ssbio Documentation

Release 1.0.0a0

Nathan Mih

Jul 19, 2017

Contents

CHAPTER 1

Introduction

This Python package provides a collection of tools for people with questions in the realm of structural systems biology. The main goals of this package are to:

1. Provide an easy way to map proteins to sequences and structures
2. Directly link structures to genome-scale SBML models
3. Prepare structures for downstream analyses, such as their use in molecular modeling software

Example questions you can answer with this package:

- How can I determine the number of protein structures available for my list of genes?
- What is the best, representative structure for my protein?
- Where, in a metabolic network, do these proteins work?
- Where do popular mutations show up on a protein?
- How can I compare the structural features of entire proteomes?
- and more...

CHAPTER 2

Installation

First install NGLview using pip:

```
pip install nglview
```

Then install ssbio:

```
pip install ssbio
```

Updating

```
pip install ssbio --upgrade
```

Uninstalling

```
pip uninstall ssbio
```

Dependencies

See: Software Installations for additional programs to install.

CHAPTER 3

Tutorials

Check out some Jupyter notebook tutorials at *The Protein Class* and *The GEM-PRO Pipeline*.

CHAPTER 4

Citation

Currently, use of this package can be cited by our 2016 paper in BMC Systems Biology[1], which details the GEM-PRO pipeline. The manuscript for the `ssbio` package itself is in preparation at this moment.

[1] Brunk, E.*, Mih, N.*, Monk, J., Zhang, Z., O'Brien, E. J., Bliven, S. E., Bourne, P. E., Palsson, B. O. (2016). Systems biology of the structural proteome. BMC Systems Biology, 10(1), 26. http://doi.org/10.1186/s12918-016-0271-6. *Authors contributed equally.

CHAPTER 5

Index

Getting Started

Introduction

This section will give a quick outline of the design of `ssbio` and the scientific topics behind it.

The basics

`ssbio` was developed with simplicity in mind, and also as a direct extension of COBRApy. Furthermore, we didn't want to reinvent the wheel wherever possible, and thus Biopython classes and modules are used wherever possible.

Structural biology

This section will give a very brief background on some structural biology topics.

Protein structures

Systems biology

This section will give a very brief background on some systems biology topics.

COBRA models & COBRApy

COBRA models & COBRApy

Reading

The StructProp Class

Introduction

This section will give an overview of the methods that can be executed for a single protein structure.

The Protein Class

Introduction

This section will give an overview of the methods that can be executed for the Protein class, which is a basic representation of a protein by a collection of amino acid sequences and 3D structures. A Jupyter notebook tutorial is available below.

Tutorials

Protein - Structure Mapping, Alignments, and Visualization

This notebook gives an example of how to **map a single protein sequence to its structure**, along with conducting sequence alignments and visualizing the mutations.

Input: Protein ID + amino acid sequence + mutated sequence(s)

Output: Representative protein structure, sequence alignments, and visualization of mutations

---

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the Protein class
        from ssbio.core.protein import Protein
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown

- ERROR
  - Major errors
- WARNING
  - Warnings that don't affect running of the pipeline
- INFO (default)
  - Info such as the number of structures mapped per gene
- DEBUG
  - Really detailed information that will print out a lot of stuff

Warning: DEBUG mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- ROOT_DIR
  - The directory where a folder named after your PROTEIN_ID will be created
- PROTEIN_ID
  - Your protein ID
- PROTEIN_SEQ
  - Your protein sequence

A directory will be created in ROOT_DIR with your PROTEIN_ID name. The folders are organized like so:

```
ROOT_DIR
- PROTEIN_ID
    - sequences  # Protein sequence files, alignments, etc.
    - structures  # Protein structure files, calculations, etc.
```

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROTEIN_ID = 'SRR1753782_00918'
        PROTEIN_SEQ = 'MSKQQIGVVGMAVMGRNLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAGA
```

```
In [7]: # Create the Protein object
        my_protein = Protein(ident=PROTEIN_ID, root_dir=ROOT_DIR)
```

```
In [8]: # Load the protein sequence
        # This sets the loaded sequence as the representative one
        my_protein.load_manual_sequence(seq=PROTEIN_SEQ, ident='WT', write_fasta_file=True, set_as_re
```

```
Out[8]: <SeqProp WT at 0x7f0685574128>
```

Mapping sequence –> structure

Since the sequence has been provided, we just need to BLAST it to the PDB.

Note: These methods do not download any 3D structure files.

Methods

Protein.**blast_representative_sequence_to_pdb**(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)

BLAST repseq to PDB and return the list of new structures added, also saves df_pdb_blast

```
In [9]: # Mapping using BLAST
        my_protein.blast_representative_sequence_to_pdb(seq_ident_cutoff=0.9, evalue=0.00001)
        my_protein.df_pdb_blast.head()
```

```
Out[9]: ['2zyd', '2zya', '3fwn', '2zyg']
```

```
Out[9]: pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
        pdb_id
        2zya              A     2319.0         0.0             0.987179
        2zya              B     2319.0         0.0             0.987179
        2zyd              A     2319.0         0.0             0.987179
        2zyd              B     2319.0         0.0             0.987179
        2zyg              A     2284.0         0.0             0.982906

                hit_percent_ident  hit_num_ident  hit_num_similar
        pdb_id
        2zya             0.963675            451              462
        2zya             0.963675            451              462
        2zyd             0.963675            451              462
        2zyd             0.963675            451              462
        2zyg             0.950855            445              460
```

Downloading and ranking structures

Methods

Protein.**pdb_downloader_and_metadata**(*outdir=None*, *pdb_file_type=None*, *force_rerun=False*)

Download experimental structures

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [10]: # Download all mapped PDBs and gather the metadata
         my_protein.pdb_downloader_and_metadata()
         my_protein.df_pdb_metadata.head(2)

Out[10]: ['2zyd', '2zya', '3fwn', '2zyg']

Out[10]: pdb_title  \
         pdb_id
         2zyd    Dimeric 6-phosphogluconate dehydrogenase compl...
         2zya    Dimeric 6-phosphogluconate dehydrogenase compl...

                                                 description experimental_method  \
         pdb_id
         2zyd    6-phosphogluconate dehydrogenase, decarboxylat...   X-RAY DIFFRACTION
         2zya    6-phosphogluconate dehydrogenase, decarboxylat...   X-RAY DIFFRACTION

                mapped_chains  resolution chemicals                          date  \
         pdb_id
         2zyd             A;B         1.5       GLO  2009-09-01;2011-07-13;2014-01-22
         2zya             A;B         1.6       6PG  2009-09-01;2011-07-13;2014-01-22

                    taxonomy_name structure_file
         pdb_id
         2zyd    Escherichia coli       2zyd.cif
         2zya    Escherichia coli       2zya.cif
```

Protein.**set_representative_structure**(*seq_outdir=None*, *struct_outdir=None*, *pdb_file_type=None*, *engine='needle'*, *always_use_homology=False*, *rez_cutoff=0.0*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*, *clean=True*, *keep_chemicals=None*, *force_rerun=False*)

Set a representative structure from a structure in self.structures

Parameters

- **seq_outdir** (*str*) – Path to output directory of sequence alignment files
- **struct_outdir** (*str*) – Path to output directory of structure files
- **pdb_file_type** (*str*) – pdb, pdb.gz, mmcif, cif, cif.gz, xml.gz, mmtf, mmtf.gz - PDB structure file type that should be downloaded
- **engine** (*str*) – "needle" or "biopython" - which pairwise sequence alignment engine should be used needle is the standard EMBOSS tool to run pairwise alignments biopython is Biopython's implementation of needle. Results can differ!
- **always_use_homology** (*bool*) – If homology models should always be set as the representative structure
- **rez_cutoff** (*float*) – Resolution cutoff, in Angstroms (only if experimental structure)
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for

- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for
- **clean** (*bool*) – If structure should be cleaned
- **keep_chemicals** (*str, list*) – Keep specified chemical names if structure is to be cleaned
- **force_rerun** (*bool*) – If sequence to structure alignment should be rerun

Returns

Representative structure from the list of structures.

This is a not a map to the original structure, it is copied from its reference.

Return type *StructProp*

```
In [11]: # Set representative structures
         my_protein.set_representative_structure()
```

```
Out[11]: <StructProp 2zyd-A at 0x7f8e304efc50>
```

Loading and aligning new sequences

You can load additional sequences into this protein object and align them to the representative sequence.

Methods

Protein.**load_manual_sequence**(*seq*, *ident=None*, *write_fasta_file=False*, *outname=None*, *outdir=None*, *set_as_representative=False*, *force_rewrite=False*)

Load a manual sequence given as a string and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **seq** (*str, Seq, SeqRecord*) – Sequence string, Biopython Seq or SeqRecord object
- **ident** (*str*) – Optional identifier for the sequence, required if seq is a string. Also will override existing IDs in Seq or SeqRecord objects if set.
- **write_fasta_file** –
- **outname** –
- **outdir** –
- **set_as_representative** –
- **force_rewrite** –

Returns:

```
In [12]: # Input your mutated sequence and load it
         mutated_protein1_id = 'N17P_SNP'
         mutated_protein1_seq = 'MSKQQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPR

         my_protein.load_manual_sequence(ident=mutated_protein1_id, seq=mutated_protein1_seq)
```

```
Out[12]: <SeqProp N17P_SNP at 0x7f8e304effd0>
```

```
In [13]: # Input another mutated sequence and load it
         mutated_protein2_id = 'Q4S_N17P_SNP'
         mutated_protein2_seq = 'MSKSQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPR

         my_protein.load_manual_sequence(ident=mutated_protein2_id, seq=mutated_protein2_seq)

Out[13]: <SeqProp Q4S_N17P_SNP at 0x7f8e3033d5c0>
```

Protein.**pairwise_align_sequences_to_representative**(*gapopen=10*, *gapextend=0.5*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*)

Align all sequences in the sequences attribute to the representative sequence. Stores the alignments the `sequence_alignments` DictList

Parameters

- **gapopen** –
- **gapextend** –
- **outdir** –
- **engine** –
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** –

