Ensembler Documentation

Release 1.0

Daniel L. Parton, John D. Chodera

October 16, 2015

Contents

Python application and library which generates diverse protein configurational ensembles suitable for seeding highly parallel molecular simulations.

Protein models are generated by mapping a set of target sequences onto a set of template structures, with a series of successive refinement and filtering stages to optimize model quality. This:

- Exploits the entire variety of available genomic and structural data to provide diverse arrays of high-quality configurational models
- Scales to allow the study of entire (super)families
- Automates much of the time-consuming process of setting up protein systems for molecular simulation

The resulting models can be used as starting configurations for highly parallel molecular simulations, which take advantage of modern high-performance computing architectures. This approach is of particular benefit when used in conjunction with recent techniques which can combine data from multiple independent trajectories to produce kinetic models, such as Markov state models.

Ensembler can be used via the command-line application (`ensembler`) or scripted via the API, and can be run on a single computer or on a parallel compute cluster. It makes use of a number of external packages:

- Modeller for comparative modeling of target sequences onto template structures
- Rosetta loopmodel for reconstruction of missing template loops
- OpenMM for model refinement with highly efficient, GPU-acclerated, molecular dynamics simulation
- MDTraj for trajectory-manipulation and fast RMSD calculation
- MSMBuilder for clustering

CHAPTER 1

Overview of the Ensembler Pipeline

The modeling and refinement process comprises a series of successive stages:

1. Retrieval of protein target sequences and template structures, e.g. from UniProt and the PDB
2. (*optional*) Reconstruction of missing template loops, using `Rosetta loopmodel`
3. Model generation - each target sequence is mapped onto each available template structure, using `Modeller`
4. Culling of models based on close structural similarity
5. Refinement with implicit solvent molecular dynamics simulation, using `OpenMM`
6. (*optional*) Solvation of models with explicit water
7. (*optional*) Refinement with explicit solvent molecular dynamics simulation, using `OpenMM`
8. (*optional*) Packaging of models, ready for transfer or set-up on production simulation platforms such as `Folding@Home`

CHAPTER 2

License

Ensembler is licensed under the GNU General Public License (GPL) v2.0.

2.1 Installation

2.1.1 Platforms

Ensembler is developed and tested on Linux and Mac. Windows is not well supported.

Supported Hardware

The simulation refinement stages of Ensembler use OpenMM, which performs best on CUDA-enabled GPUs. However, OpenMM also has an optimized CPU platform, so use of GPUs is optional.

Install with Conda

conda is a package manager built for scientific Python. Unlike `easy_install` or `pip`, it handles binaries and binary dependencies, which are critical for most scientific workflows.

Conda can be obtained by installing Continuum Anaconda - an awesome free Python distribution for scientific computing. The standard installation contains many of Ensembler's dependencies. Alternatively, for a more minimal Python set-up you can install miniconda, which contains only Conda and Python.

First get a license for Modeller from the Modeller website (registration required; free for academic non-profit use).

Save this in an environment variable as follows

```
$ export KEY_MODELLER=XXX
```

Then, to install Ensembler with Conda, use the following commands

```
$ conda config --add channels http://conda.anaconda.org/omnia
$ conda config --add channels http://conda.anaconda.org/salilab
$ conda install ensembler
```

Conda will automatically install all dependencies except for the optional dependencies Rosetta and MolProbity. These require licenses (free for academic non-profit use), and will have to be installed according to the instructions for those packages. Some limited installation instructions are included below, but these are not guaranteed to be up to date.

Install from Source

Clone the source code repository from github

```
$ git clone git://github.com/choderalab/ensembler.git
```

Then, in the directory containing the source code, you can install it with.

```
$ python setup.py install
```

2.1.2 Dependencies

To use Ensembler, the following libraries and software will need to be installed.

Linux, Mac OS X or Windows operating system We develop mainly on 64-bit Linux and Mac machines. Windows is not well supported.

Python >= 2.6 The development package (`python-dev` or `python-devel` on most Linux distributions) is recommended.

Modeller Comparative modeling of protein structures.

OpenMM Molecular simulation toolkit, with GPU-accelerated simulation platform.

MDTraj Simulation trajectory analysis library.

MSMBuilder Statistical models for biomolecular dynamics.

PDBFixer PDB structure modeling

BioPython Collection of Python tools for computational biology and bioinformatics.

