        {"name": "Elp3", "subunit": "Elp3"}

    ]
}
```

Parameters

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

type must be: em_map

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

em_restraint_type must be: FitRestraint

filename relative or absolute path to the map in the MRC format

voxel_size Pixel/voxel size of the map in Angstrom

resolution Resolution for the EM map simulated from the model for cross-correlation calculation. Can be e.g. approximately the resolution of your map.

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

optimized_components A list of *Selectors* defining molecules or parts to which this restraint should be applied to.

22.3 EnvelopePenetrationRestraint

A restraint promoting particles to be covered by an EM map envelope at a defined density threshold, implemented using EnvelopePenetrationRestraint from IMP

It can be defined in the *JSON configuration file* using blocks like this:

```
{
    "active": true,
    "type": "em_map",
    "name": "Elys_in_difference_density",
    "em_restraint_type": "EnvelopePenetrationRestraint",
    "filename": "EM/map.mrc",
    "threshold": 0.001,
    "voxel_size": 6.9,
    "resolution": 20,
    "weight": 1000,
    "first_copy_only": true,
    "repr_resolution": 10,
    "optimized_components": [
        {
            "name": "Elys",
            "subunit": "Elys"
        }
    ]
}
```

Parameters

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

type must be: em_map

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

em_restraint_type must be: EnvelopePenetrationRestraint

threshold Density threshold to define the EM envelope

filename relative or absolute path to the map in the MRC format

voxel_size Pixel/voxel size of the map in Angstrom

resolution Resolution for the EM map simulated from the model for cross-correlation calculation. Can be e.g. approximately the resolution of your map.

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

optimized_components A list of *Selectors* defining molecules or parts to which this restraint should be applied to.

22.4 ExcludeMapRestraint1

A restraint preventing particle penetration of an EM density. Can used for example to prevent penetration of a segmented membrane density or density segment known to be occupied by other subunits.

Yes, the name with 1 at the end ;-)

```
{
    "active": true,
    "type": "em_map",
    "name": "exclude_membrane",
    "em_restraint_type": "ExcludeMapRestraint1",
    "filename": "../EM/Membrane_final_resampled_no_neg.mrc",
    "threshold": 0.02,
    "voxel_size": 6.9,
    "resolution": 20,
    "weight": 100000,
    "first_copy_only": false,
    "repr_resolution": 10,
    "optimized_components": [
        {
            "name": "Nup120",
            "subunit": "Nup120",
            "copies": [0,1]
        },
        {
            "name": "Nup189c",
            "subunit": "Nup189c",
            "copies": [0,1]
        }
    ]
}
```

Parameters

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

type must be: em_map

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

em_restraint_type must be: ExcludeMapRestraint1

threshold Density threshold to define the EM envelope

filename relative or absolute path to the map in the MRC format

voxel_size Pixel/voxel size of the map in Angstrom

resolution Resolution for the EM map simulated from the model for cross-correlation calculation. Can be e.g. approximately the resolution of your map.

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

optimized_components A list of *Selectors* defining molecules or parts to which this restraint should be applied to.

22.5 CloseToEnvelopeRestraint

A restraint pulling particles towards the envelope defined density. The restraint is 0 if the particle is at the envelope defined at the provided threshold and increases with the distance from the envelope, either inside or outside of the envelope.

Can be used for example to restrain membrane binding regions to a membrane.

bring particles into the target EM density where they will be fitted by the other EM restraint.

```
{
    "active": true,
    "type": "em_map",
    "name": "Nup132_close_to_membrane1",
    "em_restraint_type": "CloseToEnvelopeRestraint",
    "filename": "../EM/Membrane_final_resampled_no_neg.mrc",
    "threshold": 0.02,
    "voxel_size": 6.9,
    "resolution": 20,
    "weight": 2000,
    "max_distance": 200,
    "first_copy_only": false,
    "repr_resolution": 10,
    "optimized_components": [
        {
            "name": "Nup132",
            "subunit": "Nup132",
            "domain": "membrane_binding",
            "copies": [
                0,1
            ],
            "serie": "NR_1"
        }
    ]
}
```

Parameters

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

type must be: em_map

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

em_restraint_type must be: CloseToEnvelopeRestraint

threshold Density threshold to define the EM envelope

filename relative or absolute path to the map in the MRC format

voxel_size Pixel/voxel size of the map in Angstrom

resolution Resolution for the EM map simulated from the model for cross-correlation calculation. Can be e.g. approximately the resolution of your map.

weight Weight of this restraint

max_distance Distance of a particle from the envelope at which to start evaluating the restraint, default 100 A. Increasing may decrease computational speed.

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

optimized_components A list of *Selectors* defining molecules or parts to which this restraint should be applied to.

22.6 EnvelopeFitRestraint

A restraint from IMP that "fits the particles into the density map by alignment of principal components of the particles with principal components of the map", read more EnvelopeFitRestraint from IMP

I never use it.

It can be defined in the *JSON configuration file* using blocks like this:

```
{
    "active": true,
    "type": "em_map",
    "name": "restraint_type_I_never_use",
    "em_restraint_type": "EnvelopeFitRestraint",
    "filename": "EM/map.mrc",
    "threshold": 0.001,
    "voxel_size": 6.9,
    "resolution": 20,
    "weight": 1000,
    "first_copy_only": true,
    "repr_resolution": 10,
    "optimized_components": [
        {
            "name": "Elys",
            "subunit": "Elys"
        }
    ]
}
```

Parameters as above for EnvelopePenetrationRestraint.

CHAPTER

TWENTYTHREE

CROSSLINK RESTRAINTS

Crosslink restraints restrain distance between crosslinked residues.

23.1 Loading crosslink data

Follow the guideline in *JSON configuration file* of how to add crosslinks using the graphical interface of Xlink Analyzer.

23.2 Defining the restraints

Crosslinks restraints are defined in the *Parameter file* by adding blocks like this:

```
add_xlink_restraints = True
x_min_ld_score = 25
x_weight = 100.0
x_xlink_distance = 30
x_xlink_score_type='HarmonicUpperBound'
```

and then adding `xlink_restraints` to `scoring_functions` (see *Parameter file*).

If multiple representation resolutions are defined, the crosslinks will be added to the highest resolution possible (the smallest beads).

23.3 Parameters

x_xlink_score_type A type of crosslink restraint.

Options:

'HarmonicUpperBound' - 0 below x_xlink_distance, harmonic above, distance calculated between centers of the beads

'HarmonicUpperBoundSphereDistancePairScore' - 0 below x_xlink_distance, harmonic above, distance calculated between surfaces of the beads

'XlinkScore' - 0 below x_xlink_distance, 1 above

'LogHarmonic' - A log harmonic potential with the maximum at x_xlink_distance

'CombinedHarmonic': - harmonic above distance of 35 A and log harmonic with the maximum at x_xlink_distance below 35 A

default: **HarmonicUpperBound**

x_min_ld_score Only crosslinks with the confidence score above this threshold will be used as restraints. The score must be defined in "score" column of crosslink CSV files (see also Xlink Analyzer documentation), default: **30.0**

x_weight Weight of crosslink restraints, default: **1.0**

x_xlink_distance Target or maximual crosslink distance (depending on the implementation), default: **30.0**

x_k Spring constant for the harmonic potential, default: **1.0**

x_inter_xlink_scale Multiply the weight of inter-molecule crosslinks by this value, default: **1.0**

x_first_copy_only Apply crosslinks only to the first molecule copy of each series, default: **False**

x_skip_pair_fn A Python function to skip specific crosslinks. Arguments: p1, p2, component1, component2, xlink Return True to skip the crosslink, default: **None**

x_log_filename Optional log file name for printing more information about added and skipped crosslinks, default: **None**

x_score_weighting Scale crosslink weights by their score, default: **False**

xlink_reweight_fn A custom Python function to scale crosslink weights. Arguments: xlink, weight Return final weight (float), default: **None**

x_random_sample_fraction Take this random fraction of crosslinks for modeling, default: **1.0**

CHAPTER

TWENTYFOUR

SYMMETRY RESTRAINTS AND CONSTRAINTS

Symmetry can be restrained by either **constraints** or **restraints**.

Constraint enforces exact symmetry, with no deviations, while **restraints** only promote symmetry with strength determined by restraint weight.

Follow instructions:

- *Symmetry constraints*
- *Symmetry restraints*

CHAPTER

TWENTYFIVE

INTERACTION RESTRAINTS

Interaction restraints can be used to define interactions between entire subunits, domains or single residues.

They can be defined in *JSON configuration file* like this:

- Interaction between two proteins:

```
{
    "type": "connectivity",
    "active": true,
    "name": "Ely5-Nup120_interaction1",
    "comment": "Ely5 - Nup120 interaction",
    "data": [
        [{"subunit": "Nup120", "serie": "NR_1", "copies": [0]}, {"subunit":
↪"Ely5", "serie": "NR_1", "copies": [0]}]
    ],
    "weight": 0.1,
    "k": 1,
    "distance": 7.0,
    "first_copy_only": true,
    "repr_resolution": 10
}
```

- Distance between specific residues:

```
{
    "type": "connectivity",
    "active": true,
    "name": "Nup107_connectivity",
    "comment": "Keep the N and C domains of Nup107 close to account for the␣
↪three missing helices",
    "data": [
        [{"subunit": "Nup107", "serie": "NR_1", "resi_ranges": [494]}, {
↪"subunit": "Nup107", "serie": "NR_1", "resi_ranges": [580]}],
        [{"subunit": "Nup107", "serie": "NR_2", "resi_ranges": [494]}, {
↪"subunit": "Nup107", "serie": "NR_2", "resi_ranges": [580]}],
        [{"subunit": "Nup107", "serie": "NR_1", "resi_ranges": [512]}, {
↪"subunit": "Nup107", "serie": "NR_1", "resi_ranges": [595]}],
        [{"subunit": "Nup107", "serie": "NR_2", "resi_ranges": [512]}, {
↪"subunit": "Nup107", "serie": "NR_2", "resi_ranges": [595]}]
    ],
    "weight": 1000.0,
    "k": 1,
```

(continues on next page)

(continued from previous page)

```
    "distance": 25.0,
    "first_copy_only": true,
    "repr_resolution": 1
}
```

Here, each pair of residues specified in the `"data"` list will be restrained to distance **below** 25A

Parameters

type Must be "connectivity"

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

comment Optional comment, useful for documenting

data A list of **pairs** of *Selectors* between which the interaction restraints should be defined

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

k Spring constant for the harmonic potential, redundant to weight, ignore

distance Distance threshold below which the restraint is satisfied. The score is 0 below this threshold and harmonic above.

CHAPTER

TWENTYSIX

DISTANCE RESTRAINTS

Currently, the distances can restrained to **below** certain length using *Interaction restraints*

CHAPTER

TWENTYSEVEN

SIMILAR ORIENTATION RESTRAINTS

These restraints promote similar orientation between pairs of identical subunits, domains or residues.

They can be defined in the *JSON configuration file* like this:

```
{
    "type": "similar_orientation",
    "active": true,
    "name": "Ely5-Nup120_similar_orientation",
    "comment": "We assume Ely5 interacts in the same way with all copies of Nup120",
    "data": [
        [
            [
                {"subunit": "Nup120", "serie": "NR_1", "copies": [0]}, {"subunit": "Ely5
↪", "serie": "NR_1", "copies": [0]}
            ],
            [
                {"subunit": "Nup120", "serie": "NR_2", "copies": [0]}, {"subunit": "Ely5
↪", "serie": "NR_2", "copies": [0]}
            ]
        ]
    ],
    "weight": 10,
    "k": 1,
    "first_copy_only": true,
    "repr_resolution": 10
}
```

Parameters

type Must be "similar_orientation"

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

comment Optional comment, useful for documenting

data A **pair of pairs** of *Selectors* between which the interaction restraints should be defined

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

k Spring constant for the harmonic potential, redundant to weight, ignore

CHAPTER

TWENTYEIGHT

ELASTIC NETWORK RESTRAINTS

Elastic network restraints allow to flexibly restrain input orientations between subunits, domains or residues.

The input orientations are as read from the input PDB files, regardless whether the particles are in the same or different PDB files.

The elastic network can be defined in the *JSON configuration file* like this:

```
{
    "active": true,
    "type": "elastic_network",
    "name": "Nup107_Nup132_elastic_network1",
    "repr_resolution": 10,
    "distance_threshold": 15,
    "random_fraction": 1.0,
    "within_molecule": false,
    "k" : 1,
    "weight": 100,
    "first_copy_only": true,
    "optimized_components": [
        {
            "name": "Nup132", "subunit": "Nup132", "serie": "NR_1",
            "resi_ranges": [
                [
                    459,
                    1162
                ]
            ]
        },
        {
            "name": "Nup107", "subunit": "Nup107", "serie": "NR_1",
            "resi_ranges": [
                [
                    581,
                    813
                ]
            ]
        }
    ]
}
```

This example will create elastic network between the specified residue ranges of Nup132 and Nup107 subunits.

Parameters

type Must be "elastic_network"

active true or false, whether the restraint should be created, you can set to false to save on loading time if you don't need this restraint (even if it's true, you still need to add it to a scoring function to have it an effect)

name whatever name. Use this name to refer to this restraint in *Parameter file* when defining the scoring functions

comment Optional comment, useful for documenting

weight Weight of this restraint

first_copy_only true or false, Apply only to the first molecule copy of each series?

repr_resolution The resolution of representation to which this restraint should be applied

k Spring constant for the harmonic potential, redundant to weight, ignore

optimized_components A list of *Selectors* defining molecules or parts to which this restraint should be applied to.

distance_threshold Residue pairs within this distance threshold will be included in the elastic network

random_fraction A fraction of distances below the thresholds to be randomly selected for creating the elastic network, default: 1.0 (all distances)

within_molecule If set to `true`, only distances within the same subunit will be added to the elastic network

CHAPTER

TWENTYNINE

PARSIMONIOUS STATES RESTRAINTS

Note: This type of restraints is still in experimental stage. In case of urgent use/set-up please contact: vasileios.rantos@embl-hamburg.de, jan.kosinski@embl.de

They are defined in *Parameter file* by setting:

```
add_parsimonious_states_restraints = True
```

parameters

add_parsimonious_states_restraints Add parsimonious states restraints?, default: **False**

parsimonious_states_weight Weight, default: **1.0**

parsimonious_states_distance_threshold Distance threshold for elastic network restraining the states, default: **0.0**

parsimonious_states_exclude_rbs Python function to exclude selected rigid bodies from the restraint. Arguments: IMP's rigid body object Return: True if the rigid body should be excluded., default: Some default function

parsimonious_states_representation_resolution Representation resolution for this restraint, default: **10**

parsimonious_states_restrain_rb_transformations Restraint rigid body transformations instead of using elastic network, default: **True**

CHAPTER

THIRTY

CUSTOM RESTRAINTS

Note: This is rather for developers!

Any custom restraints can be added through a `create_custom_restraints` function in *Parameter file*.

The function accepts `imp_uttils1.MultiRepresenation` object as an argument, and should output a dictionary mapping your chosen restraint names to lists of the restraints. The restraints must subclass the IMP.Restraint or have same interface.

So you can get or add any native IMP restraints through this function.

Warning: Don't forget to add your restraints to the scoring functions in *Parameter file*!

For example, the following example will define restraints that do nothing. It seems that such restraints have to be added if you use symmetry constraints and assign all your restraints only to the first copy of each series - in such cases it seems that the copies are not updated properly.

```
def create_custom_restraints(r):
    '''
    Added so all copies are updated by constraints:
    when the system contains molecules for which there are no restraints imposed in the␣
↪scoring function,
    they won't be updated during Optimization, only on model.update().
    '''
    for state_idx, state in enumerate(r.pmi_system.states):
        all_optimized = []
        for serie in r.config.series:
            for i, mol in enumerate(serie.molecules[state_idx]):
                rbs, beads = IMP.pmi.tools.get_rbs_and_beads([mol.get_hierarchy()])
                # print mol.get_name(), [x.get_particle().get_name() for x in ref_rbs]
                for rb in rbs:
                    p = rb.get_particle()
                    if p not in all_optimized:
                        all_optimized.append(p)

    return {'dummy_restraints': [imp_utils1.core.DummyRestraint(r.model, all_optimized)]}
```

CHAPTER

THIRTYONE

RUN THE MODELLING

Warning: Before running Assembline for your modelling target make sure you have set up properly your modelling virtual environment and dependencies, as described in *System requirements* section of this manual!

To run Assembline installed with Anaconda, activate the Assembline environment by:

```
source activate Assembline
```

or depending on your computer setup:

```
conda activate Assembline
```

31.1 Introduction

The modelling at any stage is run with the same command

```
assembline.py <options> X_config.json params.py
```

The information in `X_config.json` and `params.py` determine which step (i.e. *1. Global optimization*, *2. Recombinations* or *3. Refinement* is run)

This command can be used to execute a single "run", which generates a single model, or multiple runs, each leading to a model.

Typically hundreds to thousands runs need to be executed (leading to the number of models same as the number of runs) to ensure exhaustive sampling and find optimal models.

31.2 A single run

To generate a single model just run:

```
assembline.py --traj --prefix 0000000 -o <output directory> --models X_config.json␣
↪params.py
```

It is always useful to run this before executing the many runs as below. Run this in an interactive session to check for any errors and to check if everything has been set up correctly based on the log messages.

31.3 Multiple runs

- Method 1: Submit all runs to the computer cluster or run on a workstation in chunks of N according to the number of processors:

```
assembline.py --traj --models -o out --multi --start_idx 0 --njobs 1000 X_
↪config.json params.py &>log&
```

  - on a cluster, this will submit 1000 modelling jobs in the queue, each job leading to one model (if ntasks in params.py is set to 1)
  - if ntasks params.py is N, it will run submit 1000/N cluster jobs, each running N modelling jobs
  - on a multicore computer, it will run ntasks at a time, and keep running until all 1000 jobs are done.

Note: The number of processors or cluster submission commands and templates are specified in `params.py`

- Method 2: Dynamically adjust the number of concurrent runs (e.g. to not overload a cluster or annoy other users):

> **Warning:** The following works out of the box only on the EMBL cluster. To adjust to your cluster, modify the `assembline.py` for your cluster environment following the guidelines in the script.

