Elongator complex modelling tutorial

Release 1.0

Vasileios Rantos, Kai Karius, Jan Kosinski

Feb 20, 2022

CONTENTS

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.

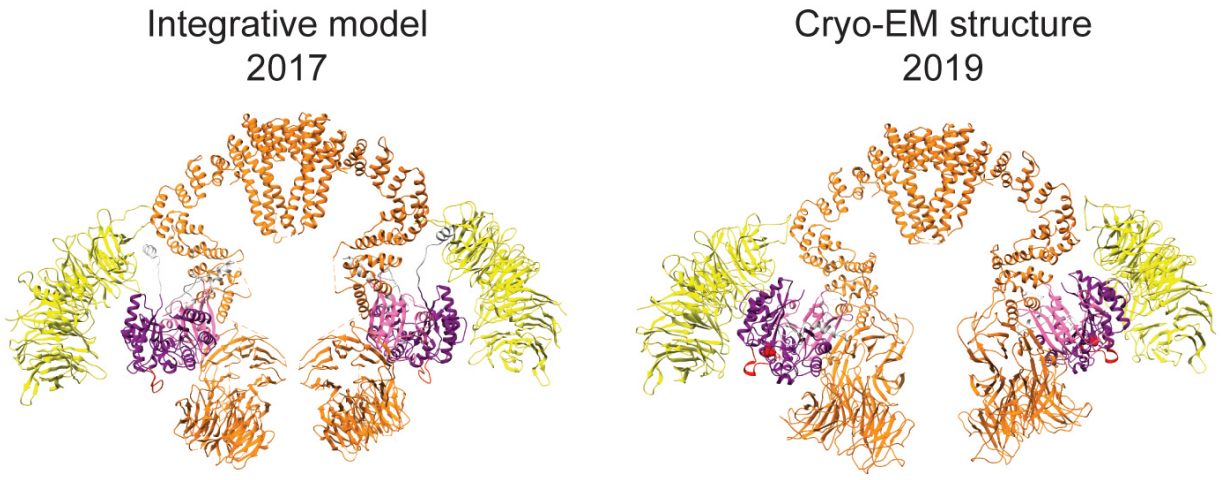
CHAPTER

ONE

ABOUT THE ELONGATOR COMPLEX

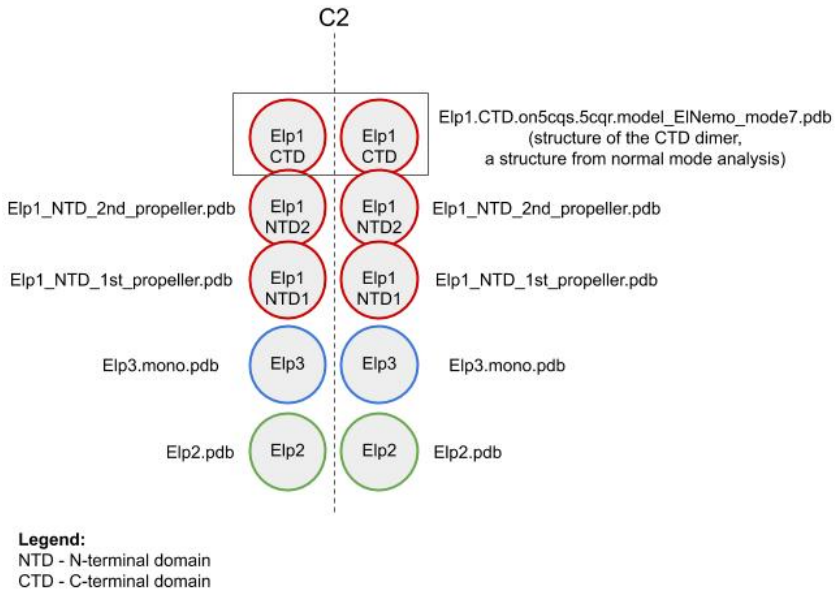
Elongator is a complex involved in tRNA modification. In yeast, it contains six subunits, two copies each.

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.

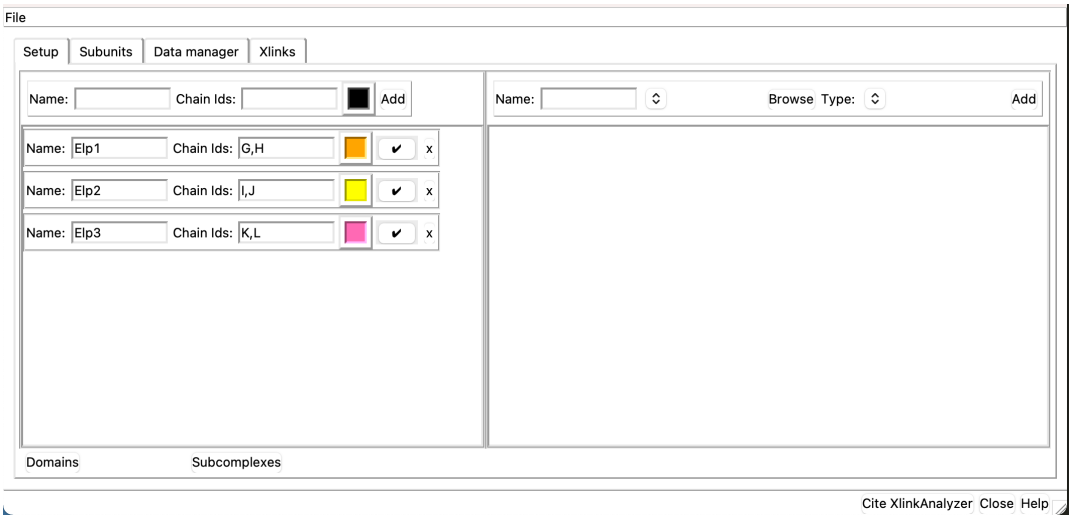
Does not matter if you do not have crosslinks - we use XlinkAnalyzer to prepare the input file for modeling.

1. Open Xlink Analyzer window:

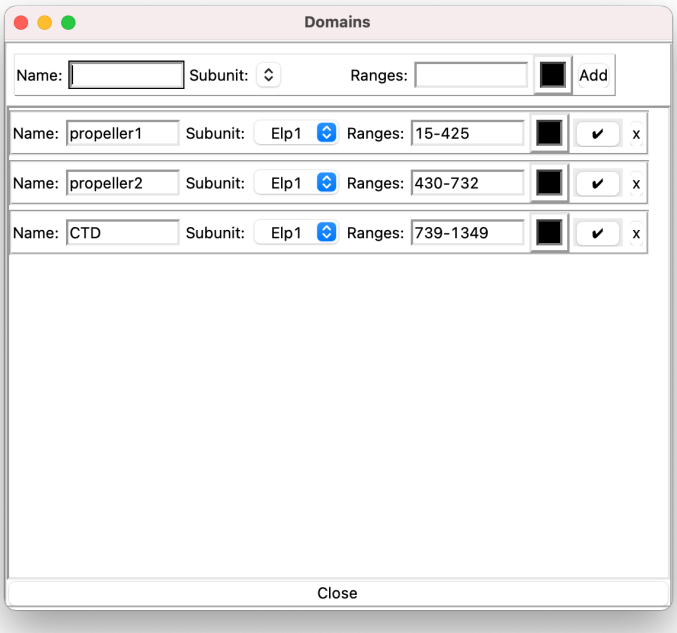
2. In the Xlink Analyzer project Setup tab, define subunits using the menu on the left. For each subunit, enter the name, the chain ID or comma-separated multiple IDs, define the color, and click Add button.

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.

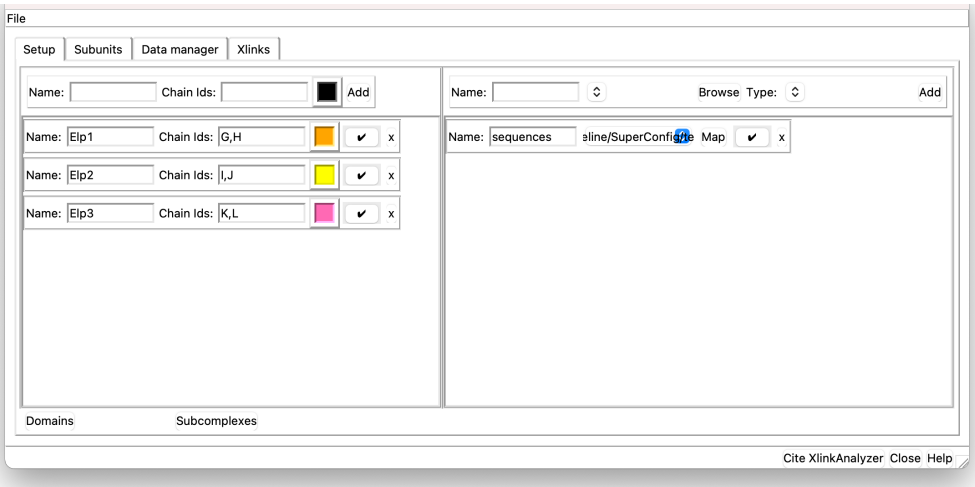
   Upload the file using the Browse button, enter a name (e.g. "sequences") and select "sequence" type

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:

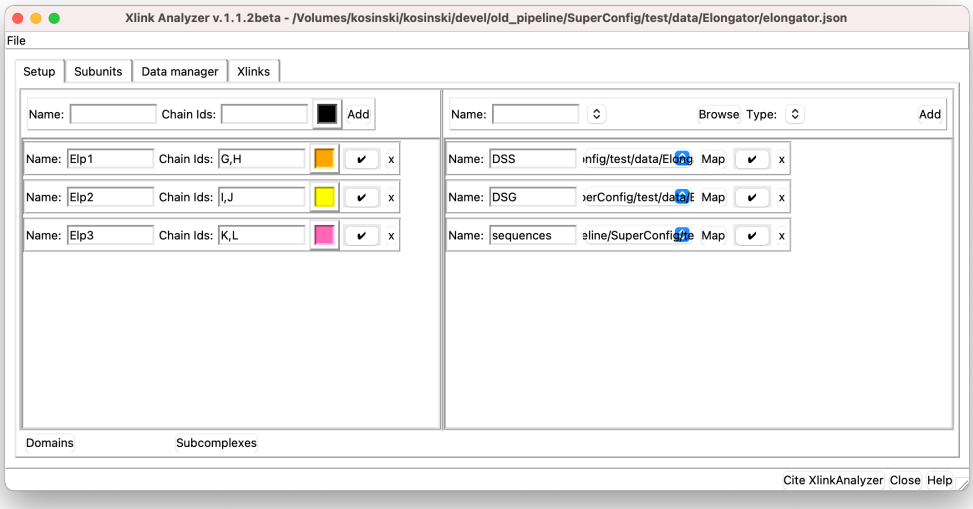
```
xlinks/
    DSS/
        inter_run3_190412.clean_strict.csv
        intra_run3_190412.clean_strict.csv
        loop_run3_190412.clean_strict.csv
        mono_run3_190412.clean_strict.csv
        sg1-inter.clean_strict.csv
        sg1-intra.clean_strict.csv
        sg1-loop.clean_strict.csv
        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 ../elongator.
↪json
```

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.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.

CHAPTER

SIXTEEN

TO DO LIST