NumPy Numpy is the base package for numerical computing in Python.

lxml For working with XML files.

PyYAML For working with YAML files.

docopt For building command-line interfaces.

mock For testing in Python

Optional packages:

MPI4Py Allows many Ensembler functions to be run in parallel using MPI.

Rosetta Protein modeling suite. The `loopmodel` function is optionally used by Ensembler to reconstruct missing loops in template structures.

subprocess32 (If running Python 2.) Backport of the Python 3 subprocess module for Python 2. Used to run command-line programs such as Rosetta loopmodel. Includes timeout functionality which is particularly useful for the template loop reconstruction routine.

Pandas Some functionality, including the `quickmodel` and `inspect` functions, requires pandas.

MolProbity For model validation. The `package_models` function can use this data to filter models by validation score.

Manually Installing the Dependencies

Linux

If you're on ubuntu and have root, you can install most dependencies through your package manager (`apt-get`).

```
$ sudo apt-get install python-dev
```

Mac

If you're on mac and want a package manager, you should be using homebrew and `brews`'s Python (see this page for details). For example, numpy can be installed with `brew` as follows:

```
$ brew tap Homebrew/python
$ brew install python
$ brew install numpy
```

Then, you can install many of the remaining packages with `pip`.

```
$ pip install lxml
```

Windows

Chris Gohlke maintains windows binary distributions for an ever-growing set of Python extensions on his website. Download and install the the installers for setuptools, nose, numpy, scipy, numexpr, pandas and tables.

2.1.3 Testing Your Installation

Running the tests is a great way to verify that everything is working. The test suite uses nose and mock, which you can pick up via conda or pip if you don't already have them.

```
$ conda install nose mock
```

To run the unit tests:

```
$ nosetests ensembler -a unit
```

Further tests are available which check interoperation of Ensembler with software dependencies such Rosetta loop-model, or with external public databases such as UniProt, or are excluded from the unit tests due to being slow. To run them:

```
$ nosetests ensembler -a non_conda_dependencies -a network -a slow
```

2.1.4 Installation of Dependencies Unavailable Through Conda

(Note: only limited instructions are included here, and these are not guaranteed to be up to date. If you encounter problems, please consult the relevant support or installation instructions for that software dependency.)

MolProbity

Download the MolProbity 4.2 release source from the GitHub repo.

Extract the zip file, enter the created directory, and run the following command:

```
$ ./configure.sh
```

This was all that was required when tested on a MacBook running OS X 10.8.

On a Linux cluster, it was first necessary to edit the file configure.sh to uncomment the following line, and comment the `make` command:

```
$ ./binlibtbx.scons -j 1
```

This forces the build to use only a single core - this ran rather slowly, but using more cores resulted in build failure. This is likely due to memory issues. After runnng `./configure.sh` it was then also necessary to run `./setup.sh`.

Binaries can found in the `[MolProbity source dir]/cmdline` directory.

2.2 Usage Examples

There are two main ways to use the Ensembler command-line interface. The `quickmodel` function performs the entire modeling pipeline in one go, and is designed to work with a single target and a small number of templates. For generating larger numbers of models (such as entire protein families), the main pipeline functions should be used. These perform each stage of the modeling process individually, and the most computationally intensive stages can be run in parallel to increase performance.

For further details on their usage, see the main command-line interface documentation.

2.2.1 Example using the quickmodel function

```
$ ensembler quickmodel --target_uniprot_entry_name EGFR_HUMAN --uniprot_domain_regex '^Protein kinase
```

Models human EGFR onto two templates selected via PDB IDs. The `quickmodel` function executes the entire modeling pipeline in one go, and is designed to work with only a few targets and templates. For generating larger numbers of models (such as entire protein families), the main pipeline functions should be used.

2.2.2 Example using the main pipeline functions

```
$ ensembler init
```

This sets up an Ensembler project in the current working directory. It creates a number of directories and a metadata file (meta0.yaml).

```
$ ensembler gather_targets --gather_from uniprot --query 'domain:"Protein kinase" AND taxonomy:9606 A
```

Queries UniProt for all human protein kinases, and selects the domains of interest, as specified by the regular expression ("regex") passed to the final flag. At the time this documentation was written, five types of protein kinase domain were returned by the UniProt search, annotated as "Protein kinase", "Protein kinase; 1", "Protein kinase; 2", "Protein kinase; truncated", and "Protein kinase; inactive". The above regex selects the first three types of domain, and excludes the latter two. Sequences are written to a fasta file: `targets/targets.fa`.

