ScNPC modelling tutorial

Release 1.0

Vasileios Rantos, Kai Karius, Jan Kosinski

Aug 24, 2021

CONTENTS

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.

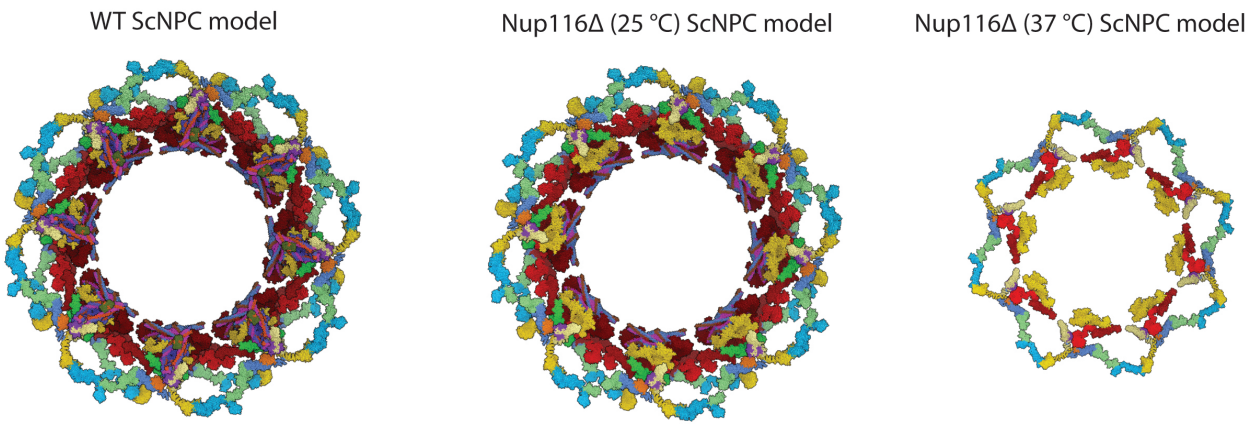
CHAPTER

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)

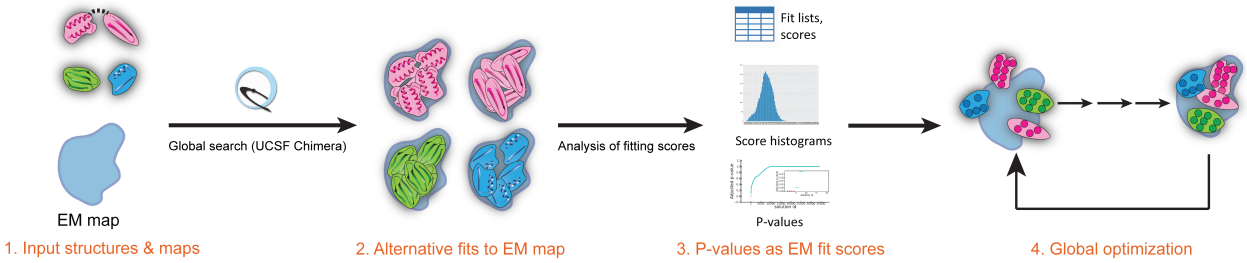
```
        #FASTA-formatted file with sequences of all subunits
        ScNPC_sequences.fasta
```

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 ScNPC_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