```
assembline.py --traj --models --multi --daemon --min_concurrent_jobs 200 --
↪max_concurrent_jobs 1000 -o out --start_idx 0 --njobs 1000 X_config.json␣
↪params.py &>log&
```

Note: Check if jobs are being submitted to the cluster queue. By default the script runs as many jobs as there are cluster slots free to avoid overloading the cluster with too many jobs thus hindering jobs/tasks of other users and/or cluster admins.:: *–min_concurrent_jobs 200 –max_concurrent_jobs 2000*

this would make sure that you have at least 200 jobs in the queue and not more than 2000 (even if there are more than 2000 free). For jobs that take longer than few minutes it is OK to use `--max_concurrent_jobs 10000` (short jobs or crashing jobs can crash the queuing systems; e.g. submitting 10000 jobs that crash after 1 minute because of an error crashes the Slurm scheduler).

Finally, log out and log back (otherwise automated session might disconnect killing your daemon)

- Method 3: If none of the above solutions works for you, you could submit multiple jobs manually using a shell loop like the following e.g. on a computer cluster with the Slurm queuing system:

```
for i in $(seq -f "%07g" 0 999)
    do
        srun assembline.py  --traj --models --prefix $i -o out X_config.
↪json params.py &>log&
    done
```

> **Warning:** Just remember to make the `prefix` unique for every run.

31.4 Assembline.py options

- `--traj` option is optional and determines whether to save trajectory of optimization.

> **Warning:** Setting a low number of `traj_frame_period` in *Parameter file* for many and long modelling runs will slow down the modelling procedure significantly.

31.5 Checking if everyhting runs correctly

1. The above command (i.e. `assembline.py --traj --models --multi --daemon ....`) redirects the output to `log` file. Check the `log` file for any errors
2. Check if the output is being created:
   - The script should create the following directories in the < output directory >
     - `logs/` - directory with the logs for each run
     - `models_rmf/` - models in the RMF format
     - `models_txt/` - models in a special format that specifies transformation matrices for each rigid body
     - `traj/` - (only if –traj option was specified) optimization trajectories in the RMF format
3. Check the < `output directory` >`/logs` in case of crashes or to check/display diagnostic messages

CHAPTER

THIRTYTWO

1. GLOBAL OPTIMIZATION

1. Enter the project directory
2. Prepare the *JSON configuration file* file, name it `X_config.json` for example.
3. Add fit libraries to *JSON configuration file* as described in *Adding precomputed fitting libraries to JSON*
4. Create *Parameter file*, name it `params.py` for example.

   For the global optimization, the following parameters are important:

   - `protocol`

     ```
     protocol = 'denovo_MC-SA'
     ```

     or, if you have flexible beads and want to optimize them using Conjugate Gradient:

     ```
     protocol = 'denovo_MC-SA-CG'
     ```

     Note: The `protocol` parameter takes as argument keywords specifying the Assembline modelling mode to be used as described in *Parameter file*.

   - in `scoring_functions`, `discrete_restraints` must be added to all scoring functions to indicate restraints from fit libraries.

     Warning: Do not run global optimization without discrete restraints as it will completely ruin the transformation matrices for the fit libraries

   - `discrete_restraints_weight`

5. Run the modeling as described in *Run the modelling*

CHAPTER

THIRTYTHREE

2. RECOMBINATIONS

1. Enter the output directory of the global optimization run
2. Prepare a JSON file for recombination

```
setup_recombination.py \
    --json <the JSON file that was used for the global optimization run> \
    --scores all_scores_uniq.csv \
    -o <output directory for the new fit libraries> \
    --json_outfile <desired name for JSON config for recombinations> \
    --project_dir <original project dir> \
    --score_thresh <score threshold for selecting the models> \
    --top <number of top scoring models to use for extracting the fit␣
↪libraries>
```

   You can use either `--score_thresh` or `--top or both`.

   This command will create a file specified in `json_outfile`, which is the input JSON for recombination.

3. Enter the original project directory
4. Run using the same `params.py` as for the previous global optimization run.

   Optionally, you can reduce the number of Simulated Annealing steps, as the number of the fits in the libraries is likely lower, and the optimization converges faster.

   Run in the same way as the *1. Global optimization*, but now pointing to the new JSON generated above and outputing to a subdirectory, and adding `--prefix` option to distinguish the recombined models from the original ones as in the example below:

```
assembline.py \
     --traj \
     --models \
     -o out \
     --multi \
     --start_idx 0 \
     --njobs 1000 \
     --prefix recomb \
     config_recomb.json params.py &>log&
```

   This will output to the same directory, so all models can be analyzed together.

CHAPTER

THIRTYFOUR

3. REFINEMENT

34.1 Refine the top-scoring models from previous modelling modes/runs

1. Enter your main project directory (where your X_config.json is)
2. Prepare a template JSON file for refinement

```
gen_refinement_template.py --out_json refine_template.json --params params.
↪py X_config.json
```

3. Modify the template according to your needs, e.g.:
    1. Add restraints specific for the refinement
    2. Change weights of restraints
    3. Re-define the rigid bodies
    4. Add extra subunits and their PDB files

    See JSON setup in the Elongator tutorial for an example.
4. Create a `params_refine.py` file as in *Parameter file* but now with the *refine* protocol and adjusting the scoring function and other settings, as appropriate for your case.

    See params set up in the Elongator tutorial for an example.

34.2 Refine a single model

1. Set up a refinement directory

```
model_id=0014032model
setup_refine.py \
    --model $model_id \
    --previous_json X_config.json \
    --refine_json_template refine_template.json \
    --refine_json_outname refine.json \
    --previous_outdir <output directory of the denovo run>/\
    --refine_outdir <desired output directory for the refinement, e.g. out/
↪refinement>
```

    The script will create a new refinement directory and copy all input files there.

2. Perform test run

```
mkdir -p <desired output directory for the refinement>/$model_id/testout
assembline.py \
    --traj \
    --models \
    -o <desired output directory for the refinement>/$model_id/testout \
    --prefix refine_"$model_id"_000000 \
    <desired output directory for the refinement>/$model_id/refine.json \
    params_refine.py
```

3. Run

```
assembline.py \
    --traj \
    --models \
    -o <desired output directory for the refinement>/$model_id/out \
    --multi \
    --start_idx 0 \
    --njobs 3 \
    --prefix refine_"$model_id" \
    <desired output directory for the refinement>/$model_id/refine.json \
    params_refine.py
```

34.3 Refine multiple models

1. Set up a refinement directory

```
setup_refine.py \
    --top 10 \
    --scores <output directory of the denovo run>/all_scores_uniq.csv \
    --previous_json X_config.json \
    --refine_json_template refine_template.json \
    --refine_json_outname refine.json \
    --previous_outdir <output directory of the denovo run>/\
    --refine_outdir <desired output directory for the refinement, e.g. out/
↪refinement>
```

2. Run recursively:

   If the output directory for the refinement was defined above as `out/refinement`:

```
for model_id in `ls --color=never out/refinement`;
    do
        echo $model_id
        assembline.py \
            --models \
            -o out/refinement/"$model_id"/out \
            --multi \
            --start_idx 0 \
            --njobs 3 \
            --prefix refine_"$model_id" out/refinement/"$model_id"/refine.
↪json \
```

(continues on next page)

(continued from previous page)

```
        params_refine.py
done
```

CHAPTER

THIRTYFIVE

RUNNING MORE

If the exhaustiveness is not met, run more jobs:

```
assembline.py --traj --models -o out --multi --start_idx 10000 --njobs 10000 config.json␣
↪params.py &>log&
```

which will run 10000 more jobs with models named with a prefix starting from `0010000` and repeat the sampling exhaustiveness analysis above

Note: To perform *Sampling exhaustiveness and precision* analysis visit the relevant setion in the manual.

CHAPTER

THIRTYSIX

MODIFY AND RE-RUN

36.1 Typical adjustments that need to be done:

- Adjust the weights of xlink (`x_weight`), alternative positions (`discrete_restraints_weight`) etc.
  - Currently, you would run the first run and adjust the weights so the scores are of similar scale.
- Adjust KT in MonteCarlo Simulated Annealing so it is on the same range as the total score (e.g. for score of 100000 you would not use KT of 1 but rather 100-10000)
- Increase/decrease resolution to adjust the speed of calculations
- Add/remove flexible beads
- Modify sampling protocol
- Add additional restraints

36.2 Was the calculation quick?

Then you may consider increasing the resolution in `params.py` of the representation to a higher one (i.e. resolution) hence probably more accurate one, e.g.: setting:

```
struct_resolutions = [0]
```

CHAPTER

THIRTYSEVEN

CHECK THE LOGS

The `assembline.py` script prints various logs to the screen or to the log files indicated in the cluster submission template.

The `log` includes:

- the list of final parameter values
- summary of the molecular system created: molecules, series, chain IDs, rigid bodies, etc.
- restraints that are being created
- optimization logs:
  - restraint values used for optimization, before and after optimization, and between each simulated annealing temperature change

At this point you need to inspect the `logs` and determine whether the diagnostic messages in `logs` directory are matching what you intended:

- Are all mappable crosslinks added properly as restraints?
- Do the created rigid bodies match what you expected?
- Is the number of connectivity restraints ok?
- Are the Monte Carlo moves as expected from the configuration file?
- The log displays how many times each rigid body moved, is the number reasonable?
  - **E.g. if you ask for 10,000 steps at each simulated annealing temperature and have 5 rigid bodies and 500 flexible bead** each rigid body only a small number of moves (each step is a single move) and you may need to increase the number of steps. For flexible beads it is not a problem if the "denovo_MC-SA-CG" protocol has been specified in `params.py` - they would move substantially with additional Conjugate gradient steps
- Are the Monte Carlo acceptance rates reasonable?
  - at high Simulated Annealing temperatures the fraction of upward moves should be high, and decrease with lower temperature
  - the number of downward movements should be non-zero and decrease with temperature

CHAPTER

THIRTYEIGHT

EXTRACT SCORES

Output all scores (and scoring terms) of the produced models

To get models with the best scores and all scoring terms reported run the following (obligatory step for downstream analysis with imp-sampcon):

```
extract_scores.py
```

or

```
extract_scores.py --multi
```

for *3. Refinement* of multiple models.

It will create a couple of files:

```
total_score_logs.txt (a list of scores with total scores in trajectories, only if print_
↪total_score_to_files was set to True in params file
all_scores_sorted_uniq.txt (txt file with models and sorted non-redundant total scores)
all_scores_sorted.txt (txt file with models and sorted total scores)
all_scores.txt (txt file with models and total scores)
all_scores_uniq.csv (CSV file with sorted non-redundant total scores)
all_scores_sorted_uniq.csv (CSV file with models and sorted non-redundant total scores)
all_scores_sorted.csv (CSV file with models and sorted total scores)
all_scores.csv (CSV file with models and total scores)
```

These files are used as input for other scripts. You can also use them for your own statistics, filter them before running the scripts.

38.1 Plot histograms of all scores

```
plot_scores.R all_scores.csv
```

CHAPTER

THIRTYNINE

VISUALIZE MODELS AND TRAJECTORIES

Assembline can output models in several formats.

By default, the following formats are produced:

- RMF format in the `models_rmf` directory

  The RMF files can be visualized in UCSF Chimera and UCSF ChimeraX

- Simple TXT format that stores transformations of rigid bodies in the `models_txt` directory

In addition, optionally, trajectories of the optimization runs can be saved in the RMF format, in `traj` directory (using `--traj` option of `assembline.py`)

These two formats can be used to generate PDB or CIF files of the resulting integrative models, using `rebuild_atomic.py` atomic script.

39.1 Create CIF format file for top models

In the output directory, run:

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1␣
↪all_scores_sorted_uniq.csv
```

Note: The `--project_dir` option is only necessary if you use relative paths in the *JSON configuration file*

Warning: This will only build the parts corresponding to rigid bodies, if had flexible beads in the system, see below how to rebuild them at the atomic resolution

39.2 Create CIF format file for a specific model

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1␣
↪models_txt/0002321model.txt
```

39.3 Rebuilding flexible beads

If you had single residue flexible beads in your modeling (i.e. at representation resolution of 1), they can be rebuilt by running this script with `--rmf` and `--rmf_auto` option:

For the top models:

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1 --
↪rmf_auto all_scores_sorted_uniq.csv
```

For a specific model:

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1 --
↪rmf models_rmf/0002321model.rmf models_txt/0002321model.txt
```

The `-rmf` options tell the script to extract flexible beads from the RMF files and re-model them at atomic resolution using Modeller.

39.4 Create PDB for top models

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1 --
↪format pdb all_scores_sorted_uniq.csv
```

Warning: The PDB format is not recommended (especially for multi-subunit complexes) - it does not support multi-character chain IDs and has a limit on the number of atoms in the file.

39.5 Create separate PDB files for rigid bodies

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 1 --
↪format multi_pdb all_scores_sorted_uniq.csv
```

This can be useful if you want to use the models for subsequent optimizations with Assembline or other software.

The scripts used for *3. Refinement* are using the multi_pdb format behind the scenes.

39.6 Other options

See `rebuild_atomic.py` usage instructions for other useful options:

```
rebuild_atomic.py -h
```

For example:

- to rebuild a specific subunit use `--subunit` option
- to rebuild only the first copy of each series (often the asymmetric unit of the complex) use `--first_copy_only` or `--copies 0` option
- to rebuild specific copies use `--copies` option, e.g. `--copies 0,1,7` option

39.7 Visualize the top model(s) in Chimera and Xlink Analyzer

Visualizing top models in Chimera and Xlink Analyzer will help to answer:

- Are crosslinks satisfied?
- Are there big clashes between subunits?
- Are models fitting other information you have but not used in the optimization?

39.8 Visualize the optimization trajectory of the top model(s) in Chimera and Xlink Analyzer

- The trajectories in the RMF format are stored in the `traj/` folder of the output directory
- UCSF Chimera allows:
  - to open the RMF format and play the trajectories
  - display plots of the total score and individual restraints over the course of the trajectory
- Analyze:
  - are all rigid bodies expected to move during the trajectory?
  - are the total scores and all restraints decreasing?
- For better visualization of crosslinks and to color the subunits in the RMF by subunit color, you can use Xlink Analyzer.

Note: For Xlink Analyzer to work, you need to replace a file `share/rmf/__init__.py` in your Chimera installation with this file: https://github.com/crosslinks/XlinkAnalyzer/blob/master/edited__init__.py_for_chimera_rmf

---

  - The `share` directory is located within the Chimera installation folder (https://www.cgl.ucsf.edu/chimera/experimental/install.html):

```
source

Windows
                C:\Program Files\Chimera\share
Linux
                /usr/local/chimera/share
Macintosh
                /Applications/Chimera.app/Contents/Resources/share
```

CHAPTER

FORTY

QUICK TEST OF CONVERGENCE

Run quick test to assess convergence of the model score in randomly selected modelling trajectories

```
cd <output directory>
plot_convergence.R total_score_logs.txt 20
```

(change `20` to have less or more trajectories in the plots).

Note: The above is a quick and optional step before running the complete sampling exhaustiveness analysis for your modelling runs.

Open the resulting `convergence.pdf` to visualize the convergence.

CHAPTER

FORTYONE

SAMPLING EXHAUSTIVENESS AND PRECISION

Run sampling performance analysis with imp-sampcon tool (described by Viswanath et al. 2017)

1. Enter the output modelling directory
2. Prepare the `density.txt` file:

```
create_density_file.py --project_dir <path to the original project dir>␣
↪config.json --by_rigid_body
```

3. For complexes containing multiple copies of the same subunit, prepare the `symm_groups.txt` file storing information necessary to properly align homo-oligomeric structures

```
create_symm_groups_file.py --project_dir <path to the original project dir>␣
↪config.json params.py
```

   By default all molecule copies of the same subunits are grouped together, and this should be sufficient in most cases.

   In some special cases where subunits are directed to specific series, to group by series use `--by series` option:

```
create_symm_groups_file.py --project_dir <path to the original project dir>␣
↪--by series config.json params.py
```

   To additionally group series into bigger groups use `--extra_series_groups` option, e.g.:

```
create_symm_groups_file.py \
--project_dir <path to the original project dir> \
--by series \
--extra_series_groups NR_1,NR2;CR_1,CR_2 \
config.json params.py
```

   which will group by series but on top will consider ambiguity between selected series

4. Run `setup_analysis.py` script to prepare input files for the sampling exhaustiveness analysis based on your resulting models:

```
setup_analysis.py -s <abs path to csv scores file produced by extract_all_
↪scores.py> \
-o <specified output dir> \
-d <density.txt file generated in the previous step> \
-n <number of top scoring models to be analyzed, default is all models> \
-k <restraint score based on which to perform the analysis, default is␣
↪total score>
```

Example:

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt -n 20000
```

To see available options and default values run:

```
setup_analysis.py -h
```

5. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the actual analysis:

```
cd <output dir created by the above setup_analysis.py script>

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d <path to density.txt file>/density.txt \
-m <calculator selection> \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step>
```

To see available options and default values for imp-sampcon exhaust analysis run:

```
imp_sampcon exhaust -h
```

In case the analysis will be run on slurm-based cluster then compile a bash script like the following and run with `sbatch`:

```
#!/bin/bash
#SBATCH --job-name=master_sampling_20000.job
#SBATCH --output=./master_sampling_20000.out
#SBATCH --error=./master_sampling_20000.err
#SBATCH --nodes=1
#SBATCH --time=10:00:00
#SBATCH --qos=highest
#SBATCH --cpus-per-task=15
#SBATCH --mem-per-cpu=4000

imp_sampcon exhaust -n CR_Y_test --rmfA sample_A/sample_A_models.rmf3 --
↪rmfB sample_B/sample_B_models.rmf3 --scoreA scoresA.txt --scoreB scoresB.
↪txt -d density.txt -m cpu_omp -c 15 -gp -g 5.0
```

6. In the output you will get, among other files:

   - `<prefix for output files>.Sampling_Precision_Stats.txt` with estimation of the sampling precision.
   - Clusters obtained after clustering at the above sampling precision in directories and files starting from `cluster` in their names, containing information about the models in the clusters and cluster localization densities
   - `<prefix for output files>.Cluster_Precision.txt` listing the precision for each cluster
   - PDF files with plots with the results of exhaustiveness tests

   See Viswanath et al. 2017 for detailed explanation of these concepts.

7. Optimize the plots

   The fonts and value ranges in X and Y axes in the default plots from `imp_sampcon exhaust` are frequently not optimal. For this you have to adjust them manually.

   1. Copy the original `gnuplot` scripts to the current `analysis` directory by executing:

      ```
      copy_sampcon_gnuplot_scripts.py
      ```

      This will copy four scripts to the current directory:

      - `Plot_Cluster_Population.plt` for the `<prefix for output files>.Cluster_Population.pdf` plot
      - `Plot_Convergence_NM.plt` for the `<prefix for output files>.ChiSquare.pdf` plot
      - `Plot_Convergence_SD.plt` for the `<prefix for output files>.Score_Dist.pdf` plot
      - `Plot_Convergence_TS.plt` for the `<prefix for output files>.Top_Score_Conv.pdf` plot

   2. Edit the scripts to adjust according to your liking or needs

   3. Run the scripts again:

      ```
      gnuplot -e "sysname='<prefix for output files>'" Plot_Cluster_
      ↪Population.plt
      gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_
      ↪NM.plt
      gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_
      ↪SD.plt
      gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_
      ↪TS.plt
      ```

      For example:

      ```
      gnuplot -e "sysname='elongator'" Plot_Cluster_Population.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_NM.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_SD.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_TS.plt
      ```

8. Extract cluster models

   For example, to extract the 5 top scoring models:

   ```
   extract_cluster_models.py \
       --project_dir <full path to the original project directory> \
       --outdir cluster.0/ \
       --ntop 5 \
       --scores ../all_scores.csv \
       Identities_A.txt Identities_B.txt cluster.0.all.txt ../config.json
   ```

9. If the exhaustiveness is not met, run more jobs:

   Running more

CHAPTER

FORTYTWO

GLOSSARY

constraint enforced and not able to be violated sets of values in modelling (e.g. distances, coordinates etc.)

restraint favored and able to be violated (up to some degree) sets of values in modelling (e.g. distances, coordinates etc.)

series groups of modelling targets (a kind of target selection) to which specific modelling parameters can be applied

subunit single protein chain which is part of the modelling target system

particle set of system entities like atom, group of atoms, residues, protein chains etc. (depends on hierarchy)

selector method for grouping (or selecting) particles, subunits, system states etc.

representation resolution target system's coarse grained or even atomic representation

transformation matrix sets of coordinates dictating the final rotation and translation moves of particles

CHAPTER

FORTYTHREE

FREQUENTLY ASKED QUESTIONS

- How do I cite Assembline?
- Whom should I contact in case of discovered bugs or for general support?
- How many and which types of input data should I have to use Assembline?
- My modelling target is a single protein, can I still use Assembline?
- Can I define custom spatial restraints/constraints?
- When should I stop running Assembline based on sampling exhaustiveness analysis?
- How should I decide (rule of thumb) on weights for scoring terms in my scoring function?
- How is Assembline performing compared to other integrative modelling packages?

CHAPTER

FORTYFOUR

TROUBLESHOOTING

CHAPTER

FORTYFIVE

TIPS AND TRICKS

45.1 Speeding up calculations

- Make your EM maps smaller

  Sometimes you may get an EM map which box is much larger than the actual map. All those extra and empty voxels will slow down calculations of cross-correlation function during calculations of *fit libraries* and *FitRestraint*.

  You can remove the extra padding voxels in an EM processing software, for example UCSF Chimera

- Downsample the EM map

  Is your map low resolution but has very small voxel size (e.g. EM map at 10 A with voxel size of 1 A)? Such unnecessarily small voxel size will slow down computations.

  Downsample your EM map then in an EM processing software, for example UCSF Chimera or EMAN2.

- Use different system representation resolutions for different restraints

  Is it necessary to use atomic representation to calculate all you pre-defined restraints? If your computational running times are relatively high then you should consider choosing more coarse grained representations for specific restraints (e.g. Excluded volume restraints).

- Always run single test runs before submitting larger modelling jobs to the cluster (or even to your local multi-core cpu).

INDEX

C

constraint, **121**

P

particle, **121**

R

representation resolution, **121**
restraint, **121**

S

selector, **121**
series, **121**
subunit, **121**

T

transformation matrix, **121**

ScNPC modelling tutorial

Release 1.0

Vasileios Rantos, Kai Karius, Jan Kosinski

Aug 24, 2021

CONTENTS

1 About the S. cerevisiae nuclear pore complex (ScNPC) 3

2 Data preparation 5

3 About fit libraries 11

4 Set up 13

5 Run 17

6 Analyze fits 19
6.1 Check if run correctly . . . 19

7 Setting up the JSON project file 21
7.1 Create Xlink Analyzer project . . . 21
7.2 Add modelling information to the project file . . . 22

8 Setting up the parameter file 43

9 Run 47

10 Analyze 49
10.1 Extract scores . . . 49
10.2 Create a CIF file of the top N models . . . 49
10.3 Assess sampling exhaustiveness . . . 49

11 Setting up the JSON project file 53

12 Setting up the parameter file 55

13 Run 59

14 Analyze 61

15 CR Y-complex modelling 63

16 IR asymmetric unit modelling 65

17 NR Y-complex modelling 67

18 Nup116 k.o. ScNPC (at 25C) modelling 71

19 Nup116 k.o. ScNPC (at 37C) modelling 73

Assembline is a software for integrative structural modelling of macromolecular assemblies - an assembly line of macromolecular assemblies!

This tutorial demonstrates the usage of Assembline on an example of S. cerevisiae nuclear pore complex (ScNPC) from wild-type (wt) and Nup116 (NPC member protein) knock-out (k.o.) cells. By following the detailed data preparation and modelling steps for Assembline the users will be able to reproduce our previously published integrative models (Allegretti et al. (2020)) of: Cytoplasmic Ring Y-complex (CR Y-complex), Inner Ring asymmetric unit (IR asymmetric unit), Nuclear Ring Y-complex (NR Y-complex), Nup116 k.o. ScNPC (at 25C), Nup116 k.o. ScNPC (at 37C).

- All the necessary input data and modelling templates are provided (as well as some precalculated output) and should be downloaded from our ScNPC_tutorial git repository. In this repository you will find the following directories which include all relevant files per modelling target (subcomplexes and k.o. ScNPC models)

```
CR_Y_complex
IR_asymmetric_unit_refinement
NR_Y_complex
Nup116delta25C
Nup116delta37C
```

Note: In case you want to directly jump to modelling ScNPCs hence skipping the theory, set ups and general usage instructions then you can continue from the *CR Y-complex modelling*.

Use the **Next** button at the bottom right or menu on the left to navigate through the tutorial.

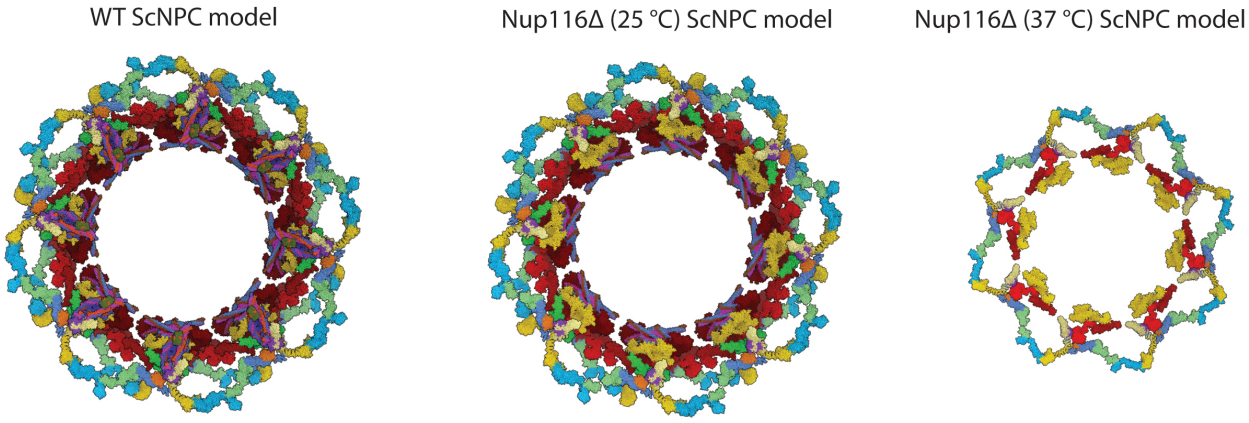
CHAPTER

ONE

ABOUT THE S. CEREVISIAE NUCLEAR PORE COMPLEX (SCNPC)

Nuclear pore complexes (NPCs) are large macromolecular assemblies that fuse the nuclear envelope and facilitate nucleocytoplasmic transport. They are built by around 30 different nucleoporins (Nups) present in multiple copies with respect to the 8-axis rotational symmetry of the complex. The structural architecture of the pore is a three-stacked-rings conformation plus peripheral elements (e.g. nuclear basket) emanating from them (i.e. rings).

We built and published an integrative model of S. cerevisiae NPC in 2020 which included all the major scaffold Nup subcomplexes (i.e. outer rings Y-complexes & Inner ring complex). The model was built based on atomic structures (experimentally determined, homology models, integrative models) and electron microscopy (EM) densities. Spatial restraints were derived from all input data while more restraints were defined as harmomic distance restraints with respect to the available bibliography. The wild-type model of ScNPC that we built served as the basis for building ScNPC models from Nup116 k.o. cells grown under different temperatures.



CHAPTER

TWO

DATA PREPARATION

Note: The data needed to follow the ScNPC modelling tutorial can be obtained from our git repository ScNPC_tutorial git repository.

Create a directory for your project and gather all your data there. For this tutorial the corresponding data files are organized per modelling target (e.g. CR Y-complex, NR Y-complex etc.) and Assembline modelling mode (sub-pipeline) that was used (see following sections of the tutorial). This repository includes the following modelling project directories (repository architecture):

```
scnpc_tutorial/

    # CR Y-complex data
    CR_Y_complex/

        # Electron microscopy maps
        EM/
            P-complex_fit1_0.86_search_100000_correlation_600_0.3_CR_with_Y_no_env_map_
↪best.mrc
            membrane_cerevisiae_no_neg_relative.mrc
            CR_with_Y_no_env_relative.mrc

        #input structures for modeling
        input_PDB/
            ScNup84_437-726_ScNup133C_490_1155_renamed_relative_3I4R.pdb
            ScNup133N_56-480_renamed.pdb
            4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F.pdb
            4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb
            4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb
            4XMM.renamed.noAb_Nup120_2-711_3F7F_updated.pdb

        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta

    # IR asymmetric unit data
    IR_asymmetric_unit_refinement/

        # Electron microscopy maps
        EM/
            IR_relative_no_env.mrc
```

(continues on next page)

(continued from previous page)

```
        #input structures for modeling
        input_PDB/
            ScNup192_NTD_nucl_in.pdb
            ScNup192_NTD_cyt_in.pdb
            ScNup192_CTD_nucl_in.pdb
            ScNup192_CTD_cyt_in.pdb
            ScNup188_NTD_nucl_out.pdb
            ScNup188_NTD_cyt_out.pdb
            ScNup188_CTD_nucl_out.pdb
            ScNup188_CTD_cyt_out.pdb
            ScNup170_NTD_nucl_in.pdb
            ScNup170_NTD_cyt_in.pdb
            ScNup170_CTD_nucl_in.pdb
            ScNup170_CTD_cyt_in.pdb
            ScNup157_NTD_nucl_out.pdb
            ScNup157_NTD_cyt_out.pdb
            ScNup157_CTD_nucl_out.pdb
            ScNup157_CTD_cyt_out.pdb
            ScNsp1_complex_nucl_out.pdb
            ScNsp1_complex_nucl_in.pdb
            ScNsp1_complex_cyt_out.pdb
            ScNsp1_complex_cyt_in.pdb
            ScNic96_CTD_nucl_out.pdb
            ScNic96_CTD_nucl_in.pdb
            ScNic96_CTD_cyt_out.pdb
            ScNic96_CTD_cyt_in.pdb

        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta

    # NR Y-complex data (master dir with refinement & global optimization data)
    NR_Y_complex/

        # Electron microscopy maps for refinement
        EM/
            NR_merged_unerased_tail_relative_clean_v1.3.mrc
            membrane_cerevisiae_no_neg_relative_NR_v3.mrc

        #input structures for refinement
        NR_Y_complex_de_novo_model_PDBs/
            ScNup84_506-726_ScNup133C_882_1155_renamed_relative_3CQC_new_1.pdb
            ScNup133N_56-480_renamed_1.pdb
            ScNup133C_490_881_renamed_relative_3CQC_1.pdb
            4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F_1.pdb
            4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated_1.pdb
            4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated_1.pdb
            4XMM.renamed.noAb_Nup120_2-711_3F7F_updated_1.pdb

        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta
```

(continues on next page)

(continued from previous page)

```
        #NR Y-complex data (dir with global optimization data)
        NR_Y_complex_de_novo_run/

            # Electron microscopy maps for global optimization
            EM/
                NR_merged_unerased_tail_relative_clean_v1.3.mrc
                membrane_cerevisiae_no_neg_relative_NR_v3.mrc

            #input structures for global optimization
            input_PDB/
                ScNup84_506-726_ScNup133C_882_1155_renamed_relative_3CQC_new.pdb
                ScNup133N_56-480_renamed.pdb
                ScNup133C_490_881_renamed_relative_3CQC.pdb
                4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F.pdb
                4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb
                4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb
                4XMM.renamed.noAb_Nup120_2-711_3F7F_updated.pdb

            #FASTA-formatted file with sequences of all subunits
            ScNPC_sequences.fasta

    # Nup116k.o. ScNPC (at 25C) data
    Nup116delta25C/

        # Electron microscopy maps
        EM/
            Spoke_116d_25C_1fold_new.mrc
            membrane_cerevisiae_wt_relative_to_Spoke_116d_25C_1fold_new.mrc

        #input structures for modeling
        input_PDB/
            ScNup84_506-726_ScNup133C_882_1155_renamed_relative_3CQC_new_NR.pdb
            ScNup84_506-726_ScNup133C_882_1155_renamed_relative_3CQC_CR.pdb
            ScNup192_NTD_nucl_in.pdb
            ScNup192_NTD_cyt_in.pdb
            ScNup192_CTD_nucl_in.pdb
            ScNup192_CTD_cyt_in.pdb
            ScNup188_NTD_nucl_out.pdb
            ScNup188_NTD_cyt_out.pdb
            ScNup188_CTD_nucl_out.pdb
            ScNup188_CTD_cyt_out.pdb
            ScNup170_NTD_nucl_in.pdb
            ScNup170_NTD_cyt_in.pdb
            ScNup170_CTD_nucl_in.pdb
            ScNup170_CTD_cyt_in.pdb
            ScNup157_NTD_nucl_out.pdb
            ScNup157_NTD_cyt_out.pdb
            ScNup157_CTD_nucl_out.pdb
            ScNup157_CTD_cyt_out.pdb
            ScNup133N_56-480_renamed_NR.pdb
```

(continues on next page)

(continued from previous page)

```
            ScNup133N_56-480_renamed_CR.pdb
            ScNup133C_490_881_renamed_relative_3CQC_NR.pdb
            ScNup133C_490_881_renamed_relative_3CQC_CR.pdb
            ScNsp1_complex_nucl_out.pdb
            ScNsp1_complex_nucl_in.pdb
            ScNsp1_complex_cyt_out.pdb
            ScNsp1_complex_cyt_in.pdb
            ScNic96_CTD_nucl_out.pdb
            ScNic96_CTD_nucl_in.pdb
            ScNic96_CTD_cyt_out.pdb
            ScNic96_CTD_cyt_in.pdb
            4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F_NR.pdb
            4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F_CR.pdb
            4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated_NR.pdb
            4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated_CR.pdb
            4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated_NR.pdb
            4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated_CR.pdb
            4XMM.renamed.noAb_Nup120_2-711_3F7F_updated_NR.pdb
            4XMM.renamed.noAb_Nup120_2-711_3F7F_updated_CR.pdb

        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta

    # Nup116k.o. ScNPC (at 37C) data
    Nup116delta37C/

        # Electron microscopy maps
        EM/
            Nup116_37C_unmasked.mrc
            NR_116d37.mrc
            membrane_116d37C_relative_to_unmasked.mrc
            membrane_116d37C.mrc

        #input structures for modeling
        input_PDB/
            ScNup84_506-726_ScNup133C_882_1155_renamed_relative_3CQC_new_NR.pdb
            ScNup188_NTD_nucl_out.pdb
            ScNup188_CTD_nucl_out.pdb
            ScNup157_NTD_nucl_out.pdb
            ScNup157_CTD_nucl_out.pdb
            ScNup133N_56-480_renamed_NR.pdb
            ScNup133C_490_881_renamed_relative_3CQC_NR.pdb
            ScNsp1_complex_nucl_out.pdb
            ScNic96_CTD_nucl_out.pdb
            4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F_NR.pdb
            4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated_NR.pdb
            4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated_NR.pdb
            4XMM.renamed.noAb_Nup120_2-711_3F7F_updated_NR.pdb
```

(continues on next page)

(continued from previous page)

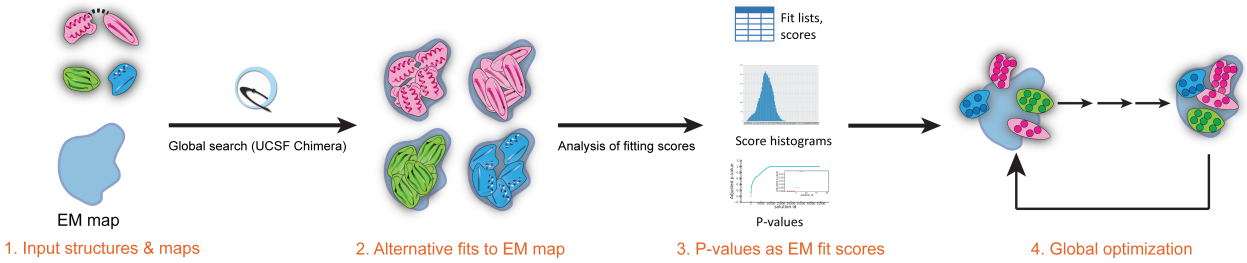
```
        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta
```

CHAPTER

THREE

ABOUT FIT LIBRARIES

Fit libraries are lists of possible positions of your input structures in the EM map (non-redundant rigid body fits). The positions are used in the global optimization step of Assembline to generate a large number of combinations of the pre-calculated fits through a Monte Carlo integrative modelling procedure. Each position (fit) has its associated score and p-value, which are used as *FitRestraints* during the modelling (see figure below).



Note: Read more in the Assembline manual about fit libriaries section.

The fit libraries are typically generated by fitting every input structure into the EM map separately, a procedure handled by the `Efitter` pipeline of Assembline.

Note: This part of the Assembline pipeline (`efitter`) will generate all requested libraries per input structure in a single run (automated) according to user's predefined parameters.

For this tutorial, the fit libraries are already provided in the `systematic_fits/` directory in the folders `CR_Y_complex/` & `NR_Y_complex/` (for the rest of the modelled subcomplexes and Nup116 k.o. models the *global optimization* from Assembline was not used) from our git repository.

The `systematic_fits/` directory is organized as the following in terms of input data:

```
systematic_fits/

    # Electron microscopy maps
    em_maps/

    #input structures for modelling
    PDB/
```

Follow the next steps if you want to generate them yourself using the `Efitter` pipeline (which is a wrapper around UCSF Chimera FitMapTool).

To skip and jump to setting up the integrative modelling sections, move to *Setting up the JSON project file*

CHAPTER

FOUR

SET UP

1. To run `Efitter` you need a parameter file in Python language format that specifies:
   - input structures to fit
   - EM maps
   - fitting parameters
   - optionally, options for execution on a computer cluster
2. For this tutorial, the parameter files are already prepared for both CR and NR Y-complexes in the `scnpc_tutorial/CR_Y_complex/systematic_fits/` and `scnpc_tutorial/NR_Y_complex/NR_Y_complex_de_novo_run/systematic_fits/` directories (from our git repository). The `systematic_fitting_parameters.py` file contains (example parameters file for `CR Y-complex` following):

```
from efitter import Map
from string import Template

method='chimera'
dry_run = False
run_crashed_only = True
master_outdir = './result_fits_chimera'

MAPS = [
    Map('em_maps/CR_with_Y_no_env_relative.mrc', threshold=0.0142,␣
↪resolution=40)
]

models_dir = './PDB'

PDB_FILES = [
    'ScNup133N_56-480_renamed.pdb',
    'ScNup84_437-726_ScNup133C_490_1155_renamed_relative_3I4R.pdb',
    '4XMM.renamed.noAb_Seh1_Nup85_47_544_3F3F.pdb',
    '4XMM.renamed.noAb_Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb',
    '4XMM.renamed.noAb_Nup120_2-711_3F7F_updated.pdb',
    '4XMM.renamed.noAb_Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb'
]

CA_only = False
backbone_only = False
```

(continues on next page)

(continued from previous page)

```
move_to_center = True

fitmap_args = [
    # We suggest to run a small test run with search 100 first
    # {
    #     'template': Template("""
    #         map $map
    #         map_threshold $threshold
    #         fitmap_args resolution $resolution metric cam envelope true␣
↪search 100 placement sr clusterAngle 3 clusterShift 3.0 radius 600 inside␣
↪.30
    #         saveFiles False
    #         """),
    #     'config_prefix': 'test'
    #     },
    # Paramters for main runs (https://www.cgl.ucsf.edu/chimera/docs/
↪UsersGuide/midas/fitmap.html)
    {
        'template': Template("""
            map $map
            map_threshold $threshold
            fitmap_args resolution $resolution metric cam envelope true␣
↪search 100000 placement sr clusterAngle 3 clusterShift 3.0 radius 600␣
↪inside .30
            saveFiles False
            """),
        'config_prefix': 'search100000_metric_cam_rad_600_inside0.3_res_40'
        }
]

# If necessary, edit the below template following specifications of
# cluster_submission_command
# and template for your cluster submission script (using Python string.
↪Template formatting)
cluster_submission_command = 'sbatch'
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --job-name=$job_name
#SBATCH --nodes=1
#SBATCH --mem-per-cpu=m=2000
#SBATCH --time=10:00:00
#SBATCH --error $pdb_outdir/log_err.txt
#SBATCH --output $pdb_outdir/log_out.txt

$cmd
""")
```

Note: Read more about fit libriaries and how to set up the parameters file in the Assembline manual. Remember that you can retrieve and inspect template parameter files in `Assembline/doc/templates` from our Assembline git repository.

3. Based on the above example parameter file, `Efitter` will run six fitting runs - one for each input PDB structure. You can run these as six independent jobs on a computer cluster or six processes on a multicore computer, or one by one on a single core computer.

4. To run on a computer cluster, adjust the `cluster_submission_command` and `run_script_templ` to the queuing system on your cluster (the provided example would work only on a Slurm cluster). It is important that you include `$job_name`, `$pdb_outdir`, and `$cmd` in your script template.

5. To run on a multicore computer, remove `cluster_submission_command` and replace the `run_script_templ` with

```
run_script_templ = Template("""#!/bin/bash
#
echo $job_name
$cmd &>$pdb_outdir/log&
""")
```

Note: In the CR Y-complex parameters file example (from above), more settings than what will actually be used for modelling were left in (all the ones with *False* value, e.g. *dry_run* = *False*) in order to give a broader overview of the available options to users.

CHAPTER

FIVE

RUN

1. Enter the `systematic_fits/` directory either in `scnpc_tutorial/CR_Y_complex/` or `scnpc_tutorial/NR_Y_complex/NR_Y_complex_de_novo_run/`.

   Note: These directories are available and should be retrieved from our git repository ScNPC_tutorial git repository.

2. Run the fitting

   ```
   fit.py systematic_fitting_parameters.py
   ```

   The fitting runs will take two to three hours (~2-3h).

3. The `fit.py` step will create and save all output to `systematic_fits/result_fits_chimera`. So, while in the `systematic_fits/` folder, calculate p-values of resulting fits

   ```
   genpval.py result_fits_chimera
   ```

Note: For this tutorial you do not have to run `Efitter` (i.e. `fit.py` script) again as all the fitting results have been already calculated for you. In case you still want to run the generation of fit libraries then rename the dir `result_fits_chimera/` and while in the `systematic_fits/` dir run the steps from above.

CHAPTER

SIX

ANALYZE FITS

6.1 Check if run correctly

1. If everything run correctly, the `result_fits_chimera` directory should contain the following structure (example for CR Y-complex rigid bodies):

```
result_fits_chimera/
    search100000_metric_cam_rad_600_inside0.3_res_40/
        CR_with_Y_no_env_relative.mrc/
            ScNup84_437-726_ScNup133C_490_1155_renamed_relative_3I4R.pdb/
                Rplots.pdf
                CR_with_Y_no_env_relative.mrc
                histogram.png
                log_err.txt
                log_out.txt
                ori_pdb.pdb
                run.sh
                solutions.csv
                solutions_pvalues.csv
            ScNup133N_56-480_renamed.pdb/
                Rplots.pdf
                CR_with_Y_no_env_relative.mrc
                histogram.png
                log_err.txt
                log_out.txt
                ori_pdb.pdb
                run.sh
                solutions.csv
                solutions_pvalues.csv

            ... and so on
```

2. If the run was successful then you can reconstruct the top fits of each input rigid body (following example for top five fits). Enter the dir `result_fits_chimera/search100000_metric_cam_rad_600_inside0.3_res_40` and run the following step which will generate a dir called `top5`

```
genPDBs_many.py -n5 top5 */*/solutions.csv
```

Warning: The **solutions_pvalues.csv** files contain the fit libraries and must be present to run global optimization.

Note: The pre-calculated output has been simplified (i.e. not all files described in the above architecture have been calculated) for the purpose of this tutorial. Read more on how to analyze the fits.

CHAPTER

SEVEN

SETTING UP THE JSON PROJECT FILE

Note: You can find a ready-to-use JSON file for global optimization (configuration file) named `config.json` in `CR_Y_complex` and `NR_Y_complex/NR_Y_complex_de_novo_run` folders of the ScNPC_tutorial git repository.

7.1 Create Xlink Analyzer project

Here is the instruction how to create the JSON file from scratch.

First, you need to create XlinkAnalyzer project file for your complex

Note: XlinkAnalyzer is used here as a graphical interface for input preparation in Assembline.

Does not matter if you do not have crosslinks - we use XlinkAnalyzer to prepare the input file for modeling.

1. Open Xlink Analyzer
2. In the Xlink Analyzer project Setup menu, define subunits

   Set up the chain IDs as you want them in the final models, they do not have to correspond to chain IDs in your input PDB files.

   Example:

   - `Nup133` - chain IDs: `K`
   - `Nup84` - chain IDs: `L`
   - `Nup120` - chain IDs: `R`

   Note: Remember that you can inspect the exact chain definition for ScNPC subcomplexes in each subcomplex folder (e.g. `CR_Y_complex`) from our ScNPC_tutorial git repository.

3. Load sequence data

   Add sequences of all proteins in a single file in FASTA format

   Here, use the `ScNPC_sequences.fasta` file provided in the tutorial materials and map the sequence names to names of subunits using the Map button

4. Save the JSON file under a name like

```
xla_project.json
```

5. And make a copy (to your modelling directory) that you will modify for modelling

```
cp xla_project.json config.json
```

7.2 Add modelling information to the project file

1. Open `config.json` in a text editor.

Note: The project file is in so-called JSON format

While it may look difficult to edit at the first time, it is actually quite OK with a proper editor (and a bit of practice ;-)

We recommend to use a good editor such as:

- SublimeText
- Atom

At this point, the JSON has the following format:

```
{
    "data": [
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

---

2. Add symmetry (if any, following example is from `IR_asymmetric_unit_refinement/`)
    1. First, specify the series of symmetry related molecules. Here, each of the three subunits is in two symmetrical copies, so we add series as below:

```
"series": [
{
"name": "IR_cyt_outer",
"subunit": "Nic96",
"mode": "auto",
"chainIds": ["A"],
```

(continues on next page)

(continued from previous page)


```
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_cyt_inner",
"subunit": "Nic96",
"mode": "auto",
"chainIds": ["A"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_inner",
"subunit": "Nic96",
"mode": "auto",
"chainIds": ["A"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},
{
"name": "IR_nuc_outer",
"subunit": "Nic96",
"mode": "auto",
"chainIds": ["A"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
},

{
"name": "IR_cyt_outer",
"subunit": "Nup157",
"mode": "auto",
"chainIds": ["D"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_outer",
"subunit": "Nup157",
"mode": "auto",
"chainIds": ["D"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
},
{
"name": "IR_cyt_inner",
"subunit": "Nup170",
"mode": "auto",
"chainIds": ["d"],
```


(continues on next page)

(continued from previous page)

```
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_inner",
"subunit": "Nup170",
"mode": "auto",
"chainIds": ["d"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},

{
"name": "IR_cyt_inner",
"subunit": "Nup192",
"mode": "auto",
"chainIds": ["C"],
"cell_count": 1,
"inipos": "input"
},

{
"name": "IR_nuc_inner",
"subunit": "Nup192",
"mode": "auto",
"chainIds": ["C"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},
{
"name": "IR_cyt_outer",
"subunit": "Nup188",
"mode": "auto",
"chainIds": ["B"],
"cell_count": 1,
"inipos": "input"
},

{
"name": "IR_nuc_outer",
"subunit": "Nup188",
"mode": "auto",
"chainIds": ["B"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
},

{
"name": "IR_cyt_outer",
```

(continues on next page)

(continued from previous page)


```
"subunit": "Nsp1",
"mode": "auto",
"chainIds": ["J"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_cyt_inner",
"subunit": "Nsp1",
"mode": "auto",
"chainIds": ["J"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_inner",
"subunit": "Nsp1",
"mode": "auto",
"chainIds": ["J"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},
{
"name": "IR_nuc_outer",
"subunit": "Nsp1",
"mode": "auto",
"chainIds": ["J"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
},

{
"name": "IR_cyt_outer",
"subunit": "Nup49",
"mode": "auto",
"chainIds": ["I"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_cyt_inner",
"subunit": "Nup49",
"mode": "auto",
"chainIds": ["I"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_inner",
"subunit": "Nup49",
```


(continues on next page)

(continued from previous page)

```
"mode": "auto",
"chainIds": ["I"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},
{
"name": "IR_nuc_outer",
"subunit": "Nup49",
"mode": "auto",
"chainIds": ["I"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
},

{
"name": "IR_cyt_outer",
"subunit": "Nup57",
"mode": "auto",
"chainIds": ["H"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_cyt_inner",
"subunit": "Nup57",
"mode": "auto",
"chainIds": ["H"],
"cell_count": 1,
"inipos": "input"
},
{
"name": "IR_nuc_inner",
"subunit": "Nup57",
"mode": "auto",
"chainIds": ["H"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_inner"
},
{
"name": "IR_nuc_outer",
"subunit": "Nup57",
"mode": "auto",
"chainIds": ["H"],
"cell_count": 1,
"inipos": "input",
"ref_serie": "IR_cyt_outer"
}
]
```

2. Second, define the coordinates of the symmetry axis:

```
"symmetry": {
    "sym_tr3ds": [
        {
            "name": "2-fold",
            "rot": [0.91878368,   0.18447527,  -0.34900635, 0.18489318,  -0.
↪98222332,  -0.03243228, -0.34878513, -0.03473065,  -0.93655898],
            "trans": [113.25839839, 916.45466348, 1106.65193191]
        }
    ],
    "apply_symmetry": [
        {
            "sym": "2-fold",
            "restraint_type": "symmetry_restraint",
            "selectors": [
                {"subunit": "Nic96", "copies": [0, 3]},
                {"subunit": "Nic96", "copies": [1, 2]},
                {"subunit": "Nup188", "copies": [0, 1]},
                {"subunit": "Nup192", "copies": [0, 1]},
                {"subunit": "Nup157", "copies": [0, 1]},
                {"subunit": "Nup170", "copies": [0, 1]}
            ]
        }
    ],
    }
```

3. Add specification of input PDB files

> **Note:** The input structures for the tutorial are in the `in_pdbs/` directories of each subcomplex in our git repository ScNPC_tutorial.

Add them to the JSON like this (following example is from `IR_asymmetric_unit_refinement/`):

```
"data": [
{
    "type": "pdb_files",
    "name": "pdb_files",
    "data": [
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_outer",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNic96_CTD_cyt_out.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
```

(continues on next page)

(continued from previous page)

```
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNic96_CTD_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_inner",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNic96_CTD_nucl_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_outer",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNic96_CTD_nucl_out.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_outer",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNsp1_complex_cyt_out.pdb"
                        },
                        {
                        "serie": "IR_cyt_outer",
                        "name": "Nsp1",
                        "subunit": "Nsp1",
                        "chain_id": "J",
                        "filename": "input_PDB/ScNsp1_complex_cyt_out.pdb"
                        },
                        {
```

(continues on next page)

(continued from previous page)

```
                        "serie": "IR_cyt_outer",
                        "name": "Nup57",
                        "subunit": "Nup57",
                        "chain_id": "H",
                        "filename": "input_PDB/ScNsp1_complex_cyt_out.pdb"
                        },
                        {
                        "serie": "IR_cyt_outer",
                        "name": "Nup49",
                        "subunit": "Nup49",
                        "chain_id": "I",
                        "filename": "input_PDB/ScNsp1_complex_cyt_out.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nic96",
                        "subunit": "Nic96",
                        "chain_id": "A",
                        "filename": "input_PDB/ScNsp1_complex_cyt_in.pdb"
                        },
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nsp1",
                        "subunit": "Nsp1",
                        "chain_id": "J",
                        "filename": "input_PDB/ScNsp1_complex_cyt_in.pdb"
                        },
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup57",
                        "subunit": "Nup57",
                        "chain_id": "H",
                        "filename": "input_PDB/ScNsp1_complex_cyt_in.pdb"
                        },
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup49",
                        "subunit": "Nup49",
                        "chain_id": "I",
                        "filename": "input_PDB/ScNsp1_complex_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
```

(continues on next page)

(continued from previous page)

```
                    "serie": "IR_nuc_inner",
                    "name": "Nic96",
                    "subunit": "Nic96",
                    "chain_id": "A",
                    "filename": "input_PDB/ScNsp1_complex_nucl_in.pdb"
                    },
                    {
                    "serie": "IR_nuc_inner",
                    "name": "Nsp1",
                    "subunit": "Nsp1",
                    "chain_id": "J",
                    "filename": "input_PDB/ScNsp1_complex_nucl_in.pdb"
                    },
                    {
                    "serie": "IR_nuc_inner",
                    "name": "Nup57",
                    "subunit": "Nup57",
                    "chain_id": "H",
                    "filename": "input_PDB/ScNsp1_complex_nucl_in.pdb"
                    },
                    {
                    "serie": "IR_nuc_inner",
                    "name": "Nup49",
                    "subunit": "Nup49",
                    "chain_id": "I",
                    "filename": "input_PDB/ScNsp1_complex_nucl_in.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nic96",
                    "subunit": "Nic96",
                    "chain_id": "A",
                    "filename": "input_PDB/ScNsp1_complex_nucl_out.pdb"
                    },
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nsp1",
                    "subunit": "Nsp1",
                    "chain_id": "J",
                    "filename": "input_PDB/ScNsp1_complex_nucl_out.pdb"
                    },
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup57",
                    "subunit": "Nup57",
                    "chain_id": "H",
                    "filename": "input_PDB/ScNsp1_complex_nucl_out.pdb"
```

(continues on next page)

(continued from previous page)

```
                    },
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup49",
                    "subunit": "Nup49",
                    "chain_id": "I",
                    "filename": "input_PDB/ScNsp1_complex_nucl_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_cyt_outer",
                    "name": "Nup188",
                    "subunit": "Nup188",
                    "chain_id": "B",
                    "filename": "input_PDB/ScNup188_NTD_cyt_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup188",
                    "subunit": "Nup188",
                    "chain_id": "B",
                    "filename": "input_PDB/ScNup188_NTD_nucl_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_cyt_outer",
                    "name": "Nup188",
                    "subunit": "Nup188",
                    "chain_id": "B",
                    "filename": "input_PDB/ScNup188_CTD_cyt_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup188",
```

(continues on next page)

(continued from previous page)

```
                        "subunit": "Nup188",
                        "chain_id": "B",
                        "filename": "input_PDB/ScNup188_CTD_nucl_out.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup192",
                        "subunit": "Nup192",
                        "chain_id": "C",
                        "filename": "input_PDB/ScNup192_NTD_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_inner",
                        "name": "Nup192",
                        "subunit": "Nup192",
                        "chain_id": "C",
                        "filename": "input_PDB/ScNup192_NTD_nucl_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup192",
                        "subunit": "Nup192",
                        "chain_id": "C",
                        "filename": "input_PDB/ScNup192_CTD_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_inner",
                        "name": "Nup192",
                        "subunit": "Nup192",
                        "chain_id": "C",
                        "filename": "input_PDB/ScNup192_CTD_nucl_in.pdb"
                        }
```

(continues on next page)

(continued from previous page)

```
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_cyt_outer",
                    "name": "Nup157",
                    "subunit": "Nup157",
                    "chain_id": "D",
                    "filename": "input_PDB/ScNup157_NTD_cyt_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup157",
                    "subunit": "Nup157",
                    "chain_id": "D",
                    "filename": "input_PDB/ScNup157_NTD_nucl_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_cyt_outer",
                    "name": "Nup157",
                    "subunit": "Nup157",
                    "chain_id": "D",
                    "filename": "input_PDB/ScNup157_CTD_cyt_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
                "components": [
                    {
                    "serie": "IR_nuc_outer",
                    "name": "Nup157",
                    "subunit": "Nup157",
                    "chain_id": "D",
                    "filename": "input_PDB/ScNup157_CTD_nucl_out.pdb"
                    }
                ]
            },
            {
                "foreach_copy": true,
```

(continues on next page)

(continued from previous page)

```
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup170",
                        "subunit": "Nup170",
                        "chain_id": "d",
                        "filename": "input_PDB/ScNup170_NTD_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_inner",
                        "name": "Nup170",
                        "subunit": "Nup170",
                        "chain_id": "d",
                        "filename": "input_PDB/ScNup170_NTD_nucl_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_cyt_inner",
                        "name": "Nup170",
                        "subunit": "Nup170",
                        "chain_id": "d",
                        "filename": "input_PDB/ScNup170_CTD_cyt_in.pdb"
                        }
                    ]
                },
                {
                    "foreach_copy": true,
                    "components": [
                        {
                        "serie": "IR_nuc_inner",
                        "name": "Nup170",
                        "subunit": "Nup170",
                        "chain_id": "d",
                        "filename": "input_PDB/ScNup170_CTD_nucl_in.pdb"
                        }
                    ]
                }
        ]
```

The `foreach_serie` and `foreach_copy` indicate the given PDB file specification will be applied to each serie with this subunit and for each copy within the series.

All PDB selections within the same `components` block will be grouped into a rigid body, unless a separate `rigid_bodies` block is specified and `add_rbs_from_pdbs` is set to `False` in *Setting up*

the parameter file

4. Add pointers to fit libraries and sequence FASTA file (following example is from `CR_Y_complex/`)

First add the pointers to input fit libraries

```
        "data": [
        {
            "components": [
                {
                "name": "Nup133",
                "subunit": "Nup133",
                "chain_id": "K",
                "filename": "input_PDB/ScNup133N_56-480_
↪renamed.pdb"
                }
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/ScNup133N_56-480_renamed.pdb/
↪solutions_pvalues.csv",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 10000, // optional, by default␣
↪all fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        },
        {
            "components": [
                {
                "name": "Nup84",
                "subunit": "Nup84",
                "chain_id": "L",
                "filename": "input_PDB/ScNup84_437-726_
↪ScNup133C_490_1155_renamed_relative_3I4R.pdb"
                },
                {
                "name": "Nup133",
                "subunit": "Nup133",
                "chain_id": "K",
                "filename": "input_PDB/ScNup84_437-726_
↪ScNup133C_490_1155_renamed_relative_3I4R.pdb"
                }
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/ScNup84_437-726_ScNup133C_490_
↪1155_renamed_relative_3I4R.pdb/solutions_pvalues.csv",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 10000, // optional, by default␣
↪all fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        },
```

(continues on next page)

(continued from previous page)

```
        {
            "components": [
                {
                "name": "Nup145c",
                "subunit": "Nup145c",
                "chain_id": "M",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb"
                },
                {
                "name": "Sec13",
                "subunit": "Sec13",
                "chain_id": "N",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb"
                },
                {
                "name": "Nup84",
                "subunit": "Nup84",
                "chain_id": "L",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup145c_149-550_Sec13_Nup84_7-436_3IKO_updated.pdb"
                }
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/4XMM.renamed.noAb_Nup145c_149-
↪550_Sec13_Nup84_7-436_3IKO_updated.pdb/solutions_pvalues.csv
↪",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 100, // optional, by default all␣
↪fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        },
        {
            "components": [

                {
                "name": "Seh1",
                "subunit": "Seh1",
                "chain_id": "O",
                "filename": "input_PDB/4XMM.renamed.noAb_Seh1_
↪Nup85_47_544_3F3F.pdb"
                },
                {
                "name": "Nup85",
                "subunit": "Nup85",
                "chain_id": "P",
                "filename": "input_PDB/4XMM.renamed.noAb_Seh1_
↪Nup85_47_544_3F3F.pdb"
                }
```

(continues on next page)

(continued from previous page)

```
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/4XMM.renamed.noAb_Seh1_Nup85_
↪47_544_3F3F.pdb/solutions_pvalues.csv",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 10000, // optional, by default␣
↪all fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        },
        {
            "components": [
                {
                "name": "Nup85",
                "subunit": "Nup85",
                "chain_id": "P",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb"
                },
                {
                "name": "Nup145c",
                "subunit": "Nup145c",
                "chain_id": "M",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb"
                },
                {
                "name": "Nup120",
                "subunit": "Nup120",
                "chain_id": "R",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup120_715-1036_Nup85_553-744_Nup145c_92-99_554-712_
↪remaining_updated.pdb"
                }
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/4XMM.renamed.noAb_Nup120_715-
↪1036_Nup85_553-744_Nup145c_92-99_554-712_remaining_updated.
↪pdb/solutions_pvalues.csv",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 10000, // optional, by default␣
↪all fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        },
        {
            "components": [
                {
```

(continues on next page)

(continued from previous page)

```
                "name": "Nup120",
                "subunit": "Nup120",
                "chain_id": "R",
                "filename": "input_PDB/4XMM.renamed.noAb_
↪Nup120_2-711_3F7F_updated.pdb"
                }
            ],
            "positions": "systematic_fits/result_fits_chimera/
↪CR_with_Y_no_env_relative.mrc/4XMM.renamed.noAb_Nup120_2-
↪711_3F7F_updated.pdb/solutions_pvalues.csv",
            "positions_type": "chimera", // optional, chimera␣
↪is default
            "max_positions": 10000, // optional, by default␣
↪all fits are read
            "positions_score": "log_BH_adjusted_pvalues_one_
↪tailed"
        }
]
```

Now add the fasta sequence pointer and maping of the subunits

```
"fileGroup": {
    "files": [
        "ScNPC_sequences.fasta"
    ]
},
"mapping": {
    "sp|P35729|NU120_YEAST Nucleoporin NUP120 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP120 PE=1␣
↪SV=1":["Nup120"],
    "sp|P49687|NU145C Nucleoporin NUP145 OS=Saccharomyces cerevisiae␣
↪(strain ATCC 204508 / S288c) OX=559292 GN=NUP145 PE=1 SV=1":[
↪"Nup145c"],
    "sp|P49687|NU145N Nucleoporin NUP145 OS=Saccharomyces cerevisiae␣
↪(strain ATCC 204508 / S288c) OX=559292 GN=NUP145 PE=1 SV=1":[
↪"Nup145n"],
    "sp|P46673|NUP85_YEAST Nucleoporin NUP85 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP85 PE=1␣
↪SV=1":["Nup85"],
    "sp|P53011|SEH1_YEAST Nucleoporin SEH1 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=SEH1 PE=1 SV=1
↪":["Seh1"],
    "sp|Q04491|SEC13_YEAST Protein transport protein SEC13␣
↪OS=Saccharomyces cerevisiae (strain ATCC 204508 / S288c) OX=559292␣
↪GN=SEC13 PE=1 SV=1":["Sec13"],
    "sp|P52891|NUP84_YEAST Nucleoporin NUP84 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP84 PE=1␣
↪SV=1":["Nup84"],
    "sp|P36161|NU133_YEAST Nucleoporin NUP133 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP133 PE=1␣
↪SV=1":["Nup133"],
    "sp|P52593|NU188_YEAST Nucleoporin NUP188 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP188 PE=1␣
↪SV=1":["Nup188"],
```

(continues on next page)

(continued from previous page)

```
    "sp|P47054|NU192_YEAST Nucleoporin NUP192 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP192 PE=1␣
↪SV=1":["Nup192"],
    "sp|P40064|NU157_YEAST Nucleoporin NUP157 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP157 PE=1␣
↪SV=1":["Nup157"],
    "sp|P38181|NU170_YEAST Nucleoporin NUP170 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP170 PE=1␣
↪SV=1":["Nup170"],
    "sp|P34077|NIC96_YEAST Nucleoporin NIC96 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NIC96 PE=1␣
↪SV=2":["Nic96"],
    "sp|Q03790|NUP53_YEAST Nucleoporin NUP53 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP53 PE=1␣
↪SV=1":["Nup53"],
    "sp|Q05166|NUP59_YEAST Nucleoporin ASM4 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=ASM4 PE=1 SV=1
↪":["Nup59"],
    "sp|P14907|NSP1_YEAST Nucleoporin NSP1 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NSP1 PE=1 SV=1
↪":["Nsp1"],
    "sp|P48837|NUP57_YEAST Nucleoporin NUP57 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP57 PE=1␣
↪SV=1":["Nup57"],
    "sp|Q02199|NUP49_YEAST Nucleoporin NUP49/NSP49 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP49 PE=1␣
↪SV=1":["Nup49"],
    "sp|P40368|NUP82_YEAST Nucleoporin NUP82 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP82 PE=1␣
↪SV=2":["Nup82"],
    "sp|P40477|NU159_YEAST Nucleoporin NUP159 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP159 PE=1␣
↪SV=1":["Nup159"],
    "sp|Q02630|NU116_YEAST Nucleoporin NUP116/NSP116␣
↪OS=Saccharomyces cerevisiae (strain ATCC 204508 / S288c) OX=559292␣
↪GN=NUP116 PE=1 SV=2":["Nup116"],
    "sp|Q02629|NU100_YEAST Nucleoporin NUP100/NSP100␣
↪OS=Saccharomyces cerevisiae (strain ATCC 204508 / S288c) OX=559292␣
↪GN=NUP100 PE=1 SV=1":["Nup100"],
    "sp|P32500|NDC1_YEAST Nucleoporin NDC1 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NDC1 PE=1 SV=1
↪":["Ndc1"],
    "sp|P39685|PO152_YEAST Nucleoporin POM152 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=POM152 PE=1␣
↪SV=1":["Pom152"],
    "sp|Q02455|MLP1_YEAST Protein MLP1 OS=Saccharomyces cerevisiae␣
↪(strain ATCC 204508 / S288c) OX=559292 GN=MLP1 PE=1 SV=2":["Mlp1"],
    "sp|P40457|MLP2_YEAST Protein MLP2 OS=Saccharomyces cerevisiae␣
↪(strain ATCC 204508 / S288c) OX=559292 GN=MLP2 PE=1 SV=1":["Mlp2"],
    "sp|P49686|NUP42_YEAST Nucleoporin NUP42 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP42 PE=1␣
↪SV=1":["Nup42"],
```

(continues on next page)

(continued from previous page)

```
    "sp|Q12315|GLE1_YEAST Nucleoporin GLE1 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=GLE1 PE=1 SV=1
↪":["Gle1"],
    "sp|P40066|GLE2_YEAST Nucleoporin GLE2 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=GLE2 PE=1 SV=1
↪":["Gle2"],
    "sp|Q12445|POM34_YEAST Nucleoporin POM34 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=POM34 PE=1␣
↪SV=1":["Pom34"],
    "sp|Q02647|DYL1_YEAST Dynein light chain 1, cytoplasmic␣
↪OS=Saccharomyces cerevisiae (strain ATCC 204508 / S288c) OX=559292␣
↪GN=DYN2 PE=1 SV=1":["Dyn2"],
    "sp|P39705|NUP60_YEAST Nucleoporin NUP60 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP60 PE=1␣
↪SV=1":["Nup60"],
    "sp|P32499|NUP2_YEAST Nucleoporin NUP2 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP2 PE=1 SV=2
↪":["Nup2"],
    "sp|P20676|NUP1_YEAST Nucleoporin NUP1 OS=Saccharomyces␣
↪cerevisiae (strain ATCC 204508 / S288c) OX=559292 GN=NUP1 PE=1 SV=1
↪":["Nup1"],
    "sp|Q09747|DBP5_SCHPO ATP-dependent RNA helicase dbp5␣
↪OS=Schizosaccharomyces pombe (strain 972 / ATCC 24843) OX=284812␣
↪GN=dbp5 PE=1 SV=1":["Dbp5"]
},
"name": "sequences",
"type": "sequences"
    }
```

5. Add spatial restraints (following example is from `IR_asymmetric_unit_refinement/`)

```
    {
"active": true,
"type": "em_map",
"name": "FitRestraint_IR_relative_no_env",
"em_restraint_type": "FitRestraint",
"filename": "EM/IR_relative_no_env.mrc",
"threshold": 0.0156,
"voxel_size": 6.74,
"resolution": 20,
"weight": 10000,
"first_copy_only": "false",
"repr_resolution": 10,
"optimized_components": [
    {"name": "Nic96", "subunit": "Nic96"},
    {"name": "Nup188", "subunit": "Nup188"},
    {"name": "Nup192", "subunit": "Nup192"},
    {"name": "Nup157", "subunit": "Nup157"},
    {"name": "Nup170", "subunit": "Nup170"},
    {"name": "Nsp1", "subunit": "Nsp1"},
    {"name": "Nup57", "subunit": "Nup57"},
    {"name": "Nup49", "subunit": "Nup49"}
```

(continues on next page)

(continued from previous page)

```
]
    }
```

And that's it!

CHAPTER

EIGHT

SETTING UP THE PARAMETER FILE

You can find a ready-to-use parameter file for global optimization named

```
params.py
```

in `CR_Y_complex` and `NR_Y_complex/NR_Y_complex_de_novo_run` folders of our ScNPC_tutorial git repository.

The file is in Python format and contains the following content (example from `CR_Y_complex`):

```
from string import Template

# modelling protocol general settings

protocol = 'denovo_MC-SA'

SA_schedule = [
    (100,   10000),
    (10,    10000),
    (1,     10000)
]

do_ini_opt = False

# output settinngs (how often and which kind of ouput to produce)
traj_frame_period = 100
print_frame_period = traj_frame_period

print_log_scores_to_files = False
print_log_scores_to_files_frame_period = 10

print_total_score_to_files = False
print_total_score_to_files_frame_period = 10

#system states and representation resolution settings
states = 1
struct_resolutions = [1,10]
add_missing = False
missing_resolution = 1

add_rbs_from_pdbs = True

#following settings are for restratins and weights
```

(continues on next page)

(continued from previous page)

```
connectivity_restraint_weight = 1.
max_conn_gap = None
conn_first_copy_only = False

rb_max_rot = 2
rb_max_trans = 3

add_symmetry_constraints = False
add_symmetry_restraints = False

add_parsimonious_states_restraints = False
parsimonious_states_weight = 10
parsimonious_states_distance = 0.0

add_xlink_restraints = False

add_em_restr = False

em_restraints = [
    {
        'name': 'em_restraints_highres',
        'type': 'FitRestraint',
        'weight': 100,
        'repr_resolution': 1
    },
    {
        'name': 'em_restraints_highres1',
        'type': 'EnvelopePenetrationRestraint',
        'weight': 50,
        'repr_resolution': 1
    },
    {
        'name': 'em_restraints_lowres',
        'type': 'EnvelopePenetrationRestraint',
        'weight': 50,
        'repr_resolution': 10
    },
]


discrete_restraints_weight=1000 # weight for the restraint derived from the␣
↪precomputed fits

ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    },
    {
        'name': 'ev_restraints_highres',
        'weight': 10,
```

(continues on next page)

(continued from previous page)

```
        'repr_resolution': 1
    }
]

# here is an example of how you could define custom restraints that have been␣
↪defined in config.json
def create_custom_restraints(r):
    em_restraints = r.add_em_restraints(
                        weight=1000,
                        first_copy_only=False,
                        resolution=10)

    restraints = {}
    for i, restr in enumerate(em_restraints):
        restraints[str(restr).replace('"','')+str(i)] = [restr]

    return restraints

# scoring functions definition and scoring terms to be included
scoring_functions = {
    'score_func_ini_opt': {
        'restraints': [
            'discrete_restraints',
            'conn_restraints',
            'ev_restraints_lowres',
            "ExcludeMapRestraint0",
            "ExcludeMapRestraint1"
        ]
    },
    'score_func_highres': {
        'restraints': [
            'discrete_restraints',
            #'xlink_restraints',
            'conn_restraints',
            'ev_restraints_highres'
        ]
    },
    'score_func_lowres': {
        'restraints': [
            'discrete_restraints',
            'conn_restraints',
            'ev_restraints_lowres',
            "ExcludeMapRestraint0",
            "ExcludeMapRestraint1"
        ]
    },
    'score_func_for_CG': {
        'restraints': [
            #'xlink_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
```

(continues on next page)

(continued from previous page)

```
    },
    'score_func_precond': {
        'restraints': [
            'discrete_restraints',
            #'xlink_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
    }
}

score_func = 'score_func_lowres'
score_func_for_CG = 'score_func_for_CG'
score_func_ini_opt = 'score_func_ini_opt'
score_func_preconditioned_mc = 'score_func_precond'

# SLURM template, adjust to your cluster environment
ntasks = 1
cluster_submission_command = 'sbatch'
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=10:00:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
srun hostname

$cmd

wait
""")
```

To generate this file yourself from scratch:

1. Download the template params_combinations.py file
2. Open the file in an editor to adjust the parameters as in the provided example.

   See parameters for explanations and description of available parameters.

CHAPTER

NINE

RUN

1. In order to run global optimization with Assembline, activate the environment before using the software by

```
source activate Assembline
```

   or depending on your computer setup:

```
conda activate Assembline
```

2. Enter the main project directory (e.g. `scnpc_tutorial/CR_Y_complex`)
3. Run a single run for testing

```
assembline.py --traj --models --prefix 0000000 -o out config.json params.py␣
↪&>log&
```

   Output is saved to `out` directory specified by `-o` option.

4. Run N amount of runs to build N number of models (following example from CR Y-complex)

   - Method 1: Submit all runs to the computer cluster or run on a workstation in chunks of N according to the number of processors:

```
assembline.py --traj --models -o out --multi --start_idx 0 --
↪njobs 20000 config.json params.py
```

     – on a cluster, this will submit 20000 modeling jobs in the queue, each job leading to one model (if ntasks in params.py is set to 1)
     – if ntasks params.py is N, it will run submit 20000/N cluster jobs, each running N modeling jobs
     – on a multicore computer, it will run ntasks at a time, and keep running until all 20000 jobs are done.

     Note: The number of processors or cluster submission commands and templates are specified in `params.py`

   - Method 2: Dynamically adjust the number of concurrent runs (e.g. to not to overload a cluster or annoy other users):

> **Warning:** The following works out of the box only on the EMBL cluster. To adjust to your cluster, modify the `assembline.py` for your cluster environment following the guidelines in the script.

```
assembline.py --traj --models --multi --daemon --min_concurrent_
↪jobs 500 --max_concurrent_jobs 5000 -o out --start_idx 0 --
↪njobs 20000 config.json params.py &>log&
```

- Method 3: If none of the above solutions work for you, you could probably submit multiple jobs manually using a mad shell loop e.g. on a computer cluster with the Slurm queuing system:

```
for i in $(seq -f "%07g" 0 19999)
    do
        srun assembline.py  --traj --models --prefix $i -o out␣
↪config.json params.py &>log&
    done
```

  Just remember to make the `prefix` unique for every run.

Read more about how to run many runs on different platforms in the manual

CHAPTER

TEN

ANALYZE

Enter the output directory.

```
cd out
```

10.1 Extract scores

```
extract_scores.py
```

This should create a couple of files including all_scores_sorted_uniq.csv

10.2 Create a CIF file of the top N models

```
rebuild_atomic.py --top 10 --project_dir <full path to the original project directory, e.
↪g. CR_Y_complex> config.json all_scores_sorted_uniq.csv
```

10.3 Assess sampling exhaustiveness

Run sampling performance analysis with imp-sampcon tool (described by Viswanath et al. 2017)

Warning: For the global optimization, the sampling exhaustiveness is not always applicable. For some cases, the optimization at this stage can actually work so well that it leads to all or most models being the same, resulting in very few clusters. In such cases, the sampling is exhaustive under the assumptions in the json but the estimation of sampling precision won't be possible. In such cases we recommend to intensively refine (e.g. with high initial temperatures in simulated annealing) the top (or all models) to create a diverse set of models for analysis.

1. Prepare the `density.txt` file

   ```
   create_density_file.py --project_dir ../ config.json --by_rigid_body
   ```

   Note: Example `density.txt` file is provided in `CR_Y_complex/`

2. Run `setup_analysis.py` script to prepare input files for the sampling exhaustiveness analysis.

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt --score_
↪thresh <score to use as threshold>
```

`--score_thresh` is optional and used to filter out some rare very poorly scoring models (the threshold can be adjusted based on the `scores.pdf` generated above)

Note: For further descriptions of settings for `setup_analysis` please see Sampling exhaustiveness and precision with Assembline

---

3. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the actual analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

---

4. In the output you will get, among other files:

   - `<prefix for output files>.Sampling_Precision_Stats.txt`

     Estimation of the sampling precision.

   - Clusters obtained after clustering at the determined (By imp-sampcon) sampling precision in directories and files starting from `cluster` in their names, containing information about the models in the clusters and cluster localization densities

   - `<prefix for output files>.Cluster_Precision.txt` listing the precision for each cluster

   - PDF files with plots with the results of exhaustiveness tests

   See Viswanath et al. 2017 for detailed explanation of these concepts.

5. Optimize the plots

   The fonts and value ranges in X and Y axes in the default plots from `imp_sampcon exhaust` are frequently not optimal. For this you have to adjust them manually.

   1. Copy the original `gnuplot` scripts to the current `analysis` directory by executing:

      ```
      copy_sampcon_gnuplot_scripts.py
      ```

      This will copy four scripts to the current directory:

      - `Plot_Cluster_Population.plt` for the `<prefix for output files>.Cluster_Population.pdf` plot

- `Plot_Convergence_NM.plt` for the `<prefix for output files>.ChiSquare.pdf` plot
- `Plot_Convergence_SD.plt` for the `<prefix for output files>.Score_Dist.pdf` plot
- `Plot_Convergence_TS.plt` for the `<prefix for output files>.Top_Score_Conv.pdf` plot

2. Edit the scripts to adjust according to your liking
3. Run the scripts again:

```
gnuplot -e "sysname='<prefix for output files>'" Plot_Cluster_
↪Population.plt
gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_NM.plt
gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_SD.plt
gnuplot -e "sysname='<prefix for output files>'" Plot_Convergence_TS.plt
```

6. Extract cluster models for visualization

```
extract_cluster_models.py \
    --project_dir <full path to the original project directory> \
    --outdir <cluster directory> \
    --ntop <number of top models to extract> \
    --scores <path to the score CSV file used as input for analysis> \
    Identities_A.txt Identities_B.txt <list of cluster models> <path to the␣
↪json>
```

For example, to extract the 5 top scoring models from cluster 0:

```
extract_cluster_models.py \
    --project_dir ../../ \
    --outdir cluster.0/ \
    --ntop 5 \
    --scores ../all_scores.csv \
    Identities_A.txt Identities_B.txt cluster.0.all.txt ../config.json
```

The models are saved in the CIF format to `cluster.0` directory

7. If you want to re-cluster at a specific threshold (e.g. to get bigger clusters), you can do:

```
mkdir recluster
cd recluster/
cp ../Distances_Matrix.data.npy .
cp ../*ChiSquare_Grid_Stats.txt .
cp ../*Sampling_Precision_Stats.txt .
imp_sampcon exhaust -n <prefix for output files> \
--rmfA ../sample_A/sample_A_models.rmf3 \
--rmfB ../sample_B/sample_B_models.rmf3 \
--scoreA ../scoresA.txt --scoreB ../scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c 4 \
-gp \
--skip \
--cluster_threshold <float greater than already calculated precision>
```

CHAPTER

ELEVEN

SETTING UP THE JSON PROJECT FILE

You can find a ready-to-use JSON template file named `config.json` in the NR_Y_complex/ directory of our Sc-NPC_tutorial git repository.

Note: A semi-automated method to generate modelling templates for refinement can be found in Elongator complex tutorial.

CHAPTER

TWELVE

SETTING UP THE PARAMETER FILE

Note: You can find a ready-to-use parameter file named

```
params.py
```

in the `IR_asymmetric_unit_refinement/` directory in our git repository ScNPC_tutorial git repository.

The file is in the Python format and contains the following content:

```
from string import Template

protocol = 'refine'

        SA_schedule = [
            (100,   5000),
        ]

        do_ini_opt = False

        traj_frame_period = 100
        print_frame_period = traj_frame_period

        print_log_scores_to_files = False
        print_log_scores_to_files_frame_period = 10

        print_total_score_to_files = False
        print_total_score_to_files_frame_period = 10

        states = 1
        struct_resolutions = [1,10]
        add_missing = False
        missing_resolution = 1

        add_rbs_from_pdbs = True


        connectivity_restraint_weight = 1.
        max_conn_gap = None
        conn_first_copy_only = False
```

(continues on next page)

(continued from previous page)

```
rb_max_rot = 2
rb_max_trans = 3

add_symmetry_constraints = False
add_symmetry_restraints = True
symmetry_restraints_weight = 1

add_xlink_restraints = False

add_em_restr = True

discrete_restraints_weight=10000

ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    }
]

scoring_functions = {
    'score_func_lowres': {
        'restraints': [
            'conn_restraints',
            'ev_restraints_lowres',
            'FitRestraint_IR_relative_no_env',
            'sym_restraints'
        ]
    }
}

score_func = 'score_func_lowres'

# SLURM template, adjust to your cluster environment
ntasks = 1
cluster_submission_command = 'sbatch'
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=10:00:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
srun hostname

$cmd

wait
```

(continues on next page)

(continued from previous page)

```
        """)
```

To generate this file yourself from scratch:

1. Copy the template parameter file from the manual (i.e. Assembline repository) to the current directory:

```
cp doc/templates/params_refine.py .
```

2. Open the file in an editor to adjust the parameters

CHAPTER

THIRTEEN

RUN

The following steps show roughly how to refine models (either top models from global fitting or locally optimize rigid bodies in general).

1. I you haven't yet, activate your modelling virtual environment before using the Assembline software by

   ```
   source activate Assembline
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline
   ```

2. Enter the main project directory (e.g. `IR_asymmetric_unit_refinement`) and create `params.py` and `config.json` files according to Assembline manual. While in the main project directory run the following

   ```
   assembline.py --traj --models -o out --multi --start_idx 0 --njobs 500␣
   ↪config.json params.py
   ```

Note: In order to set up and run refinement for multiple models then instruct the Elongator manual and Assembline manual.

CHAPTER

FOURTEEN

ANALYZE

1. Enter the refinement output directory and extract score files (following example for `IR_asymmetric_unit_refinement`):

```
cd IR_asymmetric_unit_refinement/out
extract_scores.py
```

2. **Generate PDB/CIF files (e.g. 10 top models in the following case)**

   Note: Note that no models have been precaclulated for this tutorial therefore to run the above step you need to run modelling with Assembline (follow next sections with modelling of ScNPC subcomplexes e.g. CR Y-complex).

   Note: `--project_dir` is necessary if you use relative paths in the JSON project file. To generate top models from multiple refinement runs inspect the Elongator tutorial and Assembline manual.

   rebuild_atomic.py –top 10 –project_dir <full path to the original project directory> config.json all_scores_sorted_uniq.csv

3. Assess sampling exhaustiveness by following the commands and recommendations in ScNPC global optimization analysis.

CHAPTER

FIFTEEN

CR Y-COMPLEX MODELLING

For modelling the CR Y-complex the calculation of fit libraries and global optimization with Assembline will be used. The dir `scnpc_tutorial/CR_Y_complex` includes source code files, input files and precalculated modelling results for the CR Y-complex:

```
- parameters file & configuration file for global optimization of wt CR Y-complex
- sequence fasta file, input_PDB, EM (input data)
- CR_Y_complex_final_model.pdb, out (output modelling results)
- systematic_fits (directory):

    - parameters file for systematic fitting
    - em_maps, PDB (input data)
    - result_fits_chimera (output fitting results)
```

1. First activate your virtual environment and enter the `CR_Y_complex/systematic_fits` dir

   ```
   source activate Assembline

   cd CR_Y_complex/systematic_fits/
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline

   cd CR_Y_complex/systematic_fits/
   ```

2. Run the generation of fit libraries for CR Y-complex rigid bodies (results calculated in dir `systematic_fits/result_fits_chimera/CR_with_Y_no_env_relative.mrc/`)

   ```
   fit.py systematic_fitting_parameters.py
   ```

3. The fit libraries have been precalculated and analysed in dir `systematic_fits/result_fits_chimera/CR_with_Y_no_env_relative.mrc/`. To analyse the fit results on your own run the following while in the systematic_fits/ dir

   ```
   genpval.py result_fits_chimera
   ```

4. To generate the top five fits of each input rigid body (i.e. top five fits from each fit library) run the following

   ```
   #upon successful run of fit.py and genpval.py
   cd result_fits_chimera/search100000_metric_cam_rad_600_inside0.3_res_40

   genPDBs_many.py -n5 top5 */*/solutions.csv
   ```

5. After completing the calculation of fit libraries (or use the precalculated results) enter again the main project dir (i.e. CR_Y_complex) and run the global optimization

```
cd CR_Y_complex

# this will run 20000 global optimization modelling runs on a slurm cluster␣
↪(for options/parameters or local runs inspect the Assembline manual)
assembline.py --traj --models -o out --multi --start_idx 0 --njobs 20000␣
↪config.json params.py
```

Note: There is already an output dir `CR_Y_complex/out` so in case you want to run the modelling then rename the `out/` dir as it will be overwritten from the run above

6. Enter `out/` dir, generate output scoring lists and rebuild atomic structures of models

```
cd out

extract_scores.py #this should create a couple of files including all_
↪scores_sorted_uniq.csv

rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
↪directory CR_Y_complex> config.json all_scores_sorted_uniq.csv
```

7. While in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is provided in the `CR_Y_complex/out`. To generate it yourself please inspect Assembline analysis section.

8. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

CHAPTER

SIXTEEN

IR ASYMMETRIC UNIT MODELLING

For modelling the IR asymmetric unit the local rigid body refinement method from Assembline will be used. The dir `scnpc_tutorial/IR_asymmetric_unit_refinement` includes source code files, input files and precalculated modelling results for the IR unit:

```
- parameters file & configuration file for 'refinement' integrative modelling of wt IR␣
↪unit
- sequence fasta file, input_PDB, EM (input data)
- IR_unit_final_refined_model, out (output modelling results)
```

1. First activate your virtual environment and enter the `IR_asymmetric_unit_refinement` dir

```
source activate Assembline

cd IR_asymmetric_unit_refinement/
```

or depending on your computer setup:

```
conda activate Assembline

cd IR_asymmetric_unit_refinement/
```

2. Run the refinement

```
# this will run 500 refinement modelling runs on a slurm cluster (for␣
↪options/parameters or local runs inspect the Assembline manual)
assembline.py --traj --models -o out --multi --start_idx 0 --njobs 500␣
↪config.json params.py
```

Note: There is already an output dir `IR_asymmetric_unit_refinement/out` so in case you want to run the modelling then rename the `out/` dir as it will be overwritten from the run above

3. Enter `out/` dir, generate output scoring lists and rebuild atomic structures of models

```
cd out

extract_scores.py #this should create a couple of files including all_
↪scores_sorted_uniq.csv

rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
↪directory IR_asymmetric_unit_refinement> config.json all_scores_sorted_
↪uniq.csv
```

(continues on next page)

(continued from previous page)

4. While in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is not provided therefore inspect an example file in `CR_Y_complex/out`. To generate it yourself please inspect Assembline analysis section.

5. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

CHAPTER

SEVENTEEN

NR Y-COMPLEX MODELLING

For modelling the NR Y-complex the calculation of fit libraries, global optimization and refinement of top "globally optimized" model with Assembline will be used. The dir `scnpc_tutorial/NR_Y_complex` includes source code files, input files and precalculated modelling results for the NR Y-complex:

```
- parameters file & configuration file for 'refinement' integrative modelling of wt NR Y-
↪complex
- sequence fasta file, NR_Y_complex_de_novo_model_PDBs, EM (input data)
- NR_Y_complex_final_refined_model.pdb, out (output modelling results)
- NR_Y_complex_de_novo_run (directory with global optimization data):

    - parameters file & configuration file for 'global optimization' integrative␣
↪modelling of wt NR Y-complex
    - sequence fasta file, input_PDB, EM (input data)
    - NR_Y_complex_de_novo_model.pdb,  out (output modelling results)

    - systematic_fits (directory):

        - parameters file for systematic fitting
        - em_maps, PDB (input data)
        - result_fits_chimera (output fitting results)
```

1. First activate your virtual environment and enter the `NR_Y_complex/NR_Y_complex_de_novo_run/systematic_fits` dir

   ```
   source activate Assembline

   cd NR_Y_complex/NR_Y_complex_de_novo_run/systematic_fits
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline

   cd NR_Y_complex/NR_Y_complex_de_novo_run/systematic_fits
   ```

2. Run the generation of fit libraries for NR Y-complex rigid bodies with Assembline (results calculated in dir `systematic_fits/result_fits_chimera/NR_merged_unerased_tail_relative_clean_v1.3.mrc/`)

   ```
   fit.py systematic_fitting_parameters.py
   ```

3. The fit libraries have been precalculated and analysed in dir `systematic_fits/result_fits_chimera/NR_merged_unerased_tail_relative_clean_v1.3.mrc/`. To analyse the fit results on your own run the following

```
# while in the systematic_fits/ dir
genpval.py result_fits_chimera
```

4. To generate the top five fits of each input rigid body (i.e. top five fits from each fit library) run the following

```
#upon successful run of fit.py and genpval
cd result_fits_chimera/search100000_metric_cam_rad_600_inside0.3_res_40

genPDBs_many.py -n5 top5 */*/solutions.csv

#repeat procedure for the other results in result_fits_chimera/ dir
```

5. After completing the calculation of fit libraries (or use the precalculated results) enter global optimization project dir (i.e. `NR_Y_complex_de_novo_run`) and run the global optimization

```
cd NR_Y_complex/NR_Y_complex_de_novo_run

# this will run 20000 global optimization modelling runs on a slurm cluster␣
↪(for options/parameters or local runs inspect the Assembline manual)
assembline.py --traj --models -o out --multi --start_idx 0 --njobs 20000␣
↪config.json params.py
```

Note: There is already an output dir `NR_Y_complex_de_novo_run/out` so in case you want to run the modelling then rename the `out/` dir as it will be overwritten from the run above

6. Enter `out/` dir, generate output scoring lists and rebuild atomic structures of models

```
cd out

extract_scores.py #this should create a couple of files including all_
↪scores_sorted_uniq.csv

rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
↪directory NR_Y_complex_de_novo_run> config.json all_scores_sorted_uniq.csv
```

7. While in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is not provided but only in the `CR_Y_complex/out`, therefore visit this dir to inspect it. To generate it yourself please inspect Assembline analysis section.

8. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
```

(continues on next page)

(continued from previous page)

```
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

Note: The four plots from the sampling exhaustiveness analysis are provided only in the `CR_Y_complex/out/analysis`. Therefore, visit this dir to inspect it and follow the commands above to run on your own.

9. In order to refine the best globally optimized model of NR Y-complex produced previsouly enter the main project dir (i.e. `scnpc_tutorial/NR_Y_complex/`) and run the refinement

```
# this will run 20000 refinement modelling runs on a slurm cluster (for␣
↪options/parameters or local runs inspect the Assembline manual)
assembline.py --traj --models -o out --multi --start_idx 0 --njobs 20000␣
↪config.json params.py
```

Note: There is already an output dir `NR_Y_complex/out` so in case you want to run the modelling then rename the `out/` dir as it will be overwritten from the run above. Also as you noticed the input PDBs used for refinement were stored in `NR_Y_complex_de_novo_model_PDBs` for convenience. If you want to run refinement on multiple models in parallel inspect the Assembline manual.

10. Enter `out/` dir, generate output scoring lists and rebuild atomic structures of models

```
cd out

extract_scores.py #this should create a couple of files including all_
↪scores_sorted_uniq.csv

rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
↪directory NR_Y_complex> config.json all_scores_sorted_uniq.csv
```

11. As after the `global optimization` run, while in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is not provided but only in the `CR_Y_complex/out`, therefore visit this dir to inspect it. To generate it yourself please inspect Assembline analysis section.

12. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

CHAPTER

EIGHTEEN

NUP116 K.O. SCNPC (AT 25C) MODELLING

For modelling the Nup116 k.o. ScNPC (at 25C) the local rigid body refinement method from Assembline will be used. The dir `scnpc_tutorial/Nup116delta25C` includes source code files, input files and precalculated modelling results for the Nup116 k.o. ScNPC model (25C):

```
- parameters file & configuration file for 'refinement' integrative modelling of␣
↪Nup116delta25C ScNPC
- sequence fasta file, input_PDB, EM (input data)
- ScNPC_Nup116delta25C_final_model.pdb, out (output modelling results)
```

1. First activate your virtual environment and enter the `Nup116delta25C` dir

   ```
   source activate Assembline

   cd Nup116delta25C/
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline

   cd Nup116delta25C/
   ```

2. Run the refinement

   ```
   # this will run 500 refinement modelling runs on a slurm cluster (for␣
   ↪options/parameters or local runs inspect the Assembline manual)
   assembline.py --traj --models -o out --multi --start_idx 0 --njobs 500␣
   ↪config.json params.py
   ```

   Note: There is already an output dir `Nup116delta25C/out` so in case you want to run the modelling then rename the `out/` dir as it will be overwritten from the run above

3. Enter `out/` dir, generate output scoring lists and rebuild atomic structures of models

   ```
   cd out

   extract_scores.py #this should create a couple of files including all_
   ↪scores_sorted_uniq.csv

   rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
   ↪directory Nup116delta25C> config.json all_scores_sorted_uniq.csv
   ```

4. While in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is not provided but only in the `CR_Y_complex/out`, therefore visit this dir to inspect it. To generate it yourself please inspect Assembline analysis section.

5. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

CHAPTER

NINETEEN

NUP116 K.O. SCNPC (AT 37C) MODELLING

For modelling the Nup116 k.o. ScNPC (at 37C) the local rigid body refinement method from Assembline will be used. The dir `scnpc_tutorial/Nup116delta37C` includes source code files, input files and precalculated modelling results for the Nup116 k.o. ScNPC model (37C):

```
- parameters files & configuration files for 'refinement' integrative modelling of␣
↪Nup116delta37C ScNPC
- sequence fasta file, input_PDB, EM (input data)
- ScNPC_Nup116delta37C_final_model.pdb, ScNPC_Nup116delta37C_IR_final_model.pdb, ScNPC_
↪Nup116delta37C_NR_final_model.pdb, out_IR, out_NR (output modeling results)
```

1. First activate your virtual environment and enter the `Nup116delta37C` dir

   ```
   source activate Assembline

   cd Nup116delta37C/
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline

   cd Nup116delta37C/
   ```

2. Run the refinement for NR Y-complex and half-assembly of IR unit (two separate refinement runs)

   ```
   # this will run 500 refinement modelling runs for NR Y-complex on a slurm␣
   ↪cluster (for options/parameters or local runs inspect the Assembline␣
   ↪manual)
   assembline.py --traj --models -o out --multi --start_idx 0 --njobs 500␣
   ↪config_NR.json params_NR.py

   # this will run 500 refinement modelling runs for IR unit subunits on a␣
   ↪slurm cluster (for options/parameters or local runs inspect the␣
   ↪Assembline manual)
   assembline.py --traj --models -o out --multi --start_idx 0 --njobs 500␣
   ↪config_IR.json params_IR.py
   ```

   Note: There are already output dirs `Nup116delta37C/out_NR` and `Nup116delta37C/out_IR` so in case you want to run the modelling then rename the `out_IR/` and/or `out_NR/` dir as it will be overwritten from the run above

3. Enter any of the `out/` dir (e.g. `out_NR/`), generate output scoring lists and rebuild atomic structures of models

```
#these steps can be followed for any of the out_NR and out_IR
cd out_NR

extract_scores.py #this should create a couple of files including all_
↪scores_sorted_uniq.csv

rebuild_atomic.py --top 10 --project_dir <full path to the original project␣
↪directory Nup116delta37C> config_NR.json all_scores_sorted_uniq.csv
```

4. While in the `out/` dir run the following command to prepare your output models for analysis with `imp-sampcon` tool from `IMP`

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Note: The density.txt is not provided but only in the `CR_Y_complex/out`, therefore visit this dir to inspect it. To generate it yourself please inspect Assembline analysis section.

---

5. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the sampling analysis:

```
cd analysis

imp_sampcon exhaust -n <prefix for output files> \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c <int for cores to process> \
-gp \
-g <float with clustering threshold step> \
```

Note: For further descriptions of settings for `imp_sampcon` please see Sampling exhaustiveness and precision with Assembline

---

Elongator complex modelling tutorial

Release 1.0

Vasileios Rantos, Kai Karius, Jan Kosinski

Jul 28, 2021

CONTENTS

1 About the Elongator complex **3**

2 Install Assembline **5**

3 Data preparation **7**

4 About fit libraries **9**

5 Set up **11**

6 Run **15**

7 Analyze fits **17**
7.1 Check if run correctly . . . 17

8 Setting up the JSON project file **19**
8.1 Create Xlink Analyzer project . . . 19
8.2 Add modeling information to the project file . . . 22

9 Setting up the parameter file **31**

10 Run **35**

11 Analyze **37**
11.1 Extract scores . . . 37
11.2 Create a CIF file of the top 10 models . . . 37
11.3 Quick checks . . . 37
11.4 Assess sampling exhaustiveness . . . 38
11.5 Conclusions . . . 40

12 Setting up the JSON project file **43**
12.1 Generating the file from scratch . . . 43

13 Setting up the parameter file **47**

14 Run **49**

15 Analyze **51**
15.1 Extract scores . . . 51
15.2 Create a CIF file of the top 10 models . . . 51
15.3 Quick convergence check: . . . 52
15.4 Assess sampling exhaustiveness . . . 52

15.5 Conclusions 54

16 TO DO LIST **57**

Assembline is a software for integrative structural modeling of macromolecular assemblies - an assembly line of macromolecular assemblies!

This tutorial demonstrates a basic usage of Assembline on an example of Elongator complex.

Use the **Next** button at the bottom right or menu on the left to navigate through the tutorial.

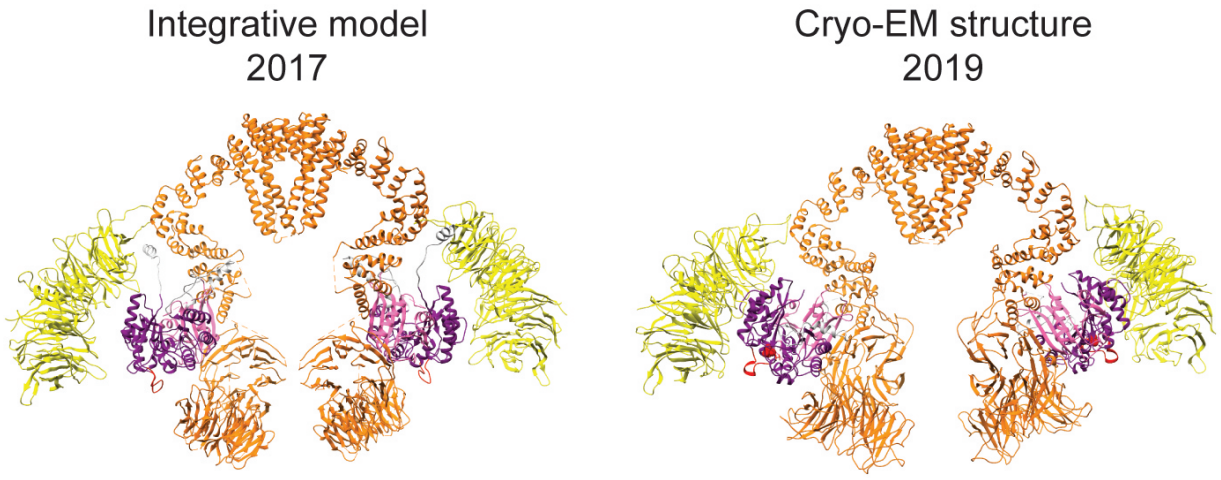
CHAPTER

ONE

ABOUT THE ELONGATOR COMPLEX

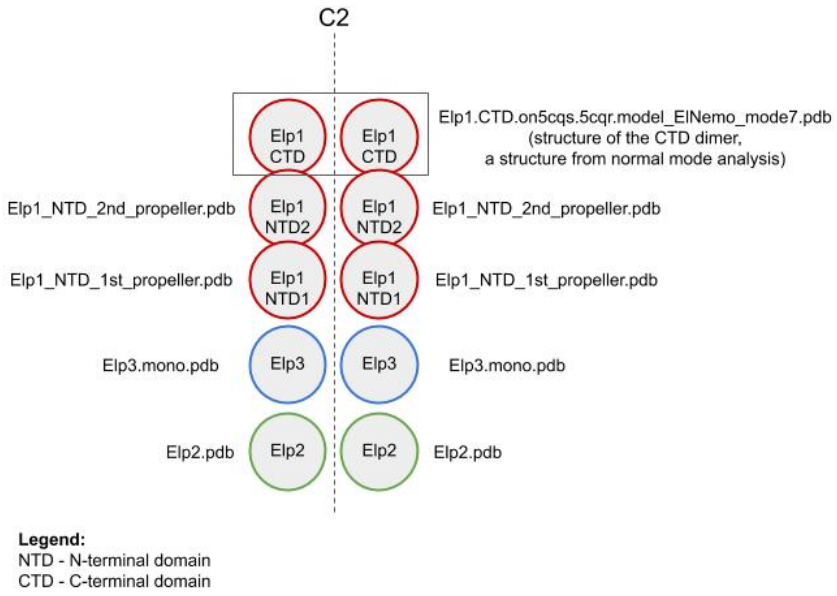
Elongator is a complex involved in tRNA modification. In yeast, it contains six subunits, two copies each.

We built and published an integrative model of yeast Elongator in 2017. The cryo-EM structure published in 2019 confirmed the model.



In this tutorial, we will model a subcomplex of Elongator composed of subunits named Elp1, Elp2, and Elp3, using the real data from the 2017 work.

The following PDB files are available for the subunits:



The following experimental data will be used as restraints:

- a negative stain EM map at 27 Å (EMD-4151):

  The map exhibits a C2 symmetry, thus the model will be built with C2 symmetry.

- crosslinks from a crosslinking mass spectrometry experiment (details here)

CHAPTER

TWO

INSTALL ASSEMBLINE

Install Assembline following instructions in the Manual.

CHAPTER

THREE

DATA PREPARATION

Download the data for this tutorial from here

Create a directory for your project and gather your data there. For this tutorial the data files are organized as follows:

```
Elongator/

    # Electron microscopy map
    EM_data/
        emd_4151_binned.mrc

    #input structures for modeling
    in_pdbs/
        Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb
        Elp1_NTD_1st_propeller.pdb
        Elp1_NTD_2nd_propeller.pdb
        Elp2.pdb
        Elp3.flex_helix.pdb
        Elp3.helix4.pdb
        Elp3.mono.pdb

    #FASTA-formatted file with sequences of all subunits
    elp_sequences.fasta

    #Crosslink files in xQuest or Xlink Analyzer format
    xlinks/
        DSS/
            inter_run3_190412.clean_strict.csv
            intra_run3_190412.clean_strict.csv
            loop_run3_190412.clean_strict.csv
            mono_run3_190412.clean_strict.csv
            sg1-inter.clean_strict.csv
            sg1-intra.clean_strict.csv
            sg1-loop.clean_strict.csv
            sg2-3-inter.clean_strict.csv
            sg2-3-intra.clean_strict.csv
            sg2-3-loop.clean_strict.csv
            sg2-3-mono.clean_strict.csv
        DSG/
            inter_dsg.clean_strict.csv
            inter_dsg_repeat.clean_strict.csv
            intra_dsg.clean_strict.csv
```

(continues on next page)

(continued from previous page)

```
            intra_dsg_repeat.clean_strict.csv
            loop_dsg.clean_strict.csv
            loop_dsg_repeat.clean_strict.csv
            mono_dsg.clean_strict.csv
            mono_dsg_repeat.clean_strict.csv
```

CHAPTER

FOUR

ABOUT FIT LIBRARIES

Fit libraries are lists of possible positions of your input structures in the EM map. The positions are used in the *global optimization* step to generate a large number of combinations of the fits through a Monte Carlo integrative modeling procedure. Each position has its associated score and p-value, which are used as EM restraints during the modeling.

Read more about fit libraries in the manual.

The fit libraries are typically generated by fitting every input structure into the EM map separately. For this tutorial, the fit libraries are already provided in the `fits/` directory.

Follow the next steps if you want to generate them yourself using Efitter package, which is a wrapper around UCSF Chimera FitMapTool .

To skip and jump to integrative modeling, move to *Setting up the JSON project file*.

CHAPTER

FIVE

SET UP

1. To run Efitter you need a parameter file in Python language format that specifies:
   - input structures to fit
   - EM map
   - fitting parameters
   - optionally, options for execution on a computer cluster
2. For this tutorial, the parameter file is already prepared in the `Elongator` folder:

   `efitter_params.py`:

   which contains:

```
from efitter import Map
from string import Template

method='chimera'
dry_run = False # Dry run would only print commands it is going to run
run_crashed_only = False # Only run the jobs that have not delivered output
master_outdir = 'fits' # relative path to the output, it will be created in␣
↪the running directory

MAPS = [
    Map('EM_data/emd_4151_binned.mrc', threshold=0.01, resolution=25),
]

models_dir = 'in_pdbs'

PDB_FILES = [
    'Elp1_NTD_1st_propeller.pdb',
    'Elp1_NTD_2nd_propeller.pdb',
    'Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb',
    'Elp2.pdb',
    'Elp3.mono.pdb',
]

CA_only = False # Calculate the fitting scores using Calpha atoms only?
backbone_only = False # Calculate the fitting scores using backbone atoms␣
↪only?
move_to_center = True # Move the PDB structure to the center of the map?
```

(continues on next page)

(continued from previous page)

```
# Each element of fitmap_args is a dictionary specifying parameters for a␣
↪run
# If multiple dictionaries are specified,
# the script will run a seperate run for each dictionary for each map-
↪structure combination.
# E.g. if two maps, three structures, and two paramaters dictionaries are␣
↪specified,
# the script will run 2 x 3 x 2 = 12 runs.
fitmap_args = [
    # Parameters for main runs (https://www.cgl.ucsf.edu/chimera/docs/
↪UsersGuide/midas/fitmap.html)
    {
        'template': Template("""
            map $map
            map_threshold $threshold
            fitmap_args resolution $resolution metric cam maxSteps 100␣
↪envelope true search 100000 placement sr clusterAngle 1 clusterShift 1.0␣
↪radius 200 inside .60
            saveFiles False
            """),
        'config_prefix': 'search100000_metric_cam_inside0.6' # some name␣
↪informative of the fitmap_args parameters
        }
]

# Adjust the parameters below to run on a cluster or standalone workstation

#For example, for a cluster with Slurm submission system, use:
cluster_submission_command = 'sbatch'
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --job-name=$job_name
#SBATCH --time=1-00:00:00
#SBATCH --error $pdb_outdir/log_err.txt
#SBATCH --output $pdb_outdir/log_out.txt
#SBATCH --mem=1000

$cmd
""")
# The run_script_templ is template for you cluster submission script (using␣
↪Python string.Template formatting)


#On a standalone computer
#1. Remove or comment out the cluster_submission_command and run_script_
↪templ
#2. Uncomment the following lines
# run_script_templ = Template("""#!/bin/bash
# #
# echo $job_name
# $cmd &>$pdb_outdir/log&
# """)
```

3. Based on the above parameter file, Efitter will run five fitting runs - one for each structure. You can run these as five independent jobs on a computer cluster or standalone workstation.

4. To run on a computer cluster, adjust the `cluster_submission_command` and `run_script_templ` to the queuing system on your cluster (the provided example would work only on a Slurm cluster). It is important that you include `$job_name`, `$pdb_outdir`, and `$cmd` in your script template.

5. To run on a standalone workstation, remove or comment out `cluster_submission_command` and replace the `run_script_templ` with

```
run_script_templ = Template("""#!/bin/bash
#
echo $job_name
$cmd &>$pdb_outdir/log&
""")
```

CHAPTER

SIX

RUN

1. Rename the *fits* directory provided in the tutorial to *fits_bk* (the fitting in the next step will write to *fits* directory)
2. Run the fitting

```
fit.py efitter_params.py
```

The fitting runs will execute in the background and take two to three hours.

The script will create the directory called `fits` with the following content:

```
fits/
    search100000_metric_cam_inside0.6/ #directory with output for the given␣
↪set of parameters
        emd_4151_binned.mrc/ #directory with all fits for this map for this␣
↪parameters

            #directories with names as the PDB files
            Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb/
            Elp1_NTD_1st_propeller.pdb/
            Elp1_NTD_2nd_propeller.pdb/
            Elp2.pdb/
            Elp3.mono.pdb/
            config.txt
            emd_4151_binned.mrc
```

Each of the `.pdb` directories should contain the following files:

```
emd_4151_binned.mrc #link to the map file
log_err.txt #error messages
log_out.txt #output messages
ori_pdb.pdb #link to the original PDB file
solutions.csv #file with transformation matrices defining the fits
```

Note: The fitting is complete when each of the `.pdb` directories contains `solutions.csv` file.

Inspect the `log_out.txt` files for status and `log_err.txt` for error messages.

3. Upon completion, calculate p-values:

```
genpval.py fits
```

This should create additional files in each `.pdb` directory:

```
Rplots.pdf
solutions_pvalues.csv
```

The `solutions_pvalues.csv` is crucial for the global optimization step.

CHAPTER

SEVEN

ANALYZE FITS

7.1 Check if run correctly

If everything run correctly, the fits directory should contain the following structure:

```
fits/
    search100000_metric_cam_inside0.6/
        emd_4151_binned.mrc/
            Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb/
                Rplots.pdf
                emd_4151_binned.mrc
                histogram.png
                log_err.txt
                log_out.txt
                ori_pdb.pdb
                run.sh
                solutions.csv
                solutions_pvalues.csv
            Elp1_NTD_1st_propeller.pdb/
                Rplots.pdf
                emd_4151_binned.mrc
                histogram.png
                log_err.txt
                log_out.txt
                ori_pdb.pdb
                run.sh
                solutions.csv
                solutions_pvalues.csv

            ... and so on
```

Warning: The **solutions_pvalues.csv** files contain the fit libraries and must be present for the next step to run.

CHAPTER

EIGHT

SETTING UP THE JSON PROJECT FILE

Note: You can find a ready-to-use JSON file named

```
elongator.json
```

in the tutorial directory and skip directly to the next step.

8.1 Create Xlink Analyzer project

Here is the instruction how to create the JSON file from scratch.

First, you need to create XlinkAnalyzer project file for your complex

Note: XlinkAnalyzer is used here as a graphical interface for input preparation in Assembline.

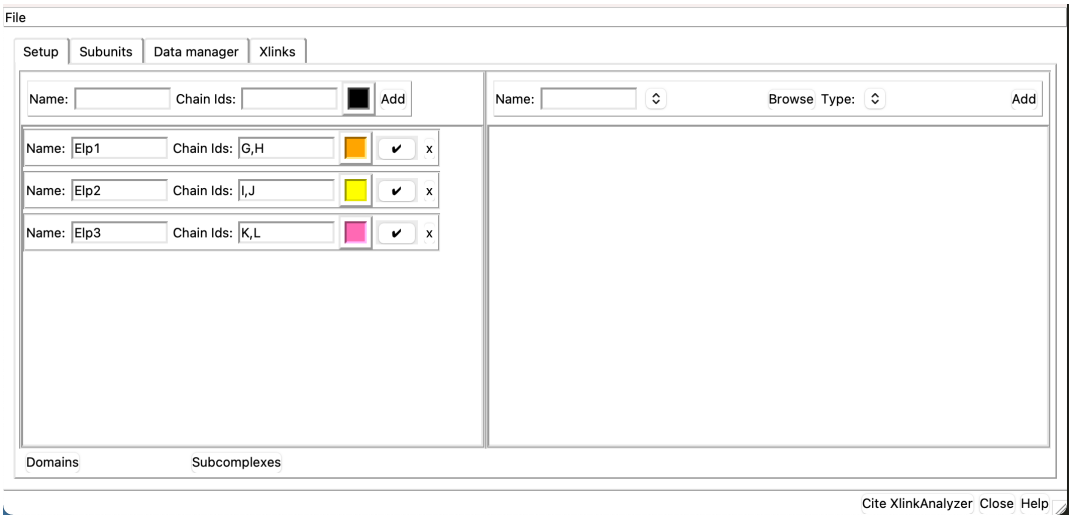
Does not matter if you do not have crosslinks - we use XlinkAnalyzer to prepare the input file for modeling.

1. Open Xlink Analyzer window:

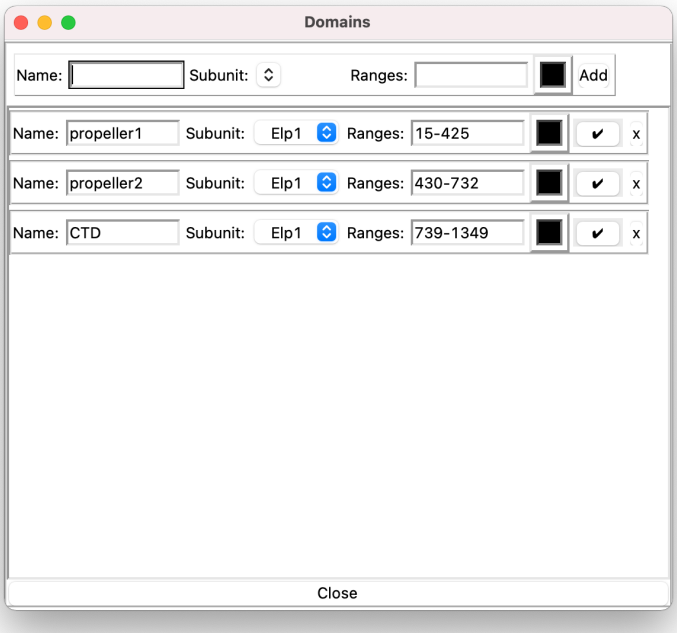
2. In the Xlink Analyzer project Setup tab, define subunits using the menu on the left. For each subunit, enter the name, the chain ID or comma-separated multiple IDs, define the color, and click Add button.

Set up the chain IDs as you want them in the final models, they do not have to correspond to chain IDs in your input PDB files.

The result should look like this:



3. Click on the Domains button and define domains of Elp1 in the window that opens - they will be used later for adding restraints:



Close the Domains window.

4. Load sequence data using the panel on the right in the Setup tab.

   For this, prepare a file with sequences of all proteins in a single file in FASTA format

   Here, use the `elp_sequences.fasta` file provided in the tutorial materials.

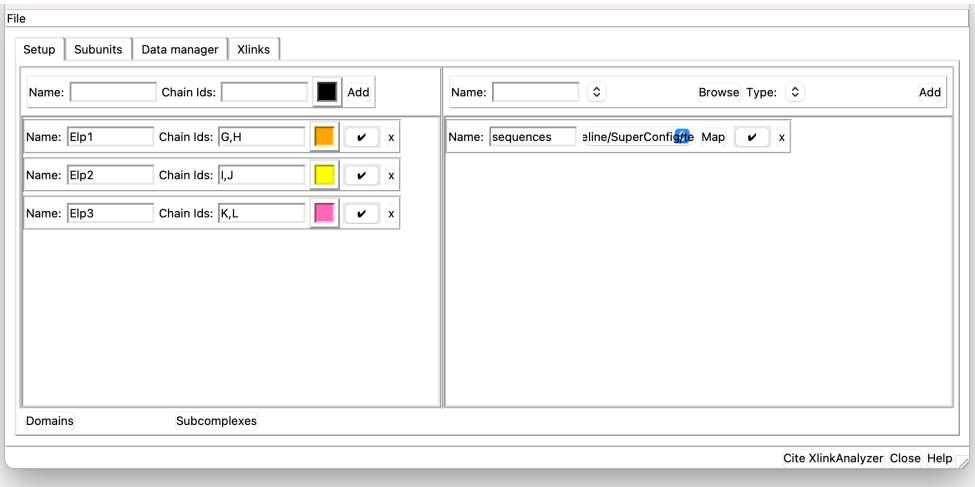
   Upload the file using the Browse button, enter a name (e.g. “sequences”) and select “sequence” type

in the drop down menu.

Click Add button.

Map the sequence names to names of subunits by clicking the Map button and selecting the subunits in a window that opens. After the mapping, click on the "check" button that turns red.

The result should look like this:



5. Load crosslink data using the same panel on the left in the Setup tab.

   The crosslink files need to be provided in Xlink Analyzer or xQuest format.

   Here, the files in xQuest format have been prepared in the tutorial materials:

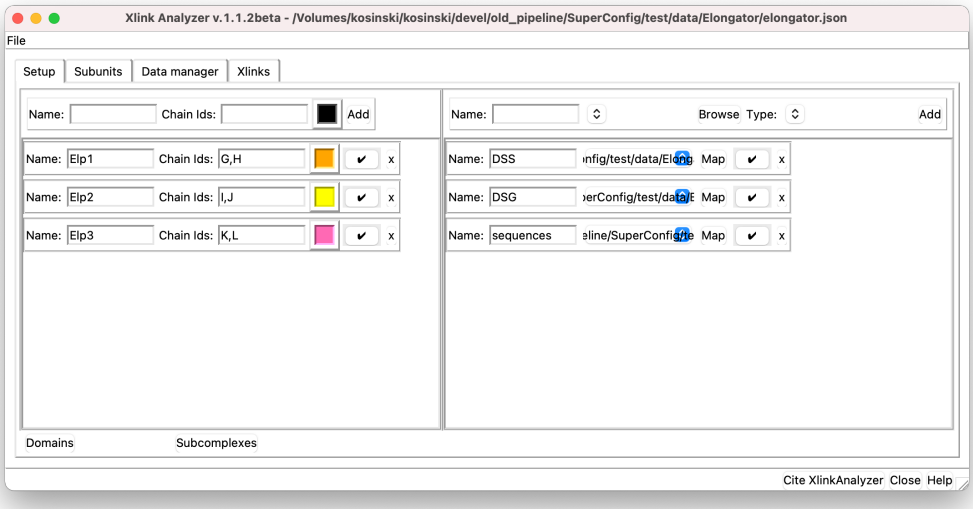
```
xlinks/
    DSS/
        inter_run3_190412.clean_strict.csv
        intra_run3_190412.clean_strict.csv
        loop_run3_190412.clean_strict.csv
        mono_run3_190412.clean_strict.csv
        sg1-inter.clean_strict.csv
        sg1-intra.clean_strict.csv
        sg1-loop.clean_strict.csv
        sg2-3-inter.clean_strict.csv
        sg2-3-intra.clean_strict.csv
        sg2-3-loop.clean_strict.csv
        sg2-3-mono.clean_strict.csv
    DSG/
        inter_dsg.clean_strict.csv
        inter_dsg_repeat.clean_strict.csv
        intra_dsg.clean_strict.csv
        intra_dsg_repeat.clean_strict.csv
        loop_dsg.clean_strict.csv
        loop_dsg_repeat.clean_strict.csv
        mono_dsg.clean_strict.csv
        mono_dsg_repeat.clean_strict.csv
```

   The files contain crosslinking results for two crosslinkers (DSS and DSG) in multiple files (multiple runs, inter/intra/loop/mono crosslinks in different files)

Name the first dataset "DSS" click Browse button and select all CSV files from the `DSS` directory. Set type to `xquest` or `XlinkAnalyzer`. Repeat for DSG.

Map crosslinked protein names to the subunit names using the Map button

After the operations above you should end up with sth like this:



6. Save the JSON file under a name like

```
xla_project.json
```

7. And make a copy that you will modify for modeling

```
cp xla_project.json elongator.json
```

8.2 Add modeling information to the project file

1. Open `elongator.json` in a text editor

Note: The project file is in so-called JSON format

While it may look difficult to edit at the first time, it is actually quite OK with a proper editor (and a bit of practice ;-)

We recommend to use a good editor such as:

- SublimeText
- Atom

At this point, the JSON has the following format:

```
{
    "data": [
        {
```

(continues on next page)

(continued from previous page)

```
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

2. Add symmetry

    1. First, specify the series of symmetry related molecules. Here, each of the three subunits is in two symmetrical copies, so we add series as below:

```
{
    "series": [
        {
            "name": "2fold",
            "subunit": "Elp1",
            "mode": "input",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        },
        {
            "name": "2fold",
            "subunit": "Elp2",
            "mode": "auto",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        },
        {
            "name": "2fold",
            "subunit": "Elp3",
            "mode": "auto",
            "cell_count": 2,
            "tr3d": "2fold",
            "inipos": "input"
        }
    ]
    "data": [
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
```

(continues on next page)

(continued from previous page)

```
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

2. Second, define the coordinates of the symmetry axis:

```
{
    "symmetry": {
        "sym_tr3ds": [

            {
                "name": "2fold",
                "axis": [0, 0, -1],
                "center": [246.39112398, 246.41114644, 248.600000],
                "type": "C2"

            }

        ]
    },
    "series": [
        "the series"
    ],
    "data": [
        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

3. Add specification of input PDB files

   The input structures for the tutorial are in the `in_pdbs/` directory:

```
Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb
Elp1_NTD_1st_propeller.pdb
```

(continues on next page)

(continued from previous page)

```
Elp1_NTD_2nd_propeller.pdb
Elp2.pdb
```

Add them to the JSON like this:

```
{
    "symmetry": {
        "symmetry axis definition"
    },
    "series": [
        "the series"
    ],
    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "domain": "propeller1",
                                "filename": "in_pdbs/Elp1_NTD_1st_propeller.
↪pdb"
                                }
                            ]
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "domain": "propeller2",
                                "filename": "in_pdbs/Elp1_NTD_2nd_propeller.
↪pdb"
                                }
                            ]
                        },
                        {
                            "components": [
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "serie": "2fold",
                                "copies": [0],
                                "chain_id": "G",
                                "domain": "CTD",
                                "filename": "in_pdbs/Elp1.CTD.on5cqs.5cqr.
↪model_ElNemo_mode7.pdb"},
```

(continues on next page)

(continued from previous page)

```
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "serie": "2fold",
                                "copies": [1],
                                "chain_id": "H",
                                "domain": "CTD",
                                "filename": "in_pdbs/Elp1.CTD.on5cqs.5cqr.
↪model_ElNemo_mode7.pdb"}
                            ]
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp2",
                                "subunit": "Elp2",
                                "filename": "in_pdbs/Elp2.pdb"}
                            ]
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp3",
                                "subunit": "Elp3",
                                "filename": "in_pdbs/Elp3.mono.pdb"}
                            ]
                        }

                ]

        },

        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

The `foreach_serie` and `foreach_copy` indicate the given PDB file specification will be applied to each serie with this subunit and for each copy within the series.

All PDB selections within the same `components` block will be grouped into a rigid body, unless a

separate `rigid_bodies` block is specified and `add_rbs_from_pdbs` is set to `False` in *Setting up the parameter file*

4. Add pointers to fit libraries

```
{
    "symmetry": {
        "symmetry axis definition"
    },
    "series": [
        "the series"
    ],
    "data": [
        {
            "type": "pdb_files",
            "name": "pdb_files",
            "data": [
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "domain": "propeller1",
                                "filename": "in_pdbs/Elp1_NTD_1st_propeller.
↪pdb"
                                }
                            ],
                            "positions": "fits/search100000_metric_cam_
↪inside0.6/emd_4151_binned.mrc/Elp1_NTD_1st_propeller.pdb/solutions_
↪pvalues.csv",
                            "positions_type": "chimera",
                            "max_positions": 10000
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "domain": "propeller2",
                                "filename": "in_pdbs/Elp1_NTD_2nd_propeller.
↪pdb"
                                }
                            ],
                            "positions": "fits/search100000_metric_cam_
↪inside0.6/emd_4151_binned.mrc/Elp1_NTD_2nd_propeller.pdb/solutions_
↪pvalues.csv",
                            "positions_type": "chimera",
                            "max_positions": 10000
                        },
                        {
                            "components": [
                                { "name": "Elp1",
```

(continues on next page)

(continued from previous page)

```
                                "subunit": "Elp1",
                                "serie": "2fold",
                                "copies": [0],
                                "chain_id": "G",
                                "domain": "CTD",
                                "filename": "in_pdbs/Elp1.CTD.on5cqs.5cqr.
↪model_ElNemo_mode7.pdb"},

                                { "name": "Elp1",
                                "subunit": "Elp1",
                                "serie": "2fold",
                                "copies": [1],
                                "chain_id": "H",
                                "domain": "CTD",
                                "filename": "in_pdbs/Elp1.CTD.on5cqs.5cqr.
↪model_ElNemo_mode7.pdb"}
                            ],
                            "positions": "fits/search100000_metric_cam_
↪inside0.6/emd_4151_binned.mrc/Elp1.CTD.on5cqs.5cqr.model_ElNemo_mode7.pdb/
↪solutions_pvalues.csv",
                            "positions_type": "chimera",
                            "max_positions": 1
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp2",
                                "subunit": "Elp2",
                                "filename": "in_pdbs/Elp2.pdb"}
                            ],
                            "positions": "fits/search100000_metric_cam_
↪inside0.6/emd_4151_binned.mrc/Elp2.pdb/solutions_pvalues.csv",
                            "positions_type": "chimera",
                            "max_positions": 10000
                        },
                        {
                            "foreach_serie": true,
                            "foreach_copy": true,
                            "components": [
                                { "name": "Elp3",
                                "subunit": "Elp3",
                                "filename": "in_pdbs/Elp3.mono.pdb"}
                            ],
                            "positions": "fits/search100000_metric_cam_
↪inside0.6/emd_4151_binned.mrc/Elp3.mono.pdb/solutions_pvalues.csv",
                            "positions_type": "chimera",
                            "max_positions": 10000
                        }

                ]
```

(continues on next page)

(continued from previous page)

```
        },

        {
            "some xlink definition 1"
        },
        {
            "some xlink definition 2"
        },
        {
            "sequence file definition"
        }
    ],
    "subunits": [
            "subunit definitions"
    ],
    "xlinkanalyzerVersion": "..."
}
```

And that's it!

CHAPTER

NINE

SETTING UP THE PARAMETER FILE

You can find a ready-to-use parameter file named

```
params.py
```

in the tutorial directory.

The file is in the Python format and contains the following content:

```
protocol = 'denovo_MC-SA'

SA_schedule = [
    (50000,   10000),
    (10000,   10000),
    (5000,    10000),
    (1000,    50000),
    (100,    50000)
]

do_ini_opt = True
ini_opt_SA_schedule = [
    (1000, 1000)
]

traj_frame_period = 10
print_frame_period = traj_frame_period

print_log_scores_to_files = True
print_log_scores_to_files_frame_period = 10

print_total_score_to_files = True
print_total_score_to_files_frame_period = 10


struct_resolutions = [0,10]
add_rbs_from_pdbs = True
connectivity_restraint_weight = 1.
conn_first_copy_only = True

add_symmetry_constraints = True
```

(continues on next page)

(continued from previous page)

```
add_xlink_restraints = True
x_xlink_score_type = 'LogHarmonic'
x_min_ld_score = 25
x_weight = 1000.0
x_xlink_distance = 15

discrete_restraints_weight=10000

ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    }
]

scoring_functions = {
    'score_func_ini_opt': {
        'restraints': [
            'discrete_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
    },
    'score_func_lowres': {
        'restraints': [
            'discrete_restraints',
            'xlink_restraints',
            'conn_restraints',
            'ev_restraints_lowres'
        ]
    }
}

score_func = 'score_func_lowres'
score_func_ini_opt = 'score_func_ini_opt'

ntasks = 1
cluster_submission_command = 'sbatch'
from string import Template
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=00:30:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
srun hostname
```

(continues on next page)

(continued from previous page)

```
$cmd

wait
""")
```

To generate this file yourself from scratch:

1. Download the template params_combinations.py file
2. Open the file in an editor to adjust the parameters as in the provided example.

   See parameters for explanations and description of available parameters.

CHAPTER

TEN

RUN

1. If you haven't yet, activate the environment before using the software by

```
source activate Assembline
```

   or depending on your computer setup:

```
conda activate Assembline
```

2. Enter the main project directory
3. Run a single run for testing

```
assembline.py  --traj --models --prefix 0000000 -o out elongator.json␣
↪params.py
```

   Output is saved to `out` directory specified by `-o` option.

4. Run 1000 runs to build 1000 models

   - Method 1: Submit all runs to the computer cluster or run on a workstation in chunks of N according to the number of processors:

```
assembline.py --traj --models -o out --multi --start_idx 0 --
↪njobs 1000 elongator.json params.py
```

     – on a cluster, this will submit 1000 modeling jobs in the queue, each job leading to one model (if ntasks in params.py is set to 1)
     – if ntasks params.py is N, it will run submit 1000/N cluster jobs, each running N modeling jobs
     – on a multicore computer, it will run ntasks at a time, and keep running until all 1000 jobs are done.

     Note: The number of processors or cluster submission commands and templates are specified in `params.py`

   - Method 2: Dynamically adjust the number of concurrent runs (e.g. to not to overload a cluster or annoy other users):

> **Warning:** The following works out of the box only on the EMBL cluster. To adjust to your cluster, modify the `assembline.py` for your cluster environment following the guidelines in the script.

```
assembline.py \
--traj \
--models \
--multi \
--daemon \
--min_concurrent_jobs 200 \
--max_concurrent_jobs 1000 \
-o out \
--start_idx 0 \
--njobs 1000 \
elongator.json params.py &>log&
```

- Method 3: If none of the above solutions work for you, you could probably submit multiple jobs manually using a mad shell loop e.g. on a computer cluster with the Slurm queuing system:

```
for i in $(seq -f "%07g" 0 999)
    do
        srun assembline.py  --traj --models --prefix $i -o out␣
↪elongator.json params.py &>log&
    done
```

  Just remember to make the `prefix` unique for every run.

Read more about how to run many runs on different platforms in the manual

CHAPTER

ELEVEN

ANALYZE

Enter the output directory.

```
cd out
```

11.1 Extract scores

```
extract_scores.py
```

This should create a couple of files including all_scores_sorted_uniq.csv

11.2 Create a CIF file of the top 10 models

```
rebuild_atomic.py --top 10 --project_dir ../ elongator.json all_scores_sorted_uniq.csv
```

Open the used EM map and models in UCSF Chimera and Xlink Analyzer. You would see that structures are fit to the map and crosslinks are satisfied only to some extent (are violated by only slightly). Spoiler: will be fixed in the refinement.

11.3 Quick checks

- Quick test of convergence

  Run quick test to assess convergence of the model score in randomly selected modelling trajectories

  ```
  plot_convergence.R total_score_logs.txt 20
  ```

  (change `20` to have less or more trajectories in the plots).

  Open the resulting `convergence.pdf` to visualize the convergence.

  It will plot score evolution for 20 runs. If the plots reach a plateau for all or most runs, the sampling converges.

- Visualize score distributions

  ```
  plot_scores.R all_scores.csv
  ```

Open the resulting `scores.pdf`

This can be used to:

- evaluate value ranges of restraints and use this information to re-scale the weights, for example, to bring all restraints to the same scale.
- select thresholds for analysis below and for selecting models for the refinement.

11.4 Assess sampling exhaustiveness

Run sampling performance analysis with imp-sampcon tool (described by Viswanath et al. 2017)

Warning: For the global optimization, the sampling exhaustiveness is not always applicable. For some cases, the optimization at this stage can actually work so well that it leads to all or most models being the same, resulting in very few clusters. In such cases, the sampling is exhaustive under the assumptions in the json but the estimation of sampling precision won't be possible. In such cases we recommend to intensively refine (e.g. with high initial temperatures in simulated annealing) the top (or all models) to create a diverse set of models for analysis.

1. Prepare the `density.txt` file

```
create_density_file.py --project_dir ../ elongator.json --by_rigid_body
```

2. Prepare the `symm_groups.txt` file storing information necessary to properly align homo-oligomeric structures

```
create_symm_groups_file.py --project_dir ../ elongator.json params.py
```

3. Run `setup_analysis.py` script to prepare input files for the sampling exhaustiveness analysis.

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt
```

Optionally, you can restrict the analysis to models scoring better than a given threshold:

```
setup_analysis.py -s all_scores.csv -o analysis -d density.txt --score_
↪thresh -16000
```

`--score_thresh` is to filtered out some rare very poorly scoring models (the threshold can be adjusted based on the `scores.pdf` generated above)

4. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the actual analysis:

```
cd analysis

imp_sampcon exhaust -n elongator \
--rmfA sample_A/sample_A_models.rmf3 \
--rmfB sample_B/sample_B_models.rmf3 \
--scoreA scoresA.txt --scoreB scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c 4 \
-gp \
-g 5.0 \
--ambiguity ../symm_groups.txt
```

5. In the output you will get, among other files:

   - `elongator.Sampling_Precision_Stats.txt`

     Estimation of the sampling precision. In this case it will be around 5-10 Angstrom

   - Clusters obtained after clustering at the above sampling precision in directories and files starting from `cluster` in their names, containing information about the models in the clusters and cluster localization densities
   - `elongator.Cluster_Precision.txt` listing the precision for each cluster, in this case between 7-20 Angstrom
   - PDF files with plots with the results of exhaustiveness tests

   See Viswanath et al. 2017 for detailed explanation of these concepts.

6. Optimize the plots

   The fonts and value ranges in X and Y axes in the default plots from `imp_sampcon exhaust` are frequently not optimal. For this you have to adjust them manually.

   1. Copy the original `gnuplot` scripts to the current `analysis` directory by executing:

      ```
      copy_sampcon_gnuplot_scripts.py
      ```

      This will copy for scripts to the current directory:

      - `Plot_Cluster_Population.plt` for the `elongator.Cluster_Population.pdf` plot
      - `Plot_Convergence_NM.plt` for the `elongator.ChiSquare.pdf` plot
      - `Plot_Convergence_SD.plt` for the `elongator.Score_Dist.pdf` plot
      - `Plot_Convergence_TS.plt` for the `elongator.Top_Score_Conv.pdf` plot

   2. Edit the scripts to adjust according to your liking
   3. Run the scripts again:

      ```
      gnuplot -e "sysname='elongator'" Plot_Cluster_Population.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_NM.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_SD.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_TS.plt
      ```

7. Extract cluster models for visualization

   ```
   extract_cluster_models.py \
       --project_dir <full path to the original project directory> \
       --outdir <cluster directory> \
       --ntop <number of top models to extract> \
       --scores <path to the score CSV file used as input for analysis> \
       Identities_A.txt Identities_B.txt <list of cluster models> <path to the␣
   ↪json>
   ```

   For example, to extract the 5 top scoring models from cluster 0:

   ```
   extract_cluster_models.py \
       --project_dir ../../ \
       --outdir cluster.0/ \
   ```

(continues on next page)

(continued from previous page)

```
    --ntop 5 \
    --scores ../all_scores.csv \
    Identities_A.txt Identities_B.txt cluster.0.all.txt ../elongator.json
```

The models are saved in the CIF format to `cluster.0` directory

8. If you want to re-cluster at a specific threshold (e.g. to get bigger clusters), you can do:

```
mkdir recluster
cd recluster/
cp ../Distances_Matrix.data.npy .
cp ../*ChiSquare_Grid_Stats.txt .
cp ../*Sampling_Precision_Stats.txt .
imp_sampcon exhaust -n elongator \
--rmfA ../sample_A/sample_A_models.rmf3 \
--rmfB ../sample_B/sample_B_models.rmf3 \
--scoreA ../scoresA.txt --scoreB ../scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c 4 \
-gp \
--ambiguity ../../symm_groups.txt \
--skip \
--cluster_threshold 40
```

And generate cluster models updating paths:

```
extract_cluster_models.py \
    --project_dir ../../../ \
    --outdir cluster.0/ \
    --ntop 1 \
    --scores ../../all_scores.csv \
    ../Identities_A.txt ../Identities_B.txt cluster.0.all.txt
```

11.5 Conclusions

The analysis shows that the sampling converged was exhaustive:

- the individual runs converge
- the exhaustiveness has been reached at 5-10 Angstrom according to the statistical test
- two random samples of models have similar score distributions and are similarly distributed over the clusters

Warning: The precision estimates should be taken with caution and do not represent "a resolution" because:

- the exhaustiveness is reached under the assumptions taken (weights, rigid body definitions etc.)
- the global optimization samples from a set of pre-calculated fits which constrain the search space, thus the precision is only defined within this space

Thus, the recombination step is not necessary.

Since, however, the crosslinks are still violated, we will still perform the refinement step.

CHAPTER

TWELVE

SETTING UP THE JSON PROJECT FILE

Enter the main directory.

Note: You can find a ready-to-use JSON template file named

```
elongator_refine_template.json
```

in the tutorial directory and immediately proceed to the next step of *Setting up the parameter file*.

12.1 Generating the file from scratch

1. Generate the template for the refinement.

```
gen_refinement_template.py --out_json elongator_refine_template.json --params␣
↪params.py elongator.json
```

2. In the following steps, modify the resulting `elongator_refine_template.json` template file by adding new restraints and inactivating or removing restraints not needed, adjusting weights.
3. Add the traditional cross-correlation-based EM fit restraint to the "data" block in the JSON file:

```
{
    "series": [

    ],
    "symmetry": {
        "sym_tr3ds": [

        ]
    },
    "data": [
        {
            "active": true,
            "type": "em_map",
            "name": "FitRestraint",
            "em_restraint_type": "FitRestraint",
            "filename": "EM_data/emd_4151_binned.mrc",
            "threshold": 0.01,
            "voxel_size": 6.6,
```

(continues on next page)

(continued from previous page)

```
            "resolution": 25,
            "weight": 1000,
            "first_copy_only": true,
            "repr_resolution": 10,
            "optimized_components": [
                {"name": "Elp1", "subunit": "Elp1"},
                {"name": "Elp2", "subunit": "Elp2"},
                {"name": "Elp3", "subunit": "Elp3"}

            ]
        }
    ],
    "subunits": [

    ],
    "xlinkanalyzerVersion": "0.1"
}
```

4. Redefine rigid bodies. In this case, we noticed that some crosslinks are in long loops and tails. We will then let them move flexibly in the refinement.

   To to do this, we add `rigid_bodies` block to the JSON file, in which we define rigid bodies with `resi_ranges`, excluding the selected loops:

```
{
    "series": [

    ],
    "rigid_bodies": [
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp1_1", "subunit": "Elp1", "domain": "propeller1
↪"}
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp1_2", "subunit": "Elp1", "domain": "propeller2
↪"}
            ]
        },
        {
            "foreach_serie": true,
            "components": [
                { "name": "Elp1_3", "subunit": "Elp1", "domain": "CTD",
↪"copies": [0, 1]}
            ],
            "freeze": true
        },
```

(continues on next page)

(continued from previous page)

```
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp2", "subunit": "Elp2",
                    "resi_ranges": [
                        [1, 239],
                        [257, 788]
                    ]
                }
            ]
        },
        {
            "foreach_serie": true,
            "foreach_copy": true,
            "components": [
                { "name": "Elp3", "subunit": "Elp3",
                    "resi_ranges": [
                        [95, 468],
                        [477, 557]
                    ]}
            ]
        }
    ],
    "symmetry": {
        "sym_tr3ds": [

        ]
    },
    "data": [

    ],
    "subunits": [

    ],
    "xlinkanalyzerVersion": "0.1"
}
```

Note: Whenever you define custom rigid bodies here, set `add_rbs_from_pdbs = False` in the parameter file.

5. Add symmetry restraint for the C-terminal domain of Elp1 (CTD).

   Why? The input structure for the CTD domain is already a symmetrical dimer and the entire dimer is defined as a rigid body. This domain cannot be constrained by the symmetry "constraint" because the constraints do not work "within rigid bodies".

   To define the symmetry restraint, add the following block:

```
{
    "series": [
```

(continues on next page)

(continued from previous page)

```
    ],
    "rigid_bodies": [

    ],
    "symmetry": {
        "sym_tr3ds": [

        ]
        "apply_symmetry": [
            {
                "sym": "2fold",
                "restraint_type": "symmetry_restraint",
                "selectors": [
                    {"subunit": "Elp1", "serie": "2fold", "copies": [0],
↪"domain": "CTD"},
                    {"subunit": "Elp1", "serie": "2fold", "copies": [1],
↪"domain": "CTD"}
                ]
            }
        ]
    },
    "data": [

    ],
    "subunits": [

    ],
    "xlinkanalyzerVersion": "0.1"
}
```

Note: Whenever you define symmetry restraints here, in the parameter file set `add_symmetry_restraints = True` and add `'sym_restraints'` to the scoring function.

CHAPTER

THIRTEEN

SETTING UP THE PARAMETER FILE

Note: You can find a ready-to-use parameter file named

```
params_refine.py
```

in the tutorial directory.

The file is in the Python format and contains the following content:

```
protocol = 'refine'

SA_schedule = [
    (100000,   10000),
    (30000,   10000),
    (20000,   10000),
    (10000,   10000)
]

traj_frame_period = 1
print_frame_period = traj_frame_period

print_log_scores_to_files = True
print_log_scores_to_files_frame_period = 10

print_total_score_to_files = True
print_total_score_to_files_frame_period = 10

struct_resolutions = [0,10]
missing_resolution = 0
add_rbs_from_pdbs = False
connectivity_restraint_weight = 1.
conn_first_copy_only = False

rb_max_rot = 0.0174533
rb_max_trans = 1

add_symmetry_constraints = True
add_symmetry_restraints = True

add_xlink_restraints = True
```

(continues on next page)

(continued from previous page)

```
x_xlink_score_type = 'LogHarmonic'
x_min_ld_score = 25
x_weight = 5000.0
x_xlink_distance = 15

ev_restraints = [
    {
        'name': 'ev_restraints_lowres',
        'weight': 10,
        'repr_resolution': 10
    }
]

scoring_functions = {
    'score_func_lowres': {
        'restraints': [
            'xlink_restraints',
            'conn_restraints',
            'ev_restraints_lowres',
            'FitRestraint',
            'sym_restraints'
        ]
    }
}

score_func = 'score_func_lowres'

ntasks = 1
cluster_submission_command = 'sbatch'
from string import Template
run_script_templ = Template("""#!/bin/bash
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=$ntasks
#SBATCH --mem-per-cpu=2000
#SBATCH --job-name=${prefix}fit
#SBATCH --time=00:30:00
#SBATCH -e $outdir/logs/${prefix}_err.txt
#SBATCH -o $outdir/logs/${prefix}_out.txt

echo "Running on:"
srun hostname

$cmd

wait
""")
```

To generate this file yourself from scratch:

1. Download the template parameter file (`params_refine.py`) from: https://git.embl.de/kosinski/Assembline/-/tree/master/doc/templates
2. Open the file in an editor to adjust the parameters

CHAPTER

FOURTEEN

RUN

This shows how to refine top all models, running three refinement runs for each model.

1. I you haven't yet, activate the environment before using the software by

   ```
   source activate Assembline
   ```

   or depending on your computer setup:

   ```
   conda activate Assembline
   ```

2. Enter the main project directory

3. Setup a refinement directory to refine all 1000 models from the global optimization

   ```
   setup_refine.py \
       --scores out/all_scores_uniq.csv \
       --previous_json elongator.json \
       --refine_json_template elongator_refine_template.json \
       --refine_json_outname elongator_refine.json \
       --previous_outdir out/\
       --refine_outdir out/refinement
   ```

   This will create a directory `out/refinement`, which will contain 1000 sub-directories. Each of the subdirectories contains input files for refining each of the 1000 scoring models. The input files include PDB files oriented as in the models and JSON files for referring to those PDB files.

4. Run a test

   ```
   model_id=`ls --color=never out/refinement | head -n 1`
   assembline.py --traj --models --prefix 0000000 -o out/refinement/"$model_id"/out␣
   ↪out/refinement/"$model_id"/elongator_refine.json params_refine.py
   rm -r out/refinement/"$model_id"/out
   ```

5. Refine the 1000 models, running 3 refinement runs for each model (3000 runs in total).

   For example like this:

   ```
   for model_id in `ls --color=never out/refinement`;
       do
           echo $model_id
           assembline.py --models -o out/refinement/"$model_id"/out --multi --
   ↪start_idx 0 --njobs 3 --prefix refine_"$model_id" out/refinement/"$model_
   ↪id"/elongator_refine.json params_refine.py
       done
   ```

This will navigate to all subdirectories of the `out/refinement` and run the refinements there. Note no `--traj` option to save disk space.

CHAPTER

FIFTEEN

ANALYZE

Enter the output directory.

```
cd out/refinement
```

15.1 Extract scores

```
extract_scores.py --multi
```

Visualize score distributions (will be needed later to infer score thresholds)

```
plot_scores.R all_scores.csv
```

15.2 Create a CIF file of the top 10 models

```
rebuild_atomic.py --project_dir <full path to the original project directory> --top 10␣
↪all_scores_sorted_uniq.csv  --rmf_auto
```

Yes - no JSON project file in argument - here each refinement run had its own project file (pointing to a different input structure), and `rebuild_atomic.py` will locate the JSON project files automatically based on the model IDs in the all_scores_sorted_uniq.csv file.

`--project_dir` is necessary if you use relative paths in the JSON project file. `` –rmf_auto`` will read the beads from RMF file and use Modeller to re-build full atomic loops!

For example here:

```
rebuild_atomic.py --project_dir ../../ --top 10 all_scores_sorted_uniq.csv  --rmf_auto
```

Open the used EM map and models in UCSF Chimera and Xlink Analyzer. You would see that structures are fit to the map still well but crosslinks are now satisfied!

15.3 Quick convergence check:

```
plot_convergence.R total_score_logs.txt 20
```

Open the resulting `scores.pdf`

15.4 Assess sampling exhaustiveness

Run sampling performance analysis with imp-sampcon tool (described by Viswanath et al. 2017)

1. Prepare the `density.txt` file

   ```
   create_density_file.py --project_dir <full path to the original project␣
   ↪directory> <path_to_one_of_the_refinement_folders>/out/elongator_refine.
   ↪json --by_rigid_body
   ```

   e.g.

   ```
   create_density_file.py --project_dir ../../ 0000171/out/elongator_refine.
   ↪json --by_rigid_body
   ```

   replacing `0000171` with the name of any directory in your refinement directory.

2. Prepare the `symm_groups.txt` file storing information necessary to properly align homo-oligomeric structures

   ```
   create_symm_groups_file.py --project_dir <full path to the original project␣
   ↪directory> <path_to_one_of_the_refinement_folders>/elongator.json
   ```

   e.g.

   ```
   create_symm_groups_file.py --project_dir ../../ 0000171/out/elongator_
   ↪refine.json 0000171/out/params_refine.py
   ```

3. Run `setup_analysis.py` script to prepare input files for the sampling exhaustiveness analysis.

   ```
   setup_analysis.py -s all_scores.csv -o analysis -d density.txt --score_
   ↪thresh 70000
   ```

   Here we use a score threshold derived from the corresponding score distribution in `scores.pdf`, to filter out poorly fitting models.

4. Run `imp-sampcon exhaust` tool (command-line tool provided with IMP) to perform the actual analysis:

   ```
   cd analysis

   imp_sampcon exhaust -n elongator \
   --rmfA sample_A/sample_A_models.rmf3 \
   --rmfB sample_B/sample_B_models.rmf3 \
   --scoreA scoresA.txt --scoreB scoresB.txt \
   -d ../density.txt \
   -m cpu_omp \
   -c 4 \
   -gp \
   ```

(continues on next page)

(continued from previous page)

```
-g 5.0 \
--ambiguity ../symm_groups.txt
```

5. In the output you will get, among other files:

   - `elongator.Sampling_Precision_Stats.txt`

     Estimation of the sampling precision. In this case it will be around 20 Angstrom

   - Clusters obtained after clustering at the above sampling precision in directories and files starting from `cluster` in their names, containing information about the models in the clusters and cluster localization densities
   - `elongator.Cluster_Precision.txt` listing the precision for each cluster, in this case around 10-20 Angstrom
   - PDF files with plots with the results of exhaustiveness tests

   See Viswanath et al. 2017 for detailed explanation of these concepts.

6. Optimize the plots

   The fonts and value ranges in X and Y axes in the default plots from `imp_sampcon exhaust` are frequently not optimal. For this you have to adjust them manually.

   1. Copy the original `gnuplot` scripts to the current `analysis` directory by executing:

      ```
      copy_sampcon_gnuplot_scripts.py
      ```

      This will copy for scripts to the current directory:

      - `Plot_Cluster_Population.plt` for the `elongator.Cluster_Population.pdf` plot
      - `Plot_Convergence_NM.plt` for the `elongator.ChiSquare.pdf` plot
      - `Plot_Convergence_SD.plt` for the `elongator.Score_Dist.pdf` plot
      - `Plot_Convergence_TS.plt` for the `elongator.Top_Score_Conv.pdf` plot

   2. Edit the scripts to adjust according to your liking
   3. Run the scripts again:

      ```
      gnuplot -e "sysname='elongator'" Plot_Cluster_Population.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_NM.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_SD.plt
      gnuplot -e "sysname='elongator'" Plot_Convergence_TS.plt
      ```

7. Extract cluster models:

   For example, for the top cluster:

   ```
   extract_cluster_models.py \
   --project_dir ../../../ \
   --outdir cluster.0/ \
   --ntop 5 \
   --scores ../all_scores.csv \
   --rebuild_loops \
   Identities_A.txt Identities_B.txt cluster.0.all.txt
   ```

Yes, no json file as the last argument (contrary to *Analyze*), for refinement of multiple models the program will find JSON files itself (as they are different for each model).

8. If you want to re-cluster at a specific threshold (e.g. to get bigger clusters), you can do:

```
mkdir recluster
cd recluster/
cp ../Distances_Matrix.data.npy .
cp ../*ChiSquare_Grid_Stats.txt .
cp ../*Sampling_Precision_Stats.txt .
imp_sampcon exhaust -n elongator \
--rmfA ../sample_A/sample_A_models.rmf3 \
--rmfB ../sample_B/sample_B_models.rmf3 \
--scoreA ../scoresA.txt --scoreB ../scoresB.txt \
-d ../density.txt \
-m cpu_omp \
-c 4 \
-gp \
--ambiguity ../../symm_groups.txt \
--skip \
--cluster_threshold 25 \
--voxel 2
```

And generate cluster models updating paths:

```
extract_cluster_models.py \
    --project_dir ../../../../ \
    --outdir cluster.0/ \
    --ntop 1 \
    --scores ../../all_scores.csv \
    --rebuild_loops \
    ../Identities_A.txt ../Identities_B.txt cluster.0.all.txt
```

15.5 Conclusions

The analysis shows that the sampling converged was exhaustive:

- the individual runs converge
- the exhaustiveness has been reached at 10-20 Angstrom according to the statistical test
- two random samples of models have similar score distributions and are similarly distributed over the clusters

Warning: The precision estimates still should be taken with caution, as for the previous stage, but here the estimates are a bit more realistic because the original models were allowed to explore larger conformational space.

The crosslinks are now satisfied in the top scoring model.

For further biological hypothesis generation based on the model, we would use the top scoring models from each cluster and from all runs.

We might use this top scoring model as a representative model for figures.

As the “final output”, however, we would use the ensemble of models (e.g. from the clusters or top 10 models from all runs) for depicting the uncertainty.