Targets are given IDs of the form `[UniProt mnemonic]_D[domain id]`, which consists of the UniProt name for the target and an identifier for the domain (since a single target protein may contain multiple domains of interest). Example: `EGFR_HUMAN_D0`.

```
$ ensembler gather_templates --gather_from uniprot --query 'domain:"Protein kinase" AND reviewed:yes
```

Queries UniProt for all protein kinases (of any species), selects the relevant domains, and retrieves sequence data and a list of associated PDB structures (X-ray and NMR only), which are then downloaded from the PDB. Template sequences are written in two forms - the first contains only residues resolved in the structure (`templates/templates-resolved-seq.fa`); the second contains the complete UniProt sequence containined within the span of the structure, including unresolved residues (`templates/templates-full-seq.fa`). Template structures (containing only resolved residues) are extracted and written to the directory `templates/structures-resolved`. Templates containing the full sequences can optionally be generated with a subsequent step - the `loopmodel` function.

Templates are given IDs of the form `[UniProt mnemonic]_D[domain id]_[PDB id]_[chain id]`, where the final two elements represent the PBD ID and chain identifier. Example: `EGFR_HUMAN_D0_2GS7_B`.

```
$ ensembler loopmodel
```

(*Optional*) Reconstruct template loops which were not resolved in the original PDB structure, using `Rosetta loopmodel`. This tends to result in higher quality models. The reconstructed template structures are written to the directory `templates/structures-modeled-loops`.

```
$ ensembler align
```

Conducts pairwise alignments of target sequences against template sequences. These alignments are used to guide the subsequent modeling step, and are stored in directories of the form `models/[target id]/[template id]/alignment.pir`. The `.pir` alignment format is an ascii-based format required by `Modeller`.

If the `loopmodel` function was used previously, then templates which have been successfully remodeled will be selected for this alignment and the subsequent modeling steps. Otherwise, Ensembler defaults to using the template structures which contain only resolved residues.

```
$ ensembler build_models
```

Creates models by mapping each target sequence onto each template structure, using the `Modeller` automodel function.

```
$ ensembler cluster
```

Filters out non-unique models by clustering on RMSD. A default cutoff of 0.06 nm is used. Unique models are given an empty file `unique_by_clustering` in their model directory.

```
$ ensembler refine_implicit
```

Refines models by performing an energy minimization followed by a short molecular dynamics simulation (default: 100 ps) with implicit solvent (Generalized Born surface area), using `OpenMM`. The final structure is written to the compressed PDB file `implicit-refined.pdb.gz`.

```
$ ensembler solvate
```

Determines the number of waters to add when solvating models with explicit water molecules. The models for each target are given the same number of waters. The function proceeds by first solvating each model individually, given a padding distance (default: 1 nm). A list of the number of waters added for each model is written to a file `nwaters.txt` in the `models/[target_id]` directory. A percentile value from the distribution of the number of waters is selected as the number to use for all models, and this number is written to the file `nwaters-use.txt`.

```
$ ensembler refine_explicit
```

Solvates models using the number of waters determined in the previous step, then performs a short molecular dynamics simulation (default: 100 ps), using `OpenMM`. The final structure is written to the compressed PDB file: `explicit-refined.pdb.gz`, as well as serialized versions of the OpenMM System, State and Integrator objects.

```
$ ensembler validate
```

(Optional; requires MolProbity command-line tools) Validates model quality using MolProbity, which uses criteria such as Ramachandran angles, backbone distortions, and atom clashes. The `package_models` command can filter models based on validation score, using the `--model_validation_score_cutoff` and `--model_validation_score_percentile` flags.

```
$ ensembler package_models --package_for FAH --nfahclones 3
```

Packages models in the necessary directory and file structure to be run as Folding@Home projects. Files are written in the directory tree `packaged_models/fah-projects/[target id]`.

2.3 Command-Line Interface Documentation

2.3.1 Overview

Ensembler can be used via the command-line tool `ensembler` or via the API.

For API documentation, see the source code.

The `ensembler` tool is operated via a number of subcommands, which should be executed successively

```
ensembler init
ensembler gather_targets
ensembler gather_templates
ensembler loopmodel
ensembler align
ensembler build_models
ensembler cluster
ensembler refine_implicit
ensembler solvate
ensembler refine_explicit
ensembler package_models
```

The optional `ensembler validate` subcommand uses the MolProbity command-line tools to conduct model quality validation based on criteria such as Ramachandran angles, backbone distortion, and atom clashes.

The `ensembler quickmodel` subcommand allows the entire modeling pipeline to be run in one go for a single target and a small number of templates. Note that this command will not work with MPI.

To print helpstrings for each subcommand, pass the `-h` flag.

If desired, target-selection and template-selection can be set up manually, rather than using the `gather_targets` and `gather_templates` subcommands. Targets should be provided as a fasta-format file (`targets/targets.fa`) containing target sequences and arbitrary identifiers. Template sequences and arbitrary identifiers should be provided in a fasta-format file (`templates/templates-resolved-seq.fa`), and structures should be provided as PDB-format coordinate files in the directory `templates/structures-resolved`. Each structure should be named XXX.pdb, where XXX matches the identifier in the fasta file. The residues in the coordinate files should also match the sequences in the fasta file.

2.3.2 Custom settings

Many aspects of the behavior of Ensember can be specified by using the Python API instead of the main command-line interface. For API documentation, see the source code, or view the docstrings in iPython.

Custom settings via the manual_overrides.yaml file

Some options can instead be specified via the `manual_overrides.yaml` file, which is created when initializing a new Ensembler project. The file contains an example configuration, with each line commented out. The user can thus uncomment the relevant lines and edit as necessary.

```
target-selection:
    domain-spans:
      ABL1_HUMAN_D0: 242-513
template-selection:
    min-domain-len: 0
    max-domain-len: 350
    domain-spans:
        ABL1_HUMAN_D0: 242-513
    skip-pdbs:
        - 4CYJ
        - 4P41
        - 4P2W
        - 4QTD
        - 4Q2A
        - 4CTB
        - 4QOX
refinement:
    ph: 8.0
    custom_residue_variants:
        DDR1_HUMAN_D0_PROTONATED:
            # keyed by 0-based residue index
            35: ASH
```

The above configuration makes the following specifications (in order of appearance):

- Specifies a custom residue span for the target `ABL1_HUMAN_D0`. This is useful in cases where a different domain span is desired from that annotated in UniProt.
- Specifies minimum and maximum domain lengths for templates. Any domain with more than 350 residues would be excluded. The same custom residue span used for target domains is also specified for the template domains.
- Certain PDB files can be skipped if they cause problems.
- A custom pH level (default: 7.0) is set, which determines how protonation states are assigned by OpenMM prior to molecular dynamics refinement.
- Custom residue variants are specified. This can be used to set specific protonation states, rather than rely purely on a defined pH level. These specified protonation states would override those determined by pH. The naming of residue variants (e.g. `ASH`) follows the OpenMM conventions.

2.3.3 Additional Tools

Ensembler includes a `tools` submodule, which allows the user to conduct various useful tasks which are not considered core pipeline functions. The use-cases for many of these tools are quite specific, so they may not be applicable to every project, and should be used with caution.

Residue renumbering according to UniProt sequence coordinates

```
$ ensembler renumber_residues --target EGFR_HUMAN_D0
```

The given target ID must begin with a UniProt mnemonic, e.g. "EGFR_HUMAN". This will output two files in the `models/[target_id]` directory: `topol-renumbered-implicit.pdb` and `topol-renumbered-explicit.pdb`. The coordinates are simply copied from the first example found for each of `refined-implicit.pdb.gz` and `refined-explicit.pdb.gz`. The residue numbers are renumbered according to the canonical isoform sequence coordinates in the UniProt entry.

Generating unrefined model structures

In some cases it may be useful to analyze model structures which have not undergone refinement, but which have topologies equivalent to the final refined models. These structures are not saved by the main pipeline functions by default, but can be regenerated using `ensembler.tools.mktraj.MkTrajImplicitStart`. This code simply loads each model structure with `openmm`, adds hydrogens, and writes the resultant structure as a pdb file (`implicit-start.pdb.gz`). It also combines the structures into a trajectory (`traj-implicit-start.xtc`). This function is accessed via the Python API as follows:

```
from ensembler.tools.mktraj import MkTrajImplicitStart
MkTrajImplicitStart(targetid='EGFR_HUMAN_D0')
```