Returns:

```
In [14]: # Conduct pairwise sequence alignments
         my_protein.pairwise_align_sequences_to_representative()

In [15]: # View the stored information for one of the alignments
         my_protein.representative_sequence.sequence_alignments
         my_protein.representative_sequence.sequence_alignments[0].annotations

         str(my_protein.representative_sequence.sequence_alignments[0][0].seq)
         str(my_protein.representative_sequence.sequence_alignments[0][1].seq)

Out[15]: [<<class 'Bio.Align.MultipleSeqAlignment'> instance (2 records of length 468, SingleLetterAl
          <<class 'Bio.Align.MultipleSeqAlignment'> instance (2 records of length 468, SingleLetterAl

Out[15]: {'a_seq': 'WT',
          'b_seq': 'N17P_SNP',
          'deletions': [],
          'insertions': [],
          'mutations': [('N', 17, 'P')],
          'percent_gaps': 0.0,
          'percent_identity': 99.8,
          'percent_similarity': 99.8,
          'score': 2381.0}

Out[15]: 'MSKQQIGVVGMAVMGRNLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAGAGTDAAIDSLKPYI

Out[15]: 'MSKQQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPRRILLMVKAGAGTDAAIDSLKPYI

In [16]: # Summarize all the mutations in all alignments
         s,f = my_protein.representative_sequence.sequence_mutation_summary()
         print('Single mutations:')
         s
         print('----------------------')
         print('Mutation fingerprints')
         f
```

```
Single mutations:
```

```
Out[16]: {('N', 17, 'P'): ['N17P_SNP', 'Q4S_N17P_SNP'], ('Q', 4, 'S'): ['Q4S_N17P_SNP']}
```

```
---------------------
Mutation fingerprints
```

```
Out[16]: {(('N', 17, 'P'),): ['N17P_SNP'],
          (('Q', 4, 'S'), ('N', 17, 'P')): ['Q4S_N17P_SNP']}
```

Some additional methods

Getting binding site/other information from UniProt

```
In [17]: import ssbio.databases.uniprot
```

```
In [18]: this_examples_uniprot = 'A0A0N2BFZ3'
         sites_df = ssbio.databases.uniprot.uniprot_sites(this_examples_uniprot)
         sites_df
```

```
Out[18]: type  seq_start  seq_end  \
         0               Domain        179      467
         1   Nucleotide binding         10       15
         2   Nucleotide binding         33       35
         3   Nucleotide binding         74       76
         4               Region        128      130
         5               Region        186      187
         6          Active site        183      183
         7          Active site        190      190
         8         Binding site        102      102
         9         Binding site        102      102
         10        Binding site        191      191
         11        Binding site        260      260
         12        Binding site        287      287
         13        Binding site        445      445
         14        Binding site        451      451

                                                notes
         0   Note=6PGD;Ontology_term=ECO:0000259;evidence=E...
         1   Note=NADP;Ontology_term=ECO:0000256;evidence=E...
         2   Note=NADP;Ontology_term=ECO:0000256;evidence=E...
         3   Note=NADP;Ontology_term=ECO:0000256;evidence=E...
         4   Note=Substrate binding;Ontology_term=ECO:00002...
         5   Note=Substrate binding;Ontology_term=ECO:00002...
         6   Note=Proton acceptor;Ontology_term=ECO:0000256...
         7   Note=Proton donor;Ontology_term=ECO:0000256;ev...
         8   Note=NADP;Ontology_term=ECO:0000256;evidence=E...
         9   Note=Substrate;Ontology_term=ECO:0000256;evide...
         10  Note=Substrate;Ontology_term=ECO:0000256;evide...
         11  Note=Substrate%3B via amide nitrogen;Ontology_...
         12  Note=Substrate;Ontology_term=ECO:0000256;evide...
         13  Note=Substrate%3B shared with dimeric partner;...
         14  Note=Substrate%3B shared with dimeric partner;...
```

```
In [19]: # Saving a list of the nucleotide binding site residues
         nucleotide_binding_sites = []
         for i, r in sites_df[sites_df.type=='Nucleotide binding'][['seq_start', 'seq_end']].iterrows
             start = r.seq_start
             end = r.seq_end
             for x in range(start, end+1):
```

```
                nucleotide_binding_sites.append(x)
        nucleotide_binding_sites
```

```
Out[19]: [10, 11, 12, 13, 14, 15, 33, 34, 35, 74, 75, 76]
```

Mapping sequence residue numbers to structure residue numbers

Methods

```
In [20]: # Returns a dictionary mapping sequence residue numbers to structure residue identifiers
         structure_sites = my_protein.representative_structure.map_repseq_resnums_to_structure_resnu
         structure_sites
```

```
Out[20]: {10: (' ', 10, ' '),
          11: (' ', 11, ' '),
          12: (' ', 12, ' '),
          13: (' ', 13, ' '),
          14: (' ', 14, ' '),
          15: (' ', 15, ' '),
          33: (' ', 33, ' '),
          34: (' ', 34, ' '),
          35: (' ', 35, ' '),
          74: (' ', 74, ' '),
          75: (' ', 75, ' '),
          76: (' ', 76, ' ')}
```

```
In [21]: # For viewing below, we can remap binding site residues to the structure (luckily they are
         nucleotide_binding_site_remapped_to_structure = [x[1] for x in structure_sites.values()]
         nucleotide_binding_site_remapped_to_structure
```

```
Out[21]: [33, 34, 35, 76, 74, 11, 12, 13, 14, 15, 75, 10]
```

```
In [22]: # Will warn you if residues are not present in the structure
         my_protein.representative_structure.map_repseq_resnums_to_structure_resnums([1,2,3])
```

```
[2017-03-09 15:29] [ssbio.structure.structprop] WARNING: 2zyd-A, 1: structure file does not contain
[2017-03-09 15:29] [ssbio.structure.structprop] WARNING: 2zyd-A, 2: structure file does not contain
```

```
Out[22]: {3: (' ', 3, ' ')}
```

Viewing structures

The awesome package nglview is utilized as a backend for viewing structures within a Jupyter notebook. There are many more options which can be set if you run:

```
import nglview
view = nglview.show_structure_file(my_protein.representative_structure.structure_path)
view
```

`ssbio` provides some wrapper functions to easily view structures and also map sequence residue numbers to structure residue numbers:

Methods

`StructProp.`**`view_structure`**(*opacity=1.0*, *recolor=True*, *gui=False*)

Use NGLviewer to display a structure in a Jupyter notebook

Parameters

- **opacity** (*float*) – Opacity of the structure
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

```
In [23]: # View just the structure
         my_protein.representative_structure.view_structure()
```

Protein.**view_all_mutations**(*grouped=False*, *color='red'*, *unique_colors=True*, *structure_opacity=0.5*, *opacity_range=(0.8, 1)*, *scale_range=(1, 5)*, *gui=False*)

Map all sequence alignment mutations to the structure.

Parameters

- **grouped** (*bool*) – If groups of mutations should be colored and sized together
- **color** (*str*) – Color of the mutations (overridden if unique_colors=True)
- **unique_colors** (*bool*) – If each mutation/mutation group should be colored uniquely
- **structure_opacity** (*float*) – Opacity of the protein structure cartoon representation
- **opacity_range** (*tuple*) – Min/max opacity values (mutations that show up more will be opaque)
- **scale_range** (*tuple*) – Min/max size values (mutations that show up more will be bigger)
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

```
In [24]: # Map the mutations on the visualization (scale increased)
         my_protein.view_all_mutations(scale_range=(4,7))
```

```
[2017-03-09 15:29] [ssbio.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and 17
[2017-03-09 15:29] [ssbio.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and 4
```

StructProp.**view_structure_and_highlight_residues**(*structure_resnums*, *chain=None*, *color='red'*, *structure_opacity=0.5*, *gui=False*)

Input a residue number or numbers to view on the structure.

Parameters

- **structure_resnums** (*int, list*) – Residue number(s) to highlight, structure numbering
- **chain** (*str, list*) – Chain ID or IDs of which residues are a part of. If not provided, all chains in the mapped_chains attribute will be used. IMPORTANT: if that is also empty, all residues in all chains matching the residue numbers will be shown, which may not always be correct.
- **color** (*str*) – Color to highlight with
- **structure_opacity** (*float*) – Opacity of the protein structure cartoon representation
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

```
In [25]: # View just the structure with selected residues
         my_protein.representative_structure.view_structure_and_highlight_residues(structure_resnums=
```

```
[2017-03-09 15:29] [ssbio.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and ( 1 or 2
```

```
In [26]: # View the previously saved binding site
         my_protein.representative_structure.view_structure_and_highlight_residues(structure_resnums=
```

```
[2017-03-09 15:29] [ssbio.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and ( 33 or
```

Saving

`Protein.`**`save_json`**(*outfile*, *compression=True*)
Save the object as a JSON file using json_tricks

```
In [27]: import os.path as op
         my_protein.save_json(op.join(my_protein.protein_dir, '{}.json'.format(my_protein.id)), compr
```

```
[2017-03-09 15:29] [ssbio.core.io] INFO: Saved <class 'ssbio.core.protein.Protein'> (id: SRR1753782_(
```

The GEM-PRO Pipeline

Introduction

The GEM-PRO pipeline is focused on annotating genome-scale models with protein structure information. Any SBML model can be used as input to the pipeline, although it is not required to have a one. Here are the possible starting points for using the pipeline:

- An SBML model in *SBML* (*.sbml*, *.xml*), or *MATLAB* (*.mat*) formats
- A list of gene IDs (*['b0001', 'b0002', ...]*)
- A dictionary of gene IDs and their sequences (*{'b0001':'MSAVEVEEAP..', 'b0002':'AERAPLS', ...}*)

Tutorials

GEM-PRO - Calculating Protein Properties

This notebook gives an example of how to **calculate protein properties** for a list of proteins, by first pulling information from UniProt and then using the 3D structure files to calculate the desired properties

---

Input: List of gene IDs

Output: Representative protein structures and properties associated with them

---

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #

In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- `ROOT_DIR`
  - The directory where a folder named after your `PROJECT` will be created
- `PROJECT`
  - Your project name
- `LIST_OF_GENES`
  - Your list of gene IDs

A directory will be created in `ROOT_DIR` with your `PROJECT` name. The folders are organized like so:

```
ROOT_DIR
- PROJECT
    - data  # General storage for pipeline outputs
    - model  # SBML and GEM-PRO models are stored here
    - genes  # Per gene information
    |   - <gene_id1>  # Specific gene directory
    |   |   - protein
    |   |       - sequences  # Protein sequence files, alignments, etc.
    |   |       - structures  # Protein structure files, calculations, etc.
    |   - <gene_id2>
    |       - protein
    |           - sequences
    |           - structures
    - reactions  # Per reaction information
    |   - <reaction_id1>  # Specific reaction directory
    |       - complex
    |           - structures  # Protein complex files
    - metabolites  # Per metabolite information
        - <metabolite_id1>  # Specific metabolite directory
            - chemical
                - structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        ROOT_DIR = '/home/nathan/projects_unsynced'
        PROJECT = 'ssbio_protein_properties'
        LIST_OF_GENES = ['b1276', 'b0118']
```

```
In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_list=LIST_OF_GENES, pdb_file_ty
```

```
[2017-05-08 13:57] [ssbio.pipeline.gempro] INFO: /home/nathan/projects_unsynced/ssbio_protein_propert
[2017-05-08 13:57] [ssbio.pipeline.gempro] INFO: 2: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping (and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

Methods

```
In [8]: # UniProt mapping
        my_gempro.uniprot_mapping_and_metadata(model_gene_source='ENSEMBLGENOME_ID')
        print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
        my_gempro.df_uniprot_metadata.head()
```

```
[2017-05-08 13:57] [root] INFO: getUserAgent: Begin
[2017-05-08 13:57] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
```

```
[2017-05-08 13:57] [root] INFO: getUserAgent: End
[2017-05-08 13:57] [ssbio.pipeline.gempro] INFO: 2/2: number of genes mapped to UniProt
[2017-05-08 13:57] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro

Missing UniProt mapping:  []
```

Out[8]:

gene	uniprot	reviewed	num_pdbs	seq_len	description	sequence_file	metadata_file
b0118	P36683	False	0	865	Aconitate hydratase B	P36683.fasta	P36683.xml
b1276	P25516	False	0	891	Aconitate hydratase A	P25516.fasta	P25516.xml

```
In [9]: # Set representative sequences
        my_gempro.set_representative_sequence()
        print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
        my_gempro.df_representative_sequences.head()

[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative sequence
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for

Missing a representative sequence:  []
```

Out[9]:

gene	uniprot	num_pdbs	seq_len	sequence_file	metadata_file
b0118	P36683	0	865	P36683.fasta	P36683.xml
b1276	P25516	0	891	P25516.fasta	P25516.xml

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

Methods

```
In [10]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()

[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-05-08 13:58] [root] INFO: getUserAgent: Begin
[2017-05-08 13:58] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-05-08 13:58] [root] INFO: getUserAgent: End
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: 1/2: number of genes with at least one experimental
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

```
Out[10]: pdb_id pdb_chain_id uniprot experimental_method  resolution  coverage  \
         gene
         b0118   1l5j            A  P36683   X-ray diffraction         2.4         1
         b0118   1l5j            B  P36683   X-ray diffraction         2.4         1

                start  end  unp_start  unp_end  rank
         gene
         b0118      1  865          1      865     1
         b0118      1  865          1      865     2
```

```
In [11]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: 0: number of genes with additional structures added
[2017-05-08 13:58] [ssbio.pipeline.gempro] WARNING: Empty dataframe
```

```
Out[11]: Empty DataFrame
         Columns: []
         Index: []
```

```
In [12]: import pandas as pd
         import os.path as op
```

```
In [13]: # Creating manual mapping dictionary for ECOLI I-TASSER models
         homology_models = '/home/nathan/projects_archive/homology_models/ECOLI/zhang/'
         homology_models_df = pd.read_csv('/home/nathan/projects_archive/homology_models/ECOLI/zhang_
         tmp = homology_models_df[['zhang_id','model_file','m_gene']].drop_duplicates()
         tmp = tmp[pd.notnull(tmp.m_gene)]

         homology_model_dict = {}

         for i,r in tmp.iterrows():
             homology_model_dict[r['m_gene']] = {r['zhang_id']: {'model_file':op.join(homology_models
                                                                 'file_type':'pdb'}}

         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Updated homology model information for 2 genes.
```

```
In [14]: # Creating manual mapping dictionary for ECOLI SUNPRO models
         homology_models = '/home/nathan/projects_archive/homology_models/ECOLI/sunpro/'
         homology_models_df = pd.read_csv('/home/nathan/projects_archive/homology_models/ECOLI/sunpro
         tmp = homology_models_df[['sunpro_id','model_file','m_gene']].drop_duplicates()
         tmp = tmp[pd.notnull(tmp.m_gene)]

         homology_model_dict = {}

         for i,r in tmp.iterrows():
             homology_model_dict[r['m_gene']] = {r['sunpro_id']: {'model_file':op.join(homology_model
                                                                 'file_type':'pdb'}}

         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Updated homology model information for 2 genes.
```

```
In [15]: # Loading a manually created homology model
         homology_model_dict = {}
         homology_model_dict['b1276'] = {'b1276_ITASSER': {'model_file':'/home/nathan/Desktop/S32808
                                                           'file_type':'pdb'}}
         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Updated homology model information for 1 genes.
```

Downloading and ranking structures

Methods

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [16]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: Saved 1 structures total
```

```
Out[16]: pdb_id                                pdb_title  \
         gene
         b0118   1l5j  CRYSTAL STRUCTURE OF E. COLI ACONITASE B.

                                      description experimental_method mapped_chains  \
         gene
         b0118  Aconitate hydratase 2 (E.C.4.2.1.3)   X-ray diffraction           A;B

                resolution chemicals                                date  \
         gene
         b0118         2.4   TRA;F3S  2002-06-12;2003-04-01;2009-02-24

                   taxonomy_name structure_file
         gene
         b0118  Escherichia coli       1l5j.pdb
```

```
In [17]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[17]: id is_experimental                       structure_file
         gene
         b0118         1l5j-A            True                       1l5j-A_clean.pdb
         b1276  ACON1_ECOLI-X           False  ACON1_ECOLI_model1_clean-X_clean.pdb
```

Computing sequence and structure properties

```
In [18]: # Requires EMBOSS "pepstats" program
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         # Install using:
         # sudo apt-get install emboss
         my_gempro.get_sequence_properties()
```

```
In [19]: # Requires SCRATCH installation, replace path_to_scratch with own path to script
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_scratch_predictions(path_to_scratch='/home/nathan/software/SCRATCH-1D_1.1/bin/
```

```
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: /home/nathan/projects_unsynced/ssbio_protein_propert
[2017-05-08 13:58] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with SCRATCH predictions loaded
```

```
In [20]: my_gempro.get_disulfide_bridges()
```

```
In [21]: # Requires DSSP installation
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_dssp_annotations()
```

```
In [22]: # Requires MSMS installation
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_msms_annotations()
```

Extracting residue-level properties for a list of residue numbers

```
In [23]: from Bio.SeqFeature import SeqFeature, FeatureLocation
```

Looking at one protein

```
In [24]: my_protein = my_gempro.genes.b0118.protein
```

```
In [25]: # Here are the features stored for the sequence, parsed from UniProt
         my_protein.representative_sequence.seq_record.features
```

```
Out[25]: [SeqFeature(FeatureLocation(ExactPosition(0), ExactPosition(865)), type='chain', id='PRO_000
          SeqFeature(FeatureLocation(ExactPosition(243), ExactPosition(246)), type='region of interes
          SeqFeature(FeatureLocation(ExactPosition(413), ExactPosition(416)), type='region of interes
          SeqFeature(FeatureLocation(ExactPosition(709), ExactPosition(710)), type='metal ion-binding
          SeqFeature(FeatureLocation(ExactPosition(768), ExactPosition(769)), type='metal ion-binding
          SeqFeature(FeatureLocation(ExactPosition(771), ExactPosition(772)), type='metal ion-binding
          SeqFeature(FeatureLocation(ExactPosition(190), ExactPosition(191)), type='binding site'),
          SeqFeature(FeatureLocation(ExactPosition(497), ExactPosition(498)), type='binding site'),
          SeqFeature(FeatureLocation(ExactPosition(790), ExactPosition(791)), type='binding site'),
          SeqFeature(FeatureLocation(ExactPosition(795), ExactPosition(796)), type='binding site'),
          SeqFeature(FeatureLocation(ExactPosition(768), ExactPosition(769)), type='mutagenesis site
          SeqFeature(FeatureLocation(ExactPosition(1), ExactPosition(14)), type='helix'),
          SeqFeature(FeatureLocation(ExactPosition(23), ExactPosition(35)), type='helix'),
          SeqFeature(FeatureLocation(ExactPosition(41), ExactPosition(51)), type='helix'),
          SeqFeature(FeatureLocation(ExactPosition(58), ExactPosition(72)), type='helix'),
```

```
SeqFeature(FeatureLocation(ExactPosition(82), ExactPosition(90)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(98), ExactPosition(104)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(104), ExactPosition(107)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(108), ExactPosition(111)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(111), ExactPosition(120)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(127), ExactPosition(137)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(140), ExactPosition(151)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(153), ExactPosition(157)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(165), ExactPosition(178)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(178), ExactPosition(182)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(184), ExactPosition(190)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(193), ExactPosition(197)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(197), ExactPosition(200)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(213), ExactPosition(216)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(219), ExactPosition(227)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(232), ExactPosition(243)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(247), ExactPosition(257)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(257), ExactPosition(261)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(263), ExactPosition(269)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(271), ExactPosition(278)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(279), ExactPosition(288)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(291), ExactPosition(295)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(305), ExactPosition(310)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(310), ExactPosition(314)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(314), ExactPosition(318)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(318), ExactPosition(321)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(323), ExactPosition(327)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(333), ExactPosition(341)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(343), ExactPosition(360)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(383), ExactPosition(391)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(391), ExactPosition(394)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(408), ExactPosition(413)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(414), ExactPosition(417)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(417), ExactPosition(427)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(437), ExactPosition(440)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(443), ExactPosition(448)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(450), ExactPosition(465)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(465), ExactPosition(468)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(478), ExactPosition(483)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(483), ExactPosition(486)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(491), ExactPosition(497)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(502), ExactPosition(507)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(511), ExactPosition(521)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(530), ExactPosition(538)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(545), ExactPosition(559)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(572), ExactPosition(576)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(576), ExactPosition(583)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(588), ExactPosition(597)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(597), ExactPosition(600)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(600), ExactPosition(603)), type='turn'),
SeqFeature(FeatureLocation(ExactPosition(604), ExactPosition(609)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(612), ExactPosition(632)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(637), ExactPosition(653)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(666), ExactPosition(673)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(673), ExactPosition(676)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(680), ExactPosition(683)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(690), ExactPosition(693)), type='strand'),
SeqFeature(FeatureLocation(ExactPosition(693), ExactPosition(696)), type='helix'),
SeqFeature(FeatureLocation(ExactPosition(696), ExactPosition(699)), type='turn'),
```

```
 SeqFeature(FeatureLocation(ExactPosition(703), ExactPosition(707)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(713), ExactPosition(726)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(731), ExactPosition(737)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(741), ExactPosition(750)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(752), ExactPosition(760)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(769), ExactPosition(772)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(774), ExactPosition(777)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(783), ExactPosition(790)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(795), ExactPosition(798)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(801), ExactPosition(805)), type='strand'),
 SeqFeature(FeatureLocation(ExactPosition(807), ExactPosition(817)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(822), ExactPosition(834)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(836), ExactPosition(840)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(845), ExactPosition(848)), type='helix'),
 SeqFeature(FeatureLocation(ExactPosition(849), ExactPosition(857)), type='helix')]
```

In [26]:
```
# Gathering properties for the metal binding sites

metal_info = []

for g in my_gempro.genes:


    p = g.protein
    print('********GENE: {}********'.format(g.id))


    # Saving the structure residue numbering for visualization
    metal_binding_structure_residues = []

    for f in p.representative_sequence.seq_record.features:
        if 'metal' in f.type.lower():
            print(f)
            gene_info = {}
            gene_info['gene'] = g.id
            gene_info['structure_id'] = g.protein.representative_structure.id
            gene_info['feature'] = f.type

            # Get sequence properties
            sr = f.extract(p.representative_sequence.seq_record)
            print('**Residue-level properties calculated or predicted from sequence:')
            print(sr.seq)
            print(sr.letter_annotations)

            gene_info['seq_resnum'] = f.location.end.position
            gene_info['seq_residue'] = sr.seq
            gene_info.update(sr.letter_annotations)

            # Get structure properties
            mapping_to_structure_index = p.map_seqprop_resnums_to_structprop_chain_index(res
                                                                                          use
            new_f = SeqFeature(FeatureLocation(mapping_to_structure_index[f.location.end.pos
                                               mapping_to_structure_index[f.location.end.pos
            sr_st = new_f.extract(p.representative_structure.representative_chain.seq_record
            print('**Residue-level properties calculated from structure:')
            print(sr_st.seq)
            print(sr_st.letter_annotations)
            print('--------------------------------')
```

```
                    # Get structure residue numbers
                    mapping_to_structure_resnum = p.map_seqprop_resnums_to_structprop_resnums(resnum
                                                                                               use_re
                    gene_info['struct_resnum'] = mapping_to_structure_resnum[f.location.end.position
                    gene_info['struct_residue'] = sr_st.seq
                    gene_info.update(sr_st.letter_annotations)

                    metal_info.append(gene_info)
            print('********************************')
            print()
```

```
********GENE: b1276********
type: metal ion-binding site
location: [434:435]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P25516_ACON1_ECOLI-X_chain_index': [435], 'RSA-accpro': '-', 'SS-sspro8': 'H', 'RSA-accpro20': [10],
**Residue-level properties calculated from structure:
S
'SS-dssp': ['-'], 'PHI-dssp': [93.200000000000003], 'structure_resnums': [(' ', 434, ' ')], 'ASA-dssp
--------------------------------
type: metal ion-binding site
location: [500:501]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P25516_ACON1_ECOLI-X_chain_index': [501], 'RSA-accpro': '-', 'SS-sspro8': 'C', 'RSA-accpro20': [0],
**Residue-level properties calculated from structure:
G
'SS-dssp': ['S'], 'PHI-dssp': [-171.30000000000001], 'structure_resnums': [(' ', 500, ' ')], 'ASA-dss
--------------------------------
type: metal ion-binding site
location: [503:504]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P25516_ACON1_ECOLI-X_chain_index': [504], 'RSA-accpro': '-', 'SS-sspro8': 'G', 'RSA-accpro20': [0],
**Residue-level properties calculated from structure:
T
'SS-dssp': ['G'], 'PHI-dssp': [-64.700000000000003], 'structure_resnums': [(' ', 503, ' ')], 'ASA-dss
--------------------------------
********************************

********GENE: b0118********
type: metal ion-binding site
location: [709:710]
```

```
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P36683_1l5j-A_chain_index': [710.0], 'SS-sspro': 'C', 'P36683_1l5j-B_chain_index': [710.0], 'RSA-acc
**Residue-level properties calculated from structure:
S
'SS-dssp': ['S'], 'PHI-dssp': [-109.59999999999999], 'structure_resnums': [(' ', 709, ' ')], 'ASA-dss
--------------------------------
type: metal ion-binding site
location: [768:769]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P36683_1l5j-A_chain_index': [769.0], 'SS-sspro': 'C', 'P36683_1l5j-B_chain_index': [769.0], 'RSA-acc
**Residue-level properties calculated from structure:
G
'SS-dssp': ['S'], 'PHI-dssp': [158.5], 'structure_resnums': [(' ', 768, ' ')], 'ASA-dssp': [4.0], 'PS
--------------------------------
type: metal ion-binding site
location: [771:772]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

**Residue-level properties calculated or predicted from sequence:
C
'P36683_1l5j-A_chain_index': [772.0], 'SS-sspro': 'H', 'P36683_1l5j-B_chain_index': [772.0], 'RSA-acc
**Residue-level properties calculated from structure:
L
'SS-dssp': ['G'], 'PHI-dssp': [-52.600000000000001], 'structure_resnums': [(' ', 771, ' ')], 'ASA-dss
--------------------------------
********************************
```

```
In [27]: cols = ['gene', 'structure_id', 'feature', 'seq_residue', 'seq_resnum',
                 'struct_residue','struct_resnum', 'SS-sspro','SS-sspro8','RSA-accpro','RSA-accpro20
                 'PHI-dssp', 'PSI-dssp', 'CA_DEPTH-msms', 'RES_DEPTH-msms', 'structure_resnums']

In [28]: pd.DataFrame.from_records(metal_info, columns=cols)
```

Out[28]:

	gene	structure_id	feature	seq_residue	seq_resnum	\
0	b1276	ACON1_ECOLI-X	metal ion-binding site	(C)	435	
1	b1276	ACON1_ECOLI-X	metal ion-binding site	(C)	501	
2	b1276	ACON1_ECOLI-X	metal ion-binding site	(C)	504	
3	b0118	1l5j-A	metal ion-binding site	(C)	710	
4	b0118	1l5j-A	metal ion-binding site	(C)	769	
5	b0118	1l5j-A	metal ion-binding site	(C)	772	

	struct_residue	struct_resnum	SS-sspro	SS-sspro8	RSA-accpro	RSA-accpro20	\
0	(S)	(, 435,)	H	H	-	[10]	
1	(G)	(, 501,)	C	C	-	[0]	

```
2            (T)    ( , 504,  )        H          G          -          [0]
3            (S)    ( , 710,  )        C          T          -         [10]
4            (G)    ( , 769,  )        C          C          -          [5]
5            (L)    ( , 772,  )        H          G          -          [5]

  SS-dssp            RSA-dssp ASA-dssp  PHI-dssp  PSI-dssp    CA_DEPTH-msms  \
0     [-]   [0.0538461538462]    [7.0]    [93.2]   [133.0]  [1.99980235487]
1     [S]   [0.0357142857143]    [3.0]  [-171.3]  [-178.9]  [4.87276526243]
2     [G]   [0.0211267605634]    [3.0]   [-64.7]   [-34.6]   [3.1546567154]
3     [S]   [0.0153846153846]    [2.0]  [-109.6]  [-172.4]  [10.0241471689]
4     [S]    [0.047619047619]    [4.0]   [158.5]  [-175.6]  [6.68098052599]
5     [G]  [0.00609756097561]    [1.0]   [-52.6]   [-24.8]  [6.80593314578]

    RES_DEPTH-msms structure_resnums
0   [2.4908282495]     [( , 434,  )]
1  [4.50809912497]     [( , 500,  )]
2  [3.01774183799]     [( , 503,  )]
3  [9.79279425043]     [( , 709,  )]
4  [6.49714241709]     [( , 768,  )]
5  [6.33928360614]     [( , 771,  )]
```

Visualizing residues

```
In [31]: metal_binding_structure_residues = [710, 769, 772]
```

```
In [29]: my_protein.representative_structure.view_structure_and_highlight_residues(structure_resnums=
```

```
[2017-06-01 09:38] [ssbio.protein.structure.structprop] INFO: Selection: ( :A ) and not hydrogen and
```

Looking at residue depths in the same protein

```
In [34]: for xx in g.protein.representative_sequence.structure_alignments:
             print(xx.id)
```

```
P36683_1l5j-A
P36683_1l5j-B
P36683_ACON2_ECOLI-X
P36683_E00113-X
```

```
In [35]: # Run MSMS for all structures
         for g in my_gempro.genes:
             print(g)
             for x in g.protein.structures:
                 print(x)
                 x.get_residue_depths(outdir=g.protein.structure_dir)
                 x.align_seqprop_to_mapped_chains(g.protein.representative_sequence)
                 x.map_seqprop_resnums_to_mapped_chains(g.protein.representative_sequence, [1,2,3])
```

```
b1276
ACON1_ECOLI
```

```
Out[35]: {'X': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
E01201
```

```
Out[35]: {'X': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
b1276_ITASSER
```

```
Out[35]: {'A': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
b0118
115j
```

```
Out[35]: {'A': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')},
          'B': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
ACON2_ECOLI
```

```
Out[35]: {'X': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
E00113
```

```
Out[35]: {'X': {1: (' ', 1, ' '), 2: (' ', 2, ' '), 3: (' ', 3, ' ')}}
```

```
In [40]: from Bio.SeqFeature import SeqFeature, FeatureLocation
```

```
In [50]: # Gathering properties for the metal binding sites

         metal_info = []

         for g in my_gempro.genes:
             print(g)
             p = g.protein
             for f in p.representative_sequence.seq_record.features:
                 if 'metal' in f.type.lower():
                     print(f)
                     # Get sequence properties
                     sr = f.extract(p.representative_sequence.seq_record)

                     for s in p.structures:
                         print(s)
                         # Get structure properties
                         new_f = SeqFeature(FeatureLocation(int(sr.letter_annotations['repchain_resnu
                         sr_st = new_f.extract(p.representative_structure.representative_chain.seq_re

                         print(sr_st.letter_annotations['RES_DEPTH-msms'])
             print()
```

```
b1276
type: metal ion-binding site
location: [434:435]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

ACON1_ECOLI
[2.8135356659097708]
E01201
[2.8135356659097708]
b1276_ITASSER
[2.8135356659097708]
type: metal ion-binding site
location: [500:501]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

ACON1_ECOLI
[2.4091192574526867]
E01201
```

```
[2.4091192574526867]
b1276_ITASSER
[2.4091192574526867]
type: metal ion-binding site
location: [503:504]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 1
    Key: type, Value: metal ion-binding site

ACON1_ECOLI
[1.961483950461508]
E01201
[1.961483950461508]
b1276_ITASSER
[1.961483950461508]

b0118
type: metal ion-binding site
location: [709:710]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

1l5j
[10.009108769044623]
ACON2_ECOLI
[10.009108769044623]
E00113
[10.009108769044623]
type: metal ion-binding site
location: [768:769]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

1l5j
[8.0498321032343032]
ACON2_ECOLI
[8.0498321032343032]
E00113
[8.0498321032343032]
type: metal ion-binding site
location: [771:772]
qualifiers:
    Key: description, Value: Iron-sulfur (4Fe-4S)
    Key: evidence, Value: 5
    Key: type, Value: metal ion-binding site

1l5j
[8.2393688432309204]
ACON2_ECOLI
[8.2393688432309204]
E00113
[8.2393688432309204]
```

```
In [ ]:
```

GEM-PRO - Genes & Sequences

This notebook gives an example of how to run the GEM-PRO pipeline with a **dictionary of gene IDs and their protein sequences**.

Input: Dictionary of gene IDs and protein sequences

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #
```

```
In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- ROOT_DIR
  - The directory where a folder named after your PROJECT will be created
- PROJECT
  - Your project name
- LIST_OF_GENES
  - Your list of gene IDs

A directory will be created in ROOT_DIR with your PROJECT name. The folders are organized like so:

```
ROOT_DIR
- PROJECT
    - data  # General storage for pipeline outputs
    - model  # SBML and GEM-PRO models are stored here
    - genes  # Per gene information
    |   - <gene_id1>  # Specific gene directory
    |   |   - protein
    |   |       - sequences  # Protein sequence files, alignments, etc.
    |   |       - structures  # Protein structure files, calculations, etc.
    |   - <gene_id2>
    |       - protein
    |           - sequences
    |           - structures
    - reactions  # Per reaction information
    |   - <reaction_id1>  # Specific reaction directory
    |       - complex
    |           - structures  # Protein complex files
    - metabolites  # Per metabolite information
        - <metabolite_id1>  # Specific metabolite directory
            - chemical
                - structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROJECT = 'genes_and_sequences_GP'
        GENES_AND_SEQUENCES = {'b0870': 'MIDLRSDTVTRPSRAMLEAMMAAPVGDDVYGDDPTVNALQDYAAELSGKEAAIFLPTGT
                               'b3041': 'MNQTLLSSFGTPFERVENALAALREGRGVMVLDDEDRENEGDMIFPAETMTVEQMALTI

In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_and_sequences=GENES_AND_SEQUENC
```

```
[2017-03-14 14:29] [ssbio.pipeline.gempro] INFO: /tmp/genes_and_sequences_GP: GEM-PRO project locatio
[2017-03-14 14:29] [ssbio.pipeline.gempro] INFO: Loaded in 2 sequences
[2017-03-14 14:29] [ssbio.pipeline.gempro] INFO: 2: number of genes
```

Mapping sequence –> structure

Since the sequences have been provided, we just need to BLAST them to the PDB.

Note: These methods do not download any 3D structure files.

Methods

```
In [8]: # Mapping using BLAST
        my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
        my_gempro.df_pdb_blast.head(2)
```

```
[2017-03-14 14:29] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-03-14 14:29] [ssbio.pipeline.gempro] INFO: 2: number of genes with additional structures added
```

```
Out[8]: pdb_id pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
        gene
        b0870   3wlx            A     1713.0         0.0                 1.0
        b0870   3wlx            B     1713.0         0.0                 1.0

               hit_percent_ident  hit_num_ident  hit_num_similar
        gene
        b0870                1.0            333              333
        b0870                1.0            333              333
```

Downloading and ranking structures

Methods

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [10]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
[2017-03-14 14:31] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-03-14 14:31] [ssbio.pipeline.gempro] INFO: Saved 11 structures total
```

```
Out[10]: pdb_id                                          pdb_title  \
         gene
         b0870   3wlx  Crystal structure of low-specificity L-threoni...
         b0870   4rjy  Crystal structure of E. coli L-Threonine Aldol...

                                                 description experimental_method  \
```

```
gene
b0870  Low specificity L-threonine aldolase (E.C.4.1....   X-RAY DIFFRACTION
b0870               Low specificity L-threonine aldolase   X-RAY DIFFRACTION

      mapped_chains  resolution chemicals                              date  \
gene
b0870           A;B        2.51       PLG                         2014-12-17
b0870       A;B;C;D        2.10        NA  2014-10-29;2015-01-21;2015-02-04

          taxonomy_name structure_file
gene
b0870  Escherichia coli       3wlx.cif
b0870  Escherichia coli       4rjy.cif
```

```
In [10]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
[2017-03-08 13:03] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-03-08 13:03] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[10]: id is_experimental reference_seq  reference_seq_top_coverage
         gene
         b3041  1iez-A            True         b3041                        100.0
         b0870  3wlx-A            True         b0870                         99.4
```

```
In [11]: # Looking at the information saved within a gene
         my_gempro.genes.get_by_id('b0870').protein.representative_structure
         my_gempro.genes.get_by_id('b0870').protein.representative_structure.get_dict()
```

```
Out[11]: <StructProp 3wlx-A at 0x7f34cc158390>
```

```
Out[11]: {'_structure_dir': '/tmp/genes_and_sequences_GP/genes/b0870/b0870_protein/structures',
          'chains': [<ChainProp A at 0x7f34cc571358>],
          'date': '2014-12-17',
          'description': 'Low specificity L-threonine aldolase (E.C.4.1.2.48)',
          'file_type': 'pdb',
          'id': '3wlx-A',
          'is_experimental': True,
          'mapped_chains': ['A'],
          'original_pdb_id': '3wlx',
          'reference_seq': <SeqProp b0870 at 0x7f34cc572d68>,
          'reference_seq_top_coverage': 99.4,
          'representative_chain': <ChainProp A at 0x7f34cc571400>,
          'resolution': 2.51,
          'structure_file': '3wlx-A_clean.pdb',
          'taxonomy_name': 'Escherichia coli'}
```

Creating homology models

For those proteins with no representative structure, we can create homology models for them. `ssbio` contains some built in functions for easily running I-TASSER locally or on machines with `SLURM` (ie. on NERSC) or `Torque` job scheduling.

You can load in I-TASSER models once they complete using the `get_itasser_models` later.

Info: Homology modeling can take a long time - about 24-72 hours per protein (highly dependent on the sequence

length, as well as if there are available templates).

Methods

```
In [12]: # Prep I-TASSER model folders
         my_gempro.prep_itasser_models('~/software/I-TASSER4.4', '~/software/ITLIB/', runtype='local
```

```
[2017-03-08 13:03] [ssbio.pipeline.gempro] INFO: Prepared I-TASSER modeling folders for 0 genes in fo
```

Saving your GEM-PRO

Warning: Saving is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

```
In [13]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-03-08 13:03] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-03-08 13:03] [ssbio.core.io] INFO: Saved <class 'ssbio.pipeline.gempro.GEMPRO'> (id: genes_and_
```

GEM-PRO - List of Gene IDs

This notebook gives an example of how to run the GEM-PRO pipeline with a **list of gene IDs**.

Input: List of gene IDs

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown

- ERROR
  - Major errors
- WARNING
  - Warnings that don't affect running of the pipeline
- INFO (default)
  - Info such as the number of structures mapped per gene
- DEBUG
  - Really detailed information that will print out a lot of stuff

Warning: DEBUG mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #

In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- ROOT_DIR
  - The directory where a folder named after your PROJECT will be created
- PROJECT
  - Your project name
- LIST_OF_GENES
  - Your list of gene IDs

A directory will be created in ROOT_DIR with your PROJECT name. The folders are organized like so:

```
ROOT_DIR
- PROJECT
    - data  # General storage for pipeline outputs
    - model  # SBML and GEM-PRO models are stored here
    - genes  # Per gene information
    |   - <gene_id1>  # Specific gene directory
    |   |   - protein
    |   |       - sequences  # Protein sequence files, alignments, etc.
    |   |       - structures  # Protein structure files, calculations, etc.
    |   - <gene_id2>
    |       - protein
    |           - sequences
    |           - structures
    - reactions  # Per reaction information
```

```
    |   - <reaction_id1>  # Specific reaction directory
    |       - complex
    |           - structures  # Protein complex files
    - metabolites  # Per metabolite information
        - <metabolite_id1>  # Specific metabolite directory
            - chemical
                - structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        PROJECT = 'genes_GP'
        LIST_OF_GENES = ['b0761', 'b0889', 'b0995', 'b1013', 'b1014', 'b1040', 'b1130', 'b1187', 'b12
```

```
In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, genes_list=LIST_OF_GENES)
```

```
[2017-03-16 16:19] [ssbio.pipeline.gempro] INFO: /tmp/genes_GP: GEM-PRO project location
[2017-03-16 16:19] [ssbio.pipeline.gempro] INFO: 10: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping (and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

Methods

```
In [8]: # KEGG mapping of gene ids
        my_gempro.kegg_mapping_and_metadata(kegg_organism_code='eco')
        print('Missing KEGG mapping: ', my_gempro.missing_kegg_mapping)
        my_gempro.df_kegg_metadata.head()
```

```
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to KEGG
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> KEGG. See the "df_kegg_meta
```

```
Missing KEGG mapping:  []
```

```
Out[8]: kegg     refseq uniprot  num_pdbs                    pdbs  \
        gene
        b1130  eco:b1130  NP_415648  P23836         2                2PL1;2PKX
        b0995  eco:b0995  NP_415515  P38684         1                     1ZGZ
        b1040  eco:b1040  NP_415558  P52106         0                      NaN
        b0761  eco:b0761  NP_415282  P0A9G8         5  1B9M;1H9S;1B9N;1O7L;1H9R
        b0889  eco:b0889  NP_415409  P0ACJ0         2                2GQQ;2L4A

               seq_len  sequence_file   metadata_file
```

```
        gene
        b1130      223  eco-b1130.faa  eco-b1130.kegg
        b0995      230  eco-b0995.faa  eco-b0995.kegg
        b1040      216  eco-b1040.faa  eco-b1040.kegg
        b0761      262  eco-b0761.faa  eco-b0761.kegg
        b0889      164  eco-b0889.faa  eco-b0889.kegg
```

In [9]:
```
        # UniProt mapping
        my_gempro.uniprot_mapping_and_metadata(model_gene_source='ENSEMBLGENOME_ID')
        print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
        my_gempro.df_uniprot_metadata.head()
```

```
[2017-03-16 16:18] [root] INFO: getUserAgent: Begin
[2017-03-16 16:18] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-03-16 16:18] [root] INFO: getUserAgent: End
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to UniProt
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro


Missing UniProt mapping:  []
```

Out[9]:
```
        uniprot reviewed gene_name                 kegg  \
        gene
        b1130  P23836     True      phoP  ecj:JW1116;eco:b1130
        b0995  P38684     True      torR  ecj:JW0980;eco:b0995
        b1040  P52106     True      csgD  ecj:JW1023;eco:b1040
        b0761  P0A9G8     True      modE  ecj:JW0744;eco:b0761
        b0889  P0ACJ0     True       lrp  ecj:JW0872;eco:b0889

                                                    refseq  num_pdbs  \
        gene
        b1130  NP_415648.1;NC_000913.3;WP_001265471.1;NZ_LN83...         2
        b0995  NP_415515.1;NC_000913.3;WP_001120125.1;NZ_LN83...         1
        b1040  NP_415558.1;NC_000913.3;WP_000481509.1;NZ_LN83...         0
        b0761  NP_415282.1;NC_000913.3;WP_001147439.1;NZ_LN83...         5
        b0889  NP_415409.1;NC_000913.3;WP_000228473.1;NZ_LN83...         2

                                pdbs ec_number             pfam  seq_len  \
        gene
        b1130                2PKX;2PL1       NaN  PF00072;PF00486      223
        b0995                     1ZGZ       NaN  PF00072;PF00486      230
        b1040                      NaN       NaN          PF00196      216
        b0761  1B9M;1H9S;1H9R;1B9N;1O7L       NaN  PF00126;PF03459      262
        b0889                2GQQ;2L4A       NaN          PF01037      164

                                                 description entry_version  \
        gene
        b1130            Transcriptional regulatory protein PhoP    2017-03-15
        b0995  TorCAD operon transcriptional regulatory prote...    2017-03-15
        b1040   CsgBAC operon transcriptional regulatory protein    2017-02-15
        b0761                     Transcriptional regulator ModE    2017-03-15
        b0889              Leucine-responsive regulatory protein    2017-03-15

              seq_version sequence_file metadata_file
        gene
        b1130  1991-11-01  P23836.fasta    P23836.txt
        b0995  1997-11-01  P38684.fasta    P38684.txt
        b1040  1996-10-01  P52106.fasta    P52106.txt
        b0761  2005-07-19  P0A9G8.fasta    P0A9G8.txt
        b0889  2007-01-23  P0ACJ0.fasta    P0ACJ0.txt
```

```
In [10]: # Set representative sequences
         my_gempro.set_representative_sequence()
         print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
         my_gempro.df_representative_sequences.head()
```

```
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: 10/10: number of genes with a representative sequenc
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for
```

```
Missing a representative sequence:  []
```

```
Out[10]: uniprot                  kegg  num_pdbs                       pdbs  \
         gene
         b1130  P23836  ecj:JW1116;eco:b1130         2                  2PKX;2PL1
         b0995  P38684  ecj:JW0980;eco:b0995         1                       1ZGZ
         b1040  P52106  ecj:JW1023;eco:b1040         0                        NaN
         b0761  P0A9G8  ecj:JW0744;eco:b0761         5  1B9M;1H9S;1H9R;1B9N;1O7L
         b0889  P0ACJ0  ecj:JW0872;eco:b0889         2                  2GQQ;2L4A

                seq_len sequence_file
         gene
         b1130      223  P23836.fasta
         b0995      230  P38684.fasta
         b1040      216  P52106.fasta
         b0761      262  P0A9G8.fasta
         b0889      164  P0ACJ0.fasta
```

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

Methods

```
In [11]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()
```

```
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-03-16 16:18] [root] INFO: getUserAgent: Begin
[2017-03-16 16:18] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-03-16 16:18] [root] INFO: getUserAgent: End
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: 8/10: number of genes with at least one experimental
[2017-03-16 16:18] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

```
Out[11]: pdb_id pdb_chain_id uniprot experimental_method  resolution  coverage  \
         gene
```

```
b1130   2pl1            A  P23836   X-ray diffraction       1.90     0.543
b1130   2pkx            A  P23836   X-ray diffraction       2.54     0.543
b1130   2pkx            B  P23836   X-ray diffraction       2.54     0.543
b0995   1zgz            A  P38684   X-ray diffraction       1.80     0.530
b0995   1zgz            B  P38684   X-ray diffraction       1.80     0.530

       start  end  unp_start  unp_end  rank
gene
b1130      1  121          1      121     1
b1130      1  121          1      121     2
b1130      1  121          1      121     3
b0995      1  122          1      122     1
b0995      1  122          1      122     2
```

```
In [12]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)
```

```
[2017-03-08 13:03] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-03-08 13:03] [ssbio.pipeline.gempro] INFO: 1: number of genes with additional structures added
```

```
Out[12]: pdb_id pdb_chain_id  hit_score     hit_evalue  hit_percent_similar  \
         gene
         b1013   4x1e            A      966.0  1.283880e-104             0.910377
         b1013   4x1e            B      966.0  1.283880e-104             0.910377

                hit_percent_ident  hit_num_ident  hit_num_similar
         gene
         b1013           0.910377            193              193
         b1013           0.910377            193              193
```

Downloading and ranking structures

Methods

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [13]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
[2017-03-08 13:04] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-03-08 13:04] [ssbio.pipeline.gempro] INFO: Saved 40 structures total
```

```
Out[13]: chemicals                              date  \
         gene
         b1130  PT;MG;BEF  2007-05-22;2007-08-14;2009-02-24
         b1130        NaN  2007-05-22;2007-08-14;2009-02-24

                                        description experimental_method  \
         gene
         b1130  Transcriptional regulatory protein phoP   X-ray diffraction
         b1130  Transcriptional regulatory protein phoP   X-ray diffraction
```

```
      mapped_chains pdb_id                                          pdb_title  \
gene
b1130             A   2pl1  Berrylium Fluoride activated receiver domain o...
b1130           A;B   2pkx     E.coli response regulator PhoP receiver domain

       resolution structure_file     taxonomy_name
gene
b1130        1.90       2pl1.cif  Escherichia coli
b1130        2.54       2pkx.cif  Escherichia coli
```

```
In [14]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
[2017-03-08 13:04] [ssbio.core.protein] WARNING: b1130: no structures meet quality checks
[2017-03-08 13:04] [ssbio.core.protein] WARNING: b1014: no structures meet quality checks
[2017-03-08 13:05] [ssbio.core.protein] WARNING: b0995: no structures meet quality checks
[2017-03-08 13:05] [ssbio.pipeline.gempro] INFO: 5/10: number of genes with a representative structur
[2017-03-08 13:05] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute fo
```

```
Out[14]: id is_experimental reference_seq  reference_seq_top_coverage
         gene
         b0889  2gqq-A            True        P0ACJ0                        93.3
         b0761  1b9m-A            True        P0A9G8                        96.2
         b1221  1a04-A            True        P0AF28                        94.9
         b1013  4jyk-A            True        P0ACU2                        94.8
         b1187  1hw1-A            True        P0A8V6                        94.6
```

```
In [15]: # Looking at the information saved within a gene
         my_gempro.genes.get_by_id('b1187').protein.representative_structure
         my_gempro.genes.get_by_id('b1187').protein.representative_structure.get_dict()
```

```
Out[15]: <StructProp 1hw1-A at 0x7f794cd88278>
```

```
Out[15]: {'_structure_dir': '/tmp/genes_GP/genes/b1187/b1187_protein/structures',
          'chains': [<ChainProp A at 0x7f79452bfb70>],
          'date': ['2001-01-24', '2001-06-06', '2003-04-01', '2009-02-24'],
          'description': 'FATTY ACID METABOLISM REGULATOR PROTEIN',
          'file_type': 'pdb',
          'id': '1hw1-A',
          'is_experimental': True,
          'mapped_chains': ['A'],
          'original_pdb_id': '1hw1',
          'reference_seq': <SeqProp P0A8V6 at 0x7f79452beb70>,
          'reference_seq_top_coverage': 94.6,
          'representative_chain': <ChainProp A at 0x7f79452be7f0>,
          'resolution': 1.5,
          'structure_file': '1hw1-A_clean.pdb',
          'taxonomy_name': 'Escherichia coli'}
```

Saving your GEM-PRO

Warning: Saving is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

```
In [12]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-03-16 16:18] [ssbio.core.io] INFO: Saved <class 'ssbio.pipeline.gempro.GEMPRO'> (id: genes_GP)
```

GEM-PRO - SBML Model (iNJ661)

This notebook gives an example of how to run the GEM-PRO pipeline with a **SBML model**, in this case *i*NJ661, the metabolic model of *M. tuberculosis*.

Input: GEM (in SBML, JSON, or MAT formats)

Output: GEM-PRO model

Imports

```
In [1]: import sys
        import logging
```

```
In [2]: # Import the GEM-PRO class
        from ssbio.pipeline.gempro import GEMPRO
```

```
In [3]: # Printing multiple outputs per cell
        from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

Logging

Set the logging level in `logger.setLevel(logging.<LEVEL_HERE>)` to specify how verbose you want the pipeline to be. Debug is most verbose.

- `CRITICAL`
  - Only really important messages shown
- `ERROR`
  - Major errors
- `WARNING`
  - Warnings that don't affect running of the pipeline
- `INFO` (default)
  - Info such as the number of structures mapped per gene
- `DEBUG`
  - Really detailed information that will print out a lot of stuff

Warning: `DEBUG` mode prints out a large amount of information, especially if you have a lot of genes. This may stall your notebook!

```
In [4]: # Create logger
        logger = logging.getLogger()
        logger.setLevel(logging.INFO)  # SET YOUR LOGGING LEVEL HERE #

In [5]: # Other logger stuff for Jupyter notebooks
        handler = logging.StreamHandler(sys.stderr)
        formatter = logging.Formatter('[%(asctime)s] [%(name)s] %(levelname)s: %(message)s', datefmt=
        handler.setFormatter(formatter)
        logger.handlers = [handler]
```

Initialization of the project

Set these three things:

- `ROOT_DIR`
  - The directory where a folder named after your `PROJECT` will be created
- `PROJECT`
  - Your project name
- `LIST_OF_GENES`
  - Your list of gene IDs

A directory will be created in `ROOT_DIR` with your `PROJECT` name. The folders are organized like so:

```
ROOT_DIR
- PROJECT
    - data  # General storage for pipeline outputs
    - model  # SBML and GEM-PRO models are stored here
    - genes  # Per gene information
    |   - <gene_id1>  # Specific gene directory
    |   |   - protein
    |   |       - sequences  # Protein sequence files, alignments, etc.
    |   |       - structures  # Protein structure files, calculations, etc.
    |   - <gene_id2>
    |       - protein
    |           - sequences
    |           - structures
    - reactions  # Per reaction information
    |   - <reaction_id1>  # Specific reaction directory
    |       - complex
    |           - structures  # Protein complex files
    - metabolites  # Per metabolite information
        - <metabolite_id1>  # Specific metabolite directory
            - chemical
                - structures  # Metabolite 2D and 3D structure files
```

Note: Methods for protein complexes and metabolites are still in development.

```
In [6]: # SET FOLDERS AND DATA HERE
        import tempfile
        ROOT_DIR = tempfile.gettempdir()
        # ROOT_DIR = '/home/nathan/projects_unsynced/'
        PROJECT = 'mtuberculosis_gp_atlas'
        GEM_FILE = '/home/nathan/projects_unsynced/mtuberculosis_gp_atlas/model/iNJ661.json'
        GEM_FILE_TYPE = 'json'
```

```
In [7]: # Create the GEM-PRO project
        my_gempro = GEMPRO(gem_name=PROJECT, root_dir=ROOT_DIR, gem_file_path=GEM_FILE, gem_file_type

[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: iNJ661: loaded model
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 1025: number of reactions
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 720: number of reactions linked to a gene
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 661: number of genes (excluding spontaneous)
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 826: number of metabolites
[2017-03-29 19:24] [ssbio.pipeline.gempro] WARNING: IMPORTANT: All Gene objects have been transformed
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: /home/nathan/projects_unsynced/mtuberculosis_gp_atla
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 661: number of genes
```

Mapping gene ID –> sequence

First, we need to map these IDs to their protein sequences. There are 2 ID mapping services provided to do this - through **KEGG** or **UniProt**. The end goal is to map a UniProt ID to each ID, since there is a comprehensive mapping (and some useful APIs) between UniProt and the PDB.

Note: You only need to map gene IDs using one service. However you can run both if some genes don't map in one service and do map in another!

However, you don't need to map using these services if you already have the amino acid sequences for each protein. You can just manually load in the sequences as shown using the method `manual_seq_mapping`. Or, if you already have the UniProt IDs, you can load those in using the method `manual_uniprot_mapping`.

Methods

```
In [8]: gene_to_seq_dict = {'Rv1295': 'MTVPPTATHQPWPGVIAAYRDRLPVGDDWTPVTLLEGGTPLIAATNLSKQTGCTIHLKVEGI
                            'Rv2233': 'VSSPRERRPASQAPRLSRRPPAHQTSRSSPDTTAPTGSGLSNRFVNDNGIVTDTTASGTNCI
        my_gempro.manual_seq_mapping(gene_to_seq_dict)

[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Loaded in 2 sequences

In [9]: manual_uniprot_dict = {'Rv1755c': 'P9WIA9', 'Rv2321c': 'P71891', 'Rv0619': 'Q79FY3', 'Rv0618
        my_gempro.manual_uniprot_mapping(manual_uniprot_dict)
        my_gempro.df_uniprot_metadata.tail(4)

[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Completed manual ID mapping --> UniProt. See the "di

Out[9]: uniprot reviewed gene_name            kegg  \
        gene
        Rv0619   Q79FY3    False     galTb             NaN
        Rv0618   Q79FY4    False     galTa   mtv:RVBD_0618
        Rv2321c  P71891    False     rocD2  mtv:RVBD_2321c
        Rv2322c  P71890    False     rocD1  mtv:RVBD_2322c

                                    refseq  num_pdbs ec_number     pfam  seq_len  \
        gene
        Rv0619                         NaN         0       NaN  PF02744      181
        Rv0618                         NaN         0       NaN  PF01087      231
        Rv2321c                        NaN         0       NaN  PF00202      181
        Rv2322c  WP_003411957.1;NZ_KK339370.1         0       NaN  PF00202      221

                                                    description entry_version  \
        gene
        Rv0619   Probable galactose-1-phosphate uridylyltransfe...    2016-11-02
```

```
        Rv0618   Probable galactose-1-phosphate uridylyltransfe...    2016-11-30
        Rv2321c  Probable ornithine aminotransferase (C-terminu...    2016-11-02
        Rv2322c  Probable ornithine aminotransferase (N-terminu...    2016-11-30

                seq_version sequence_file metadata_file
        gene
        Rv0619   2004-07-05  Q79FY3.fasta    Q79FY3.txt
        Rv0618   2004-07-05  Q79FY4.fasta    Q79FY4.txt
        Rv2321c  1997-02-01  P71891.fasta    P71891.txt
        Rv2322c  1997-02-01  P71890.fasta    P71890.txt
```

In [10]:
```
# KEGG mapping of gene ids
my_gempro.kegg_mapping_and_metadata(kegg_organism_code='mtu')
print('Missing KEGG mapping: ', my_gempro.missing_kegg_mapping)
my_gempro.df_kegg_metadata.head()
```

```
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv1755c: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv1755c: no metadata file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2233: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2233: no metadata file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv0619: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv0619: no metadata file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv0618: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2321c: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2321c: no metadata file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2322c: no sequence file available
[2017-03-29 19:24] [root] WARNING: status is not ok with Not Found
[2017-03-29 19:24] [ssbio.databases.kegg] WARNING: mtu:Rv2322c: no metadata file available
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 655/661: number of genes mapped to KEGG
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> KEGG. See the "df_kegg_meta

Missing KEGG mapping:  ['Rv2322c', 'Rv2233', 'Rv0619', 'Rv1755c', 'Rv0618', 'Rv2321c']
```

Out[10]:
```
        kegg     refseq uniprot  num_pdbs  pdbs  seq_len  \
        gene
        Rv0417  mtu:Rv0417  NP_214931  P9WG73         0   NaN      252
        Rv2291  mtu:Rv2291  NP_216807  P9WHF5         0   NaN      284
        Rv3737  mtu:Rv3737  NP_218254  O69704         0   NaN      529
        Rv1295  mtu:Rv1295  NP_215811  P9WG59         1  2D1F      360
        Rv1559  mtu:Rv1559  NP_216075  P9WG95         0   NaN      429

                 sequence_file    metadata_file
        gene
        Rv0417  mtu-Rv0417.faa  mtu-Rv0417.kegg
        Rv2291  mtu-Rv2291.faa  mtu-Rv2291.kegg
        Rv3737  mtu-Rv3737.faa  mtu-Rv3737.kegg
        Rv1295  mtu-Rv1295.faa  mtu-Rv1295.kegg
        Rv1559  mtu-Rv1559.faa  mtu-Rv1559.kegg
```

In [11]:
```
# UniProt mapping
```

```
my_gempro.uniprot_mapping_and_metadata(model_gene_source='TUBERCULIST_ID')
print('Missing UniProt mapping: ', my_gempro.missing_uniprot_mapping)
my_gempro.df_uniprot_metadata.head()
```

```
[2017-03-29 19:24] [root] INFO: getUserAgent: Begin
[2017-03-29 19:24] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-03-29 19:24] [root] INFO: getUserAgent: End
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 589/661: number of genes mapped to UniProt
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro
```

```
Missing UniProt mapping:  ['Rv0156', 'Rv2398c', 'Rv0649', 'Rv0511', 'Rv0266c', 'Rv2458', 'Rv2062c',
```

Out[11]:

gene	uniprot	reviewed	gene_name	kegg	refseq	num_pdbs	pdbs	ec_number	pfam	seq_len	description	entry_version	seq_version	sequence_file	metadata_file
Rv0417	P9WG73	True	thiG	mtu:Rv0417	NP_214931.1;NC_000962.3;WP_003916659.1;NZ_KK33...	0	NaN	2.8.1.10	NaN	252	Thiazole synthase {ECO:0000255|HAMAP-Rule:MF_0...	2016-11-02	2014-04-16	P9WG73.fasta	P9WG73.txt
Rv2291	P9WHF5	True	sseB	mtu:Rv2291	NP_216807.1;NC_000962.3;WP_003899253.1;NZ_KK33...	0	NaN	2.8.1.1	PF00581	284	Putative thiosulfate sulfurtransferase SseB	2016-11-02	2014-04-16	P9WHF5.fasta	P9WHF5.txt
Rv1295	P9WG59	True	thrC	mtu:Rv1295	NP_215811.1;NC_000962.3;WP_003406652.1;NZ_KK33...	1	2D1F	4.2.3.1	PF00291	360	TS;Threonine synthase	2016-11-02	2014-04-16	P9WG59.fasta	P9WG59.txt
Rv1559	P9WG95	True	ilvA	mtu:Rv1559	NP_216075.1;NC_000962.3;WP_003407781.1;NZ_KK33...	0	NaN	4.3.1.19	PF00291;PF00585	429	Threonine deaminase;L-threonine dehydratase bi...	2016-11-02	2014-04-16	P9WG95.fasta	P9WG95.txt
Rv2447c	I6Y0R5	False	folC	mtu:Rv2447c;mtv:RVBD_2447c	NP_216963.1;NC_000962.3;WP_003899324.1;NZ_KK33...	0	NaN	NaN	PF02875;PF08245	487	Probable folylpolyglutamate synthase protein F...	2016-11-02	2012-10-03	I6Y0R5.fasta	I6Y0R5.txt

If you have mapped with both KEGG and UniProt mappers, then you can set a representative sequence for the gene using this function. If you used just one, this will just set that ID as representative.

- If any sequences or IDs were provided manually, these will be set as representative first.
- UniProt mappings override KEGG mappings except when KEGG mappings have PDBs associated with them and UniProt doesn't.

```
In [12]: # Set representative sequences
         my_gempro.set_representative_sequence()
         print('Missing a representative sequence: ', my_gempro.missing_representative_sequence)
         my_gempro.df_representative_sequences.head()
```

```
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 661/661: number of genes with a representative seque
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for
```

```
Missing a representative sequence:  []
```

```
Out[12]: uniprot        kegg  num_pdbs  pdbs  seq_len   sequence_file
         gene
         Rv0417  P9WG73  mtu:Rv0417         0   NaN      252    P9WG73.fasta
         Rv2291  P9WHF5  mtu:Rv2291         0   NaN      284    P9WHF5.fasta
         Rv3737  O69704  mtu:Rv3737         0   NaN      529  mtu-Rv3737.faa
         Rv1295  P9WG59  mtu:Rv1295         1  2D1F      360      Rv1295.faa
         Rv1559  P9WG95  mtu:Rv1559         0   NaN      429    P9WG95.fasta
```

Mapping representative sequence –> structure

These are the ways to map sequence to structure:

1. Use the UniProt ID and their automatic mappings to the PDB
2. BLAST the sequence to the PDB
3. Make homology models or
4. Map to existing homology models

You can only utilize option #1 to map to PDBs if there is a mapped UniProt ID set in the representative sequence. If not, you'll have to BLAST your sequence to the PDB or make a homology model. You can also run both for maximum coverage.

Methods

```
In [13]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()
```

```
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-03-29 19:24] [root] INFO: getUserAgent: Begin
[2017-03-29 19:24] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-03-29 19:24] [root] INFO: getUserAgent: End
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: 178/661: number of genes with at least one experimer
[2017-03-29 19:24] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

```
Out[13]: pdb_id pdb_chain_id uniprot experimental_method  resolution  coverage  \
         gene
         Rv1295    2d1f            A  P9WG59   X-ray diffraction        2.50     1.000
         Rv1295    2d1f            B  P9WG59   X-ray diffraction        2.50     1.000
         Rv1201c   3fsy            A  P9WP21   X-ray diffraction        1.97     0.997
         Rv1201c   3fsy            B  P9WP21   X-ray diffraction        1.97     0.997
         Rv1201c   3fsy            C  P9WP21   X-ray diffraction        1.97     0.997

                  start  end  unp_start  unp_end  rank
         gene
```

```
Rv1295       1  360          1      360     1
Rv1295       1  360          1      360     2
Rv1201c      4  319          2      317     1
Rv1201c      4  319          2      317     2
Rv1201c      4  319          2      317     3
```

```
In [14]: # Mapping using BLAST
         my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.9, evalue=0.00001)
         my_gempro.df_pdb_blast.head(2)
```

```
[2017-03-29 19:25] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-03-29 19:25] [ssbio.pipeline.gempro] INFO: 30: number of genes with additional structures added
```

```
Out[14]: pdb_id pdb_chain_id  hit_score  hit_evalue  hit_percent_similar  \
         gene
         Rv1908c   4c50            A     3652.0         0.0             0.974324
         Rv1908c   4c50            B     3652.0         0.0             0.974324

                  hit_percent_ident  hit_num_ident  hit_num_similar
         gene
         Rv1908c           0.974324            721              721
         Rv1908c           0.974324            721              721
```

```
In [15]: tb_homology_dir = '/home/nathan/projects_archive/homology_models/MTUBERCULOSIS/'

         ##### EXAMPLE SPECIFIC CODE #####
         # Needed to map to older IDs used in this example
         import pandas as pd
         import os.path as op
         old_gene_to_homology = pd.read_csv(op.join(tb_homology_dir, 'data/161031-old_gene_to_uniprot
         gene_to_uniprot = old_gene_to_homology.set_index('m_gene').to_dict()['u_uniprot_acc']
         my_gempro.get_itasser_models(homology_raw_dir=op.join(tb_homology_dir, 'raw'), custom_itasse
         ### END EXAMPLE SPECIFIC CODE ###

         # Organizing I-TASSER homology models
         my_gempro.get_itasser_models(homology_raw_dir=op.join(tb_homology_dir, 'raw'))
         my_gempro.df_homology_models.head()
```

```
[2017-03-29 19:25] [ssbio.pipeline.gempro] INFO: Completed copying of 435 I-TASSER models to GEM-PRO
```

```
[2017-03-29 19:25] [ssbio.pipeline.gempro] INFO: Completed copying of 9 I-TASSER models to GEM-PRO di
```

```
Out[15]: c_score difficulty      id  model_date         model_file  rmsd  \
         gene
         Rv0417      1.66       easy  P9WG73  2015-12-30  P9WG73_model1.pdb   2.6
         Rv2291      1.38       easy  P9WHF5  2016-01-04  P9WHF5_model1.pdb   3.3
         Rv1559      0.73       easy  P9WG95  2016-01-08  P9WG95_model1.pdb   5.4
         Rv3113      0.72       easy  O05790  2015-12-30  O05790_model1.pdb   4.1
         Rv2447c     0.07       easy  I6Y0R5  2016-01-08  I6Y0R5_model1.pdb   7.1

                  rmsd_err  tm_score  tm_score_err top_template_chain top_template_pdb
         gene
         Rv0417        1.9      0.95          0.05                 C             2htm
         Rv2291        2.3      0.91          0.06                 A             3olh
         Rv1559        3.4      0.81          0.09                 A             1tdj
         Rv3113        2.8      0.81          0.09                 A             3sd7
         Rv2447c       4.2      0.72          0.11                 A             2vos
```

```
In [16]: homology_model_dict = {}
         my_gempro.get_manual_homology_models(homology_model_dict)
```

```
[2017-03-29 19:25] [ssbio.pipeline.gempro] INFO: Updated homology model information for 0 genes.
```

Downloading and ranking structures

Methods

Warning: Downloading all PDBs takes a while, since they are also parsed for metadata. You can skip this step and just set representative structures below if you want to minimize the number of PDBs downloaded.

```
In [17]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
616/|/ 93%|| 616/661 [31:01<02:16,  3.02s/it]
```

```
[2017-03-29 19:57] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-03-29 19:57] [ssbio.pipeline.gempro] INFO: Saved 937 structures total
```

```
Out[17]: chemicals                                          date  \
         gene
         Rv1295                  PLP  2006-09-05;2009-02-24;2009-04-28;2011-07-13
         Rv1201c  SCA;MPD;MG;NA;ACY                      2009-06-23;2011-07-13

                                                    description  \
         gene
         Rv1295                     Threonine synthase (E.C.4.2.3.1)
         Rv1201c  Tetrahydrodipicolinate N-succinyltransferase (...

                 experimental_method mapped_chains pdb_id  \
         gene
         Rv1295    X-ray diffraction           A;B   2d1f
         Rv1201c   X-ray diffraction     A;B;C;D;E   3fsy

                                                    pdb_title  resolution  \
         gene
         Rv1295   Structure of Mycobacterium tuberculosis threon...        2.50
         Rv1201c  Structure of tetrahydrodipicolinate N-succinyl...        1.97

                 structure_file               taxonomy_name
         gene
         Rv1295        2d1f.cif  Mycobacterium tuberculosis
         Rv1201c       3fsy.cif  Mycobacterium tuberculosis
```

```
In [18]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
[2017-03-29 19:58] [ssbio.core.protein] WARNING: Rv0432: no structures meet quality checks
[2017-03-29 19:58] [ssbio.core.protein] WARNING: Rv1286: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2934: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2932: no structures meet quality checks
```

```
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2933: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2931: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2945c: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2941: no structures meet quality checks
[2017-03-29 19:59] [ssbio.core.protein] WARNING: Rv2495c: no structures meet quality checks
[2017-03-29 20:00] [ssbio.core.protein] WARNING: Rv2380c: no structures meet quality checks
[2017-03-29 20:01] [ssbio.core.protein] WARNING: Rv2987c: no structures meet quality checks
[2017-03-29 20:02] [ssbio.core.protein] WARNING: Rv3859c: no structures meet quality checks
[2017-03-29 20:02] [ssbio.core.protein] WARNING: Rv2476c: no structures meet quality checks
[2017-03-29 20:03] [ssbio.core.protein] WARNING: Rv1653: no structures meet quality checks
[2017-03-29 20:04] [ssbio.core.protein] WARNING: Rv3800c: no structures meet quality checks
[2017-03-29 20:05] [ssbio.core.protein] WARNING: Rv2498c: no structures meet quality checks
[2017-03-29 20:05] [ssbio.core.protein] WARNING: Rv1885c: no structures meet quality checks
[2017-03-29 20:05] [ssbio.core.protein] WARNING: Rv3601c: no structures meet quality checks
[2017-03-29 20:05] [ssbio.core.protein] WARNING: Rv3330: no structures meet quality checks
[2017-03-29 20:06] [ssbio.core.protein] WARNING: Rv3793: no structures meet quality checks
[2017-03-29 20:06] [ssbio.core.protein] WARNING: Rv2940c: no structures meet quality checks
[2017-03-29 20:06] [ssbio.core.protein] WARNING: Rv1662: no structures meet quality checks
[2017-03-29 20:06] [ssbio.core.protein] WARNING: Rv2524c: no structures meet quality checks
[2017-03-29 20:06] [ssbio.core.protein] WARNING: Rv1625c: no structures meet quality checks
[2017-03-29 20:07] [ssbio.pipeline.gempro] INFO: 590/661: number of genes with a representative struc
[2017-03-29 20:07] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

Out[18]:
```
         id is_experimental  reference_seq_top_coverage  \
gene
Rv0417  P9WG73-X          False                       100.0
Rv2291  P9WHF5-X          False                       100.0
Rv1295    2d1f-A           True                        96.9
Rv1559  P9WG95-X          False                       100.0
Rv3113  O05790-X          False                       100.0

                    structure_file
gene
Rv0417  P9WG73_model1-X_clean.pdb
Rv2291  P9WHF5_model1-X_clean.pdb
Rv1295          2d1f-A_clean.pdb
Rv1559  P9WG95_model1-X_clean.pdb
Rv3113  O05790_model1-X_clean.pdb
```

In [19]:
```
# Looking at the information saved within a gene
my_gempro.genes.get_by_id('Rv1295').protein.representative_structure
my_gempro.genes.get_by_id('Rv1295').protein.representative_structure.get_dict()
```

Out[19]:
```
<StructProp 2d1f-A at 0x7fb5340cbb00>
```

Out[19]:
```
{'_structure_dir': '/home/nathan/projects_unsynced/mtuberculosis_gp_atlas/genes/Rv1295/Rv129
 'chains': [<ChainProp A at 0x7fb52c11cfd0>],
 'date': ['2006-09-05', '2009-02-24', '2009-04-28', '2011-07-13'],
 'description': 'Threonine synthase (E.C.4.2.3.1)',
 'file_type': 'pdb',
 'id': '2d1f-A',
 'is_experimental': True,
 'mapped_chains': ['A'],
 'original_pdb_id': '2d1f',
 'reference_seq_top_coverage': 96.9,
 'representative_chain': <ChainProp A at 0x7fb52c11cc88>,
 'resolution': 2.5,
 'structure_file': '2d1f-A_clean.pdb',
 'taxonomy_name': 'Mycobacterium tuberculosis'}
```

Creating homology models

For those proteins with no representative structure, we can create homology models for them. `ssbio` contains some built in functions for easily running I-TASSER locally or on machines with `SLURM` (ie. on NERSC) or `Torque` job scheduling.

You can load in I-TASSER models once they complete using the `get_itasser_models` later.

Info: Homology modeling can take a long time - about 24-72 hours per protein (highly dependent on the sequence length, as well as if there are available templates).

Methods

```
In [20]: # Prep I-TASSER model folders
         my_gempro.prep_itasser_models('~/software/I-TASSER4.4', '~/software/ITLIB/', runtype='local
```

```
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2934: I-TASSER r
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2932: I-TASSER r
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2933: I-TASSER r
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2931: I-TASSER r
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2380c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv3859c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2476c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv3800c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv0107c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2940c: I-TASSER
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv1662: I-TASSER r
[2017-03-29 20:07] [ssbio.protein.structure.homology.itasser.itasserprep] WARNING: Rv2524c: I-TASSER
[2017-03-29 20:07] [ssbio.pipeline.gempro] INFO: Prepared I-TASSER modeling folders for 71 genes in
```

Saving your GEM-PRO

Finally, you can save your GEM-PRO as a `JSON` or `pickle` file, so you don't have to run the pipeline again.

For most functions, if you rerun them, they will check for existing results saved as files. The only function that would take a long time is setting the representative structure, as they are each rechecked and cleaned. This is where saving helps!

Warning: Saving in JSON format is still experimental. For a full GEM-PRO with sequences & structures, depending on the number of genes, saving can take >5 minutes.

```
In [21]: import os.path as op
         my_gempro.save_pickle(op.join(my_gempro.model_dir, '{}.pckl'.format(my_gempro.id)))
```

```
Out[21]: '/home/nathan/projects_unsynced/mtuberculosis_gp_atlas/model/mtuberculosis_gp_atlas.pckl'
```

```
In [22]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-03-29 20:07] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-03-29 20:08] [ssbio.core.io] INFO: Saved <class 'ssbio.pipeