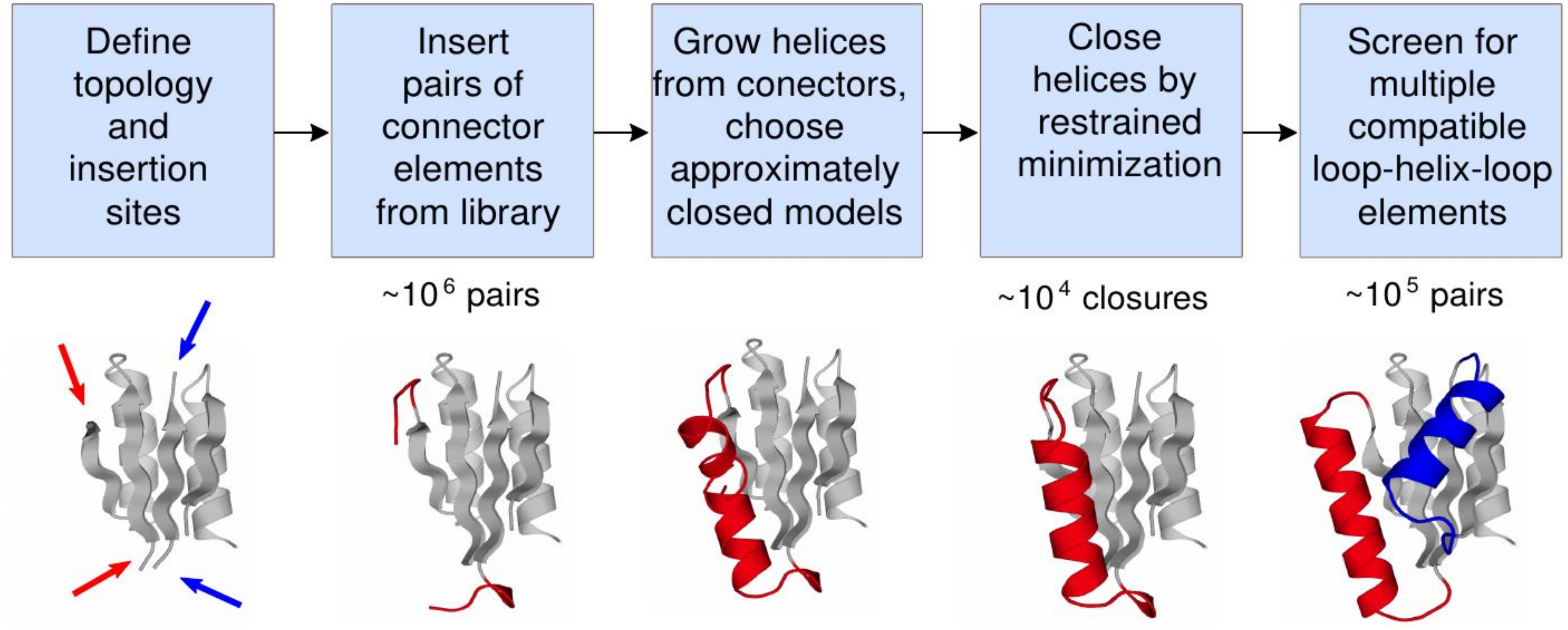
LUCS: Method, Code and Usage

Xingjie Pan
2020-07-21

Loop-helix-loop unit combinatorial sampling (LUCS)



Define topology and insertion sites → Insert pairs of connector elements from library → Grow helices from conectors, choose approximately closed models → Close helices by restrained minimization → Screen for multiple compatible loop-helix-loop elements

For detailed explanation of the algorithm, refer to:

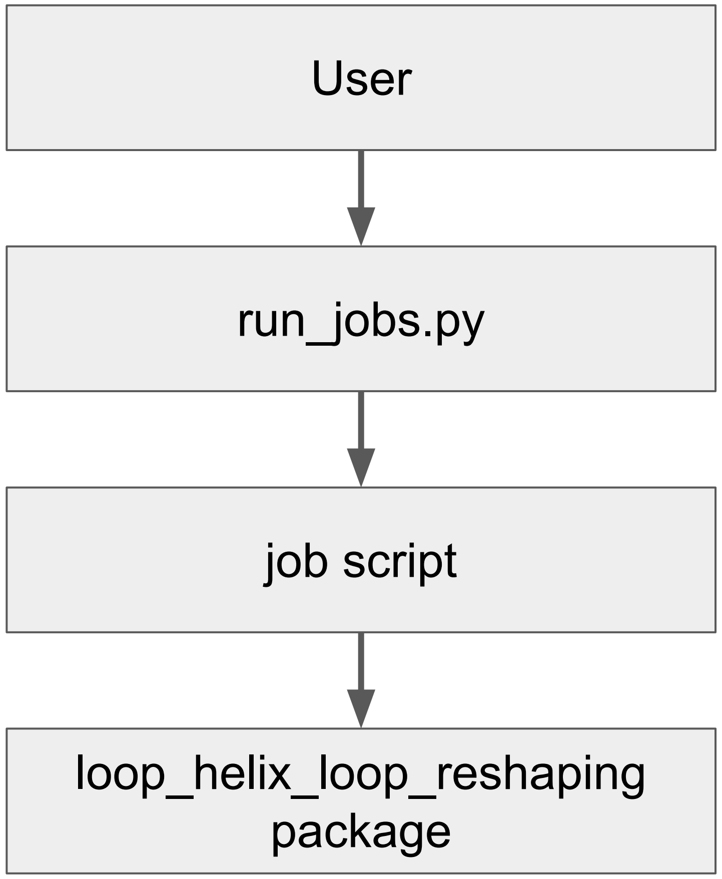
Pan, X., Thompson, M., Zhang, Y., Lin, L., Fraser, J.S., Kelly, M.J. and Kortemme, T., 2020. Expanding the space of protein geometries by computational design of de novo fold families.

The loop_helix_loop_reshaping repository

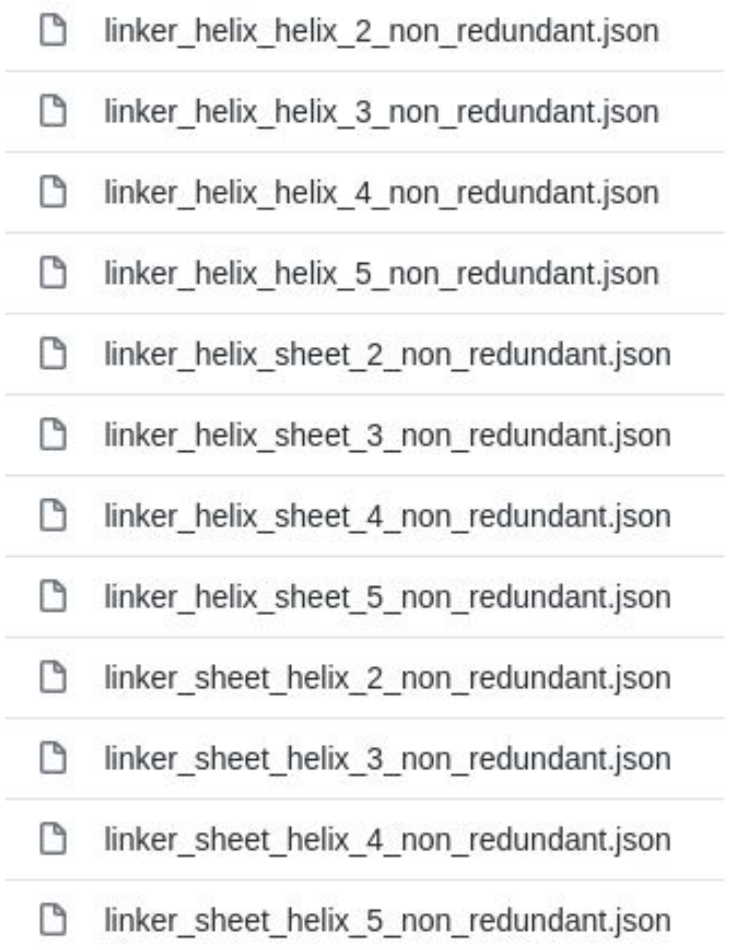
https://github.com/Kortemme-Lab/loop_helix_loop_reshaping

Main components:

- database
- loop_helix_loop_reshaping package
- job_scripts
- run_jobs.py



database/linker_database



linker_helix_helix_2_non_redundant.json
linker_helix_helix_3_non_redundant.json
linker_helix_helix_4_non_redundant.json
linker_helix_helix_5_non_redundant.json
linker_helix_sheet_2_non_redundant.json
linker_helix_sheet_3_non_redundant.json
linker_helix_sheet_4_non_redundant.json
linker_helix_sheet_5_non_redundant.json
linker_sheet_helix_2_non_redundant.json
linker_sheet_helix_3_non_redundant.json
linker_sheet_helix_4_non_redundant.json
linker_sheet_helix_5_non_redundant.json

linker_helix_helix_2_non_redundant.json

[{"sequence": ["K", "K", "H", "L"], "pdb_id": "7odc", "phis": [-64.512, -64.796, -67.053, -61.949], "psis": [-38.563, -38.021, -43.075, -41.748], "start_position": 15, "omegas": [177.556, 179.386, 175.068, 176.707]}, ...]

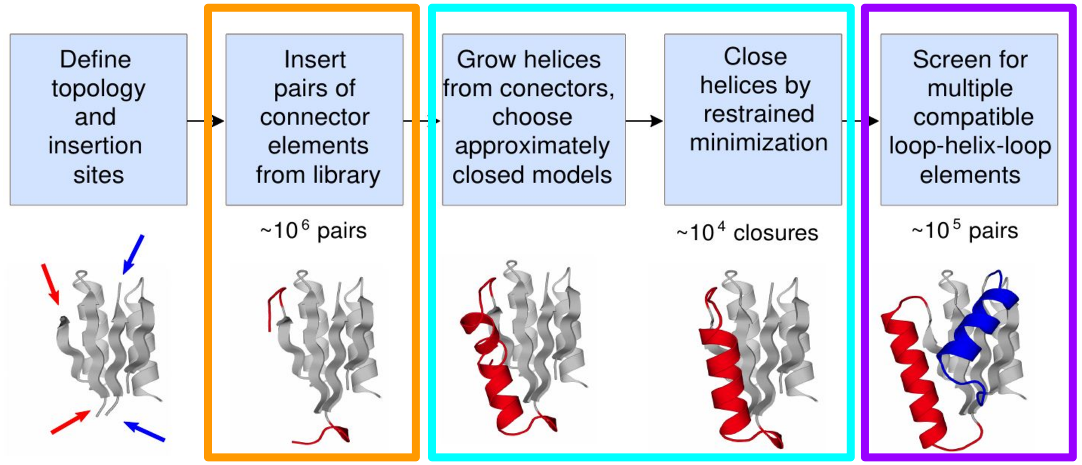
Torsions for the two anchoring points are also included!

loop_helix_loop_reshaping package

Designed as a python package for the ease to be incorporated into other codes.

Core functions are:

- select_linkers.select_non_clashing_linkers()
- build_loop_helix_loop_unit.screen_all_loop_helix_loop_units()
- screen_compatible_loop_helix_loop_units.screen_compatible_loop_helix_loop_units()


Define topology and insertion sites
Insert pairs of connector elements from library
~10^6 pairs
Grow helices from conectors, choose approximately closed models
Close helices by restrained minimization
~10^4 closures
Screen for multiple compatible loop-helix-loop elements
~10^5 pairs

job_scripts

A job script calls the functions from the loop_helix_loop_reshaping package to perform user customized jobs.

The job_scripts directory contains examples of job scripts for each step of the LUCS pipeline.

It should be straightforward for a user to copy the example job scripts and edit the inputs, outputs and parameters.

run_jobs.py

Unified interface for running the code locally and on the cluster.

Usage is documented in the script.

Examples:

```
./run_jobs.py my_output_data_set job_scripts/user/my_script.py

./run_jobs.py my_output_data_set job_scripts/user/my_script.py -d SGE -n 100
```

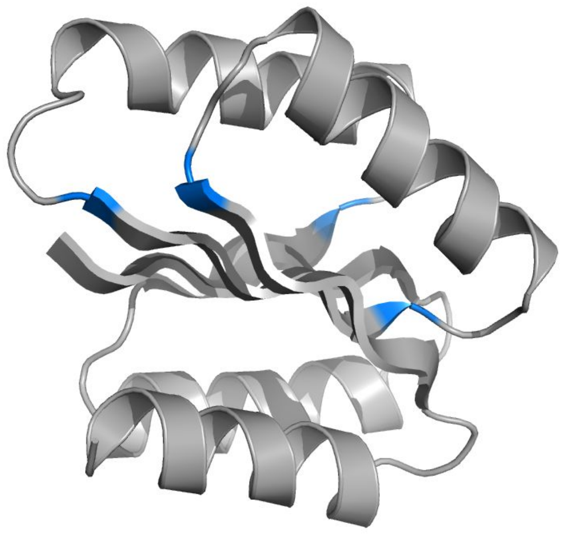
Use LUCS

The README.md files has instructions for installing the package and running an example.

A user can easily understands the LUCS pipeline by running the example.

Here I will go through the example with more detailed explanation of each step.

Select linker loops: inputs



A PDB file of the scaffold structure

A json file that defines the insertion points

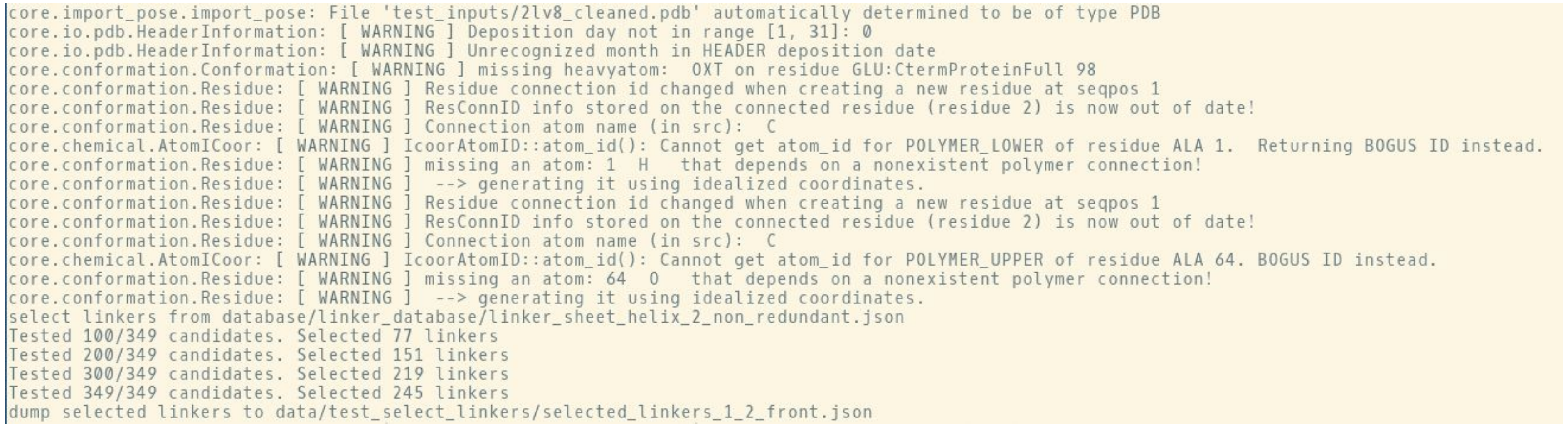
```
[
{"start":31, "stop":51, "start_ss":"sheet", "stop_ss":"sheet"},
{"start":58, "stop":76, "start_ss":"sheet", "stop_ss":"sheet"}
]
```

Residue numbers are Rosetta numbering instead of PDB numbering.

Residue 31, 51, 58 and 76 are the anchoring positions on the sheets instead of positions on the loop

Select linker loops: command line

./run_jobs.py test_select_linkers job_scripts/select_linkers_example.py

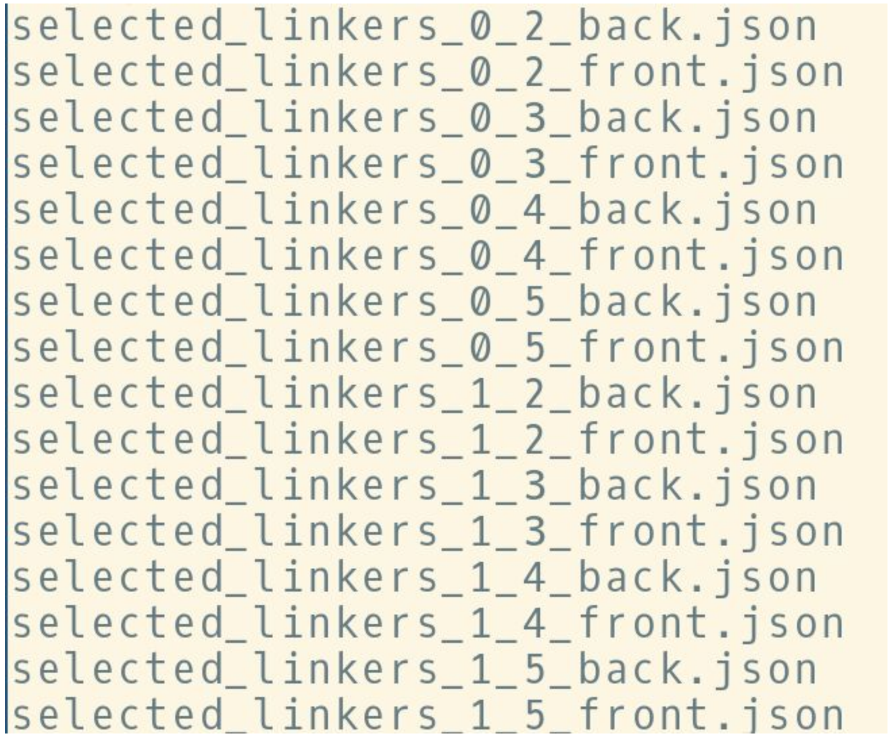


```
core.import_pose.import_pose: File 'test_inputs/2lv8_cleaned.pdb' automatically determined to be of type PDB
core.io.pdb.HeaderInformation: [ WARNING ] Deposition day not in range [1, 31]: 0
core.io.pdb.HeaderInformation: [ WARNING ] Unrecognized month in HEADER deposition date
core.conformation.Conformation: [ WARNING ] missing heavyatom:  OXT on residue GLU:CtermProteinFull 98
core.conformation.Residue: [ WARNING ] Residue connection id changed when creating a new residue at seqpos 1
core.conformation.Residue: [ WARNING ] ResConnID info stored on the connected residue (residue 2) is now out of date!
core.conformation.Residue: [ WARNING ] Connection atom name (in src):  C
core.chemical.AtomICoor: [ WARNING ] IcoorAtomID::atom_id(): Cannot get atom_id for POLYMER_LOWER of residue ALA 1.  Returning BOGUS ID instead.
core.conformation.Residue: [ WARNING ] missing an atom: 1  H   that depends on a nonexistent polymer connection!
core.conformation.Residue: [ WARNING ]  --> generating it using idealized coordinates.
core.conformation.Residue: [ WARNING ] Residue connection id changed when creating a new residue at seqpos 1
core.conformation.Residue: [ WARNING ] ResConnID info stored on the connected residue (residue 2) is now out of date!
core.conformation.Residue: [ WARNING ] Connection atom name (in src):  C
core.chemical.AtomICoor: [ WARNING ] IcoorAtomID::atom_id(): Cannot get atom_id for POLYMER_UPPER of residue ALA 64. BOGUS ID instead.
core.conformation.Residue: [ WARNING ] missing an atom: 64  O   that depends on a nonexistent polymer connection!
core.conformation.Residue: [ WARNING ]  --> generating it using idealized coordinates.
select linkers from database/linker_database/linker_sheet_helix_2_non_redundant.json
Tested 100/349 candidates. Selected 77 linkers
Tested 200/349 candidates. Selected 151 linkers
Tested 300/349 candidates. Selected 219 linkers
Tested 349/349 candidates. Selected 245 linkers
dump selected linkers to data/test_select_linkers/selected_linkers_1_2_front.json
```

This step is fast. Even for a production run, you can do it on a local machine.

Select linker loops: outputs

data/test_select_linkers directory



```
selected_linkers_0_2_back.json
selected_linkers_0_2_front.json
selected_linkers_0_3_back.json
selected_linkers_0_3_front.json
selected_linkers_0_4_back.json
selected_linkers_0_4_front.json
selected_linkers_0_5_back.json
selected_linkers_0_5_front.json
selected_linkers_1_2_back.json
selected_linkers_1_2_front.json
selected_linkers_1_3_back.json
selected_linkers_1_3_front.json
selected_linkers_1_4_back.json
selected_linkers_1_4_front.json
selected_linkers_1_5_back.json
selected_linkers_1_5_front.json
```

In selected_linkers_0_2_back.json, 0 is the index of the insertion point; 2 is the loop length and “back” means the loop is at the downstream of the helix to be modeled.

[{"sequence": ["I", "G", "T", "G"], "pdb_id": "7odc", "phis": [-81.685, 93.898, -85.0, -89.047], "psis": [-7.55, 32.441, 162.558, 178.074], "start_position": 48, "omegas": [161.083, 172.635, 169.776, -178.366]}, …]

Screen loop helix loop units: inputs

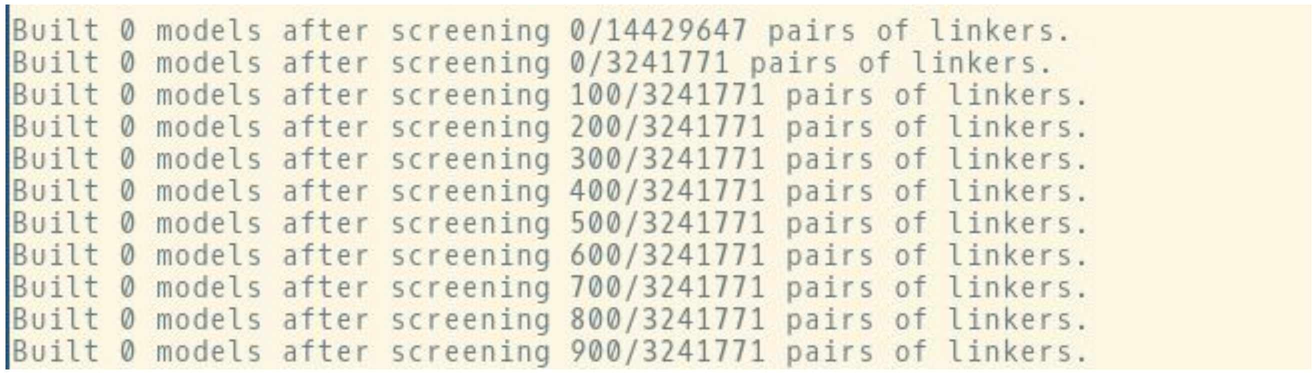
The PDB file of the scaffold structure (same as in the select linker loops step).

The json file that defines the insertion points (same as in the select linker loops step).

The json linker files generated by the select linker loops step.

Screen loop helix loop units: command line

./run_jobs.py test_screen_single_insertion_loop_helix_loop_units job_scripts/screen_single_insertion_loop_helix_loop_units_example.py



```
Built 0 models after screening 0/14429647 pairs of linkers.
Built 0 models after screening 0/3241771 pairs of linkers.
Built 0 models after screening 100/3241771 pairs of linkers.
Built 0 models after screening 200/3241771 pairs of linkers.
Built 0 models after screening 300/3241771 pairs of linkers.
Built 0 models after screening 400/3241771 pairs of linkers.
Built 0 models after screening 500/3241771 pairs of linkers.
Built 0 models after screening 600/3241771 pairs of linkers.
Built 0 models after screening 700/3241771 pairs of linkers.
Built 0 models after screening 800/3241771 pairs of linkers.
Built 0 models after screening 900/3241771 pairs of linkers.
```

Don’t worry that there are always “0 models”. This is because we have max_num_success_each_db_pair=1 . Once the script successful build a model, it will break to the next pair of linker databases. In a production run, you would like to set max_num_success_each_db_pair=None.

This is the most time consuming step. You would like to use 1,000 CPUs in parallel for a production run.

Screen loop helix loop units: outputs

data/test_screen_single_insertion_loop_helix_loop_units directory

```
selected_lhl_units_0_0.json  selected_lhl_units_1_0.json
```

In selected_lhl_units_0_0.json, the first 0 is the index of the insertion point; the second 0 is the ID of the job that generated this data.

[{"phis": [...], "psis": [...], "omegas": [...], "cutpoint": 7}, …]

Cutpoint is the position where the two half helices meet.

When running in parallel, each job would write N output files where N is the number of insertion points. You can merge these files using the script job_scripts/merge_lhl_unit_libraries.py

Screen compatible loop helix loop units: inputs

The PDB file of the scaffold structure (same as in the select linker loops step).

The json file that defines the insertion points (same as in the select linker loops step).

The json LHL unit files generated by the screen loop helix loop units step.

Screen compatible loop helix loop units: command line

```
./run_jobs.py test_screen_compatible_loop_helix_loop_units
job_scripts/screen_compatible_loop_helix_loop_units_example.py
```

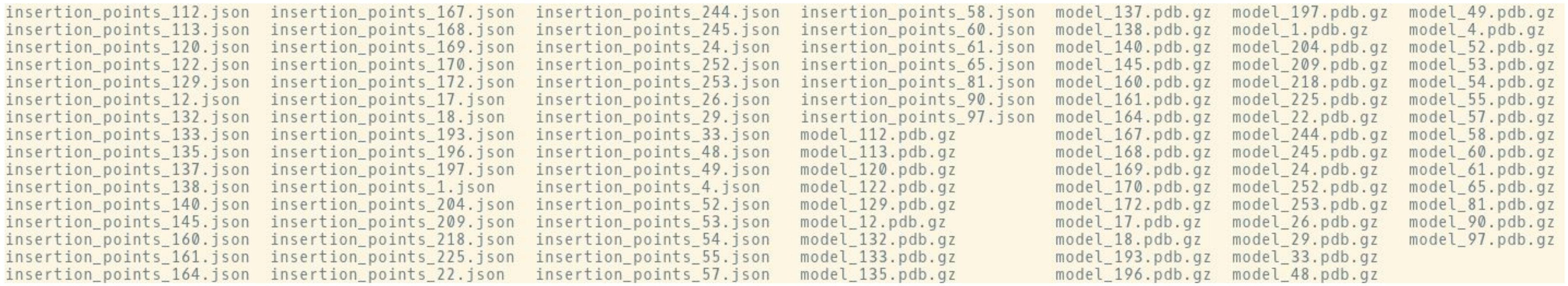
The speed of this step highly depends on number of combinations of LHL units.

If there are to many possible combinations, I usually pass max_num_to_screen=1,000,000 to the screen_compatible_loop_helix_loop_units.screen_compatible_loop_helix_loop_units() function.

Using 1,000 CPUs, 1,000,000 combinations can be screened within a few hours.

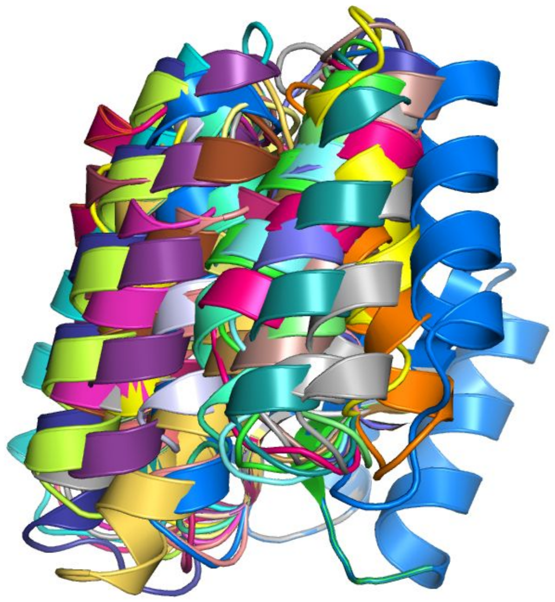
Screen compatible loop helix loop units: outputs

data/test_screen_compatible_loop_helix_loop_units directory.



```
insertion_points_112.json  insertion_points_167.json  insertion_points_244.json  insertion_points_58.json  model_137.pdb.gz  model_197.pdb.gz  model_49.pdb.gz
insertion_points_113.json  insertion_points_168.json  insertion_points_245.json  insertion_points_60.json  model_138.pdb.gz  model_1.pdb.gz    model_4.pdb.gz
insertion_points_120.json  insertion_points_169.json  insertion_points_24.json   insertion_points_61.json  model_140.pdb.gz  model_204.pdb.gz  model_52.pdb.gz
insertion_points_122.json  insertion_points_170.json  insertion_points_252.json  insertion_points_65.json  model_145.pdb.gz  model_209.pdb.gz  model_53.pdb.gz
insertion_points_129.json  insertion_points_172.json  insertion_points_253.json  insertion_points_81.json  model_160.pdb.gz  model_218.pdb.gz  model_54.pdb.gz
insertion_points_12.json   insertion_points_17.json   insertion_points_26.json   insertion_points_90.json  model_161.pdb.gz  model_225.pdb.gz  model_55.pdb.gz
insertion_points_132.json  insertion_points_18.json   insertion_points_29.json   insertion_points_97.json  model_164.pdb.gz  model_22.pdb.gz   model_57.pdb.gz
insertion_points_133.json  insertion_points_193.json  insertion_points_33.json   model_112.pdb.gz          model_167.pdb.gz  model_244.pdb.gz  model_58.pdb.gz
insertion_points_135.json  insertion_points_196.json  insertion_points_48.json   model_113.pdb.gz          model_168.pdb.gz  model_245.pdb.gz  model_60.pdb.gz
insertion_points_137.json  insertion_points_197.json  insertion_points_49.json   model_120.pdb.gz          model_169.pdb.gz  model_24.pdb.gz   model_61.pdb.gz
insertion_points_138.json  insertion_points_1.json    insertion_points_4.json    model_122.pdb.gz          model_170.pdb.gz  model_252.pdb.gz  model_65.pdb.gz
insertion_points_140.json  insertion_points_204.json  insertion_points_52.json   model_129.pdb.gz          model_172.pdb.gz  model_253.pdb.gz  model_81.pdb.gz
insertion_points_145.json  insertion_points_209.json  insertion_points_53.json   model_12.pdb.gz           model_17.pdb.gz   model_26.pdb.gz   model_90.pdb.gz
insertion_points_160.json  insertion_points_218.json  insertion_points_54.json   model_132.pdb.gz          model_18.pdb.gz   model_29.pdb.gz   model_97.pdb.gz
insertion_points_161.json  insertion_points_225.json  insertion_points_55.json   model_133.pdb.gz          model_193.pdb.gz  model_33.pdb.gz
insertion_points_164.json  insertion_points_22.json   insertion_points_57.json   model_135.pdb.gz          model_196.pdb.gz  model_48.pdb.gz
```

For each compatible combination of LHL units, the script writes a PDB file and a json file recording the insertion points (models can have different insertion points because the LHL unit lengths are different).



Sequence design

I use code in the https://github.com/Kortemme-Lab/local_protein_sequence_design repository for designing sequences to stabilize the backbone structures.

What would the future be?

LUCS proved a new paradigm for protein backbone sampling. As a prototype, the method has many limitations:

- LUCS does not work for other types of secondary structures.
- The LHL units must be anchored to fixed secondary structure elements.
- When modeling 3 or more LHL units simultaneously, the number of combinations is too big to enumerate.
- LUCS needs to evaluate if a backbone can be stabilized by at least one sequence without running sequence design. The designability is crudely estimated by contact degree and steric clashes after mutating side chains to valines.

I implemented LUCS in PyRosetta because python is easy for developers to implement new ideas. I expect the next generation LUCS or new methods can address this limitations.

What would the future be? (cont.)

For the next generation LUCS, one can:

- Apply LUCS to more protein fold families and optimize its parameters.
- Use stochastic sampling for compatibility test to address the combinatorial problem.
- Couple LHL sampling with sidechain design.
- Use sidechain independent score functions to filter designable structures.

One possible way to sample arbitrary secondary structures is combining tertiary structure motifs (TERMs) and loop modeling. One can remove the loops in a region of interest, sample secondary structure geometry with TERMs and reintroduce the loops using loop hash KIC.

Generative machine learning methods are also potential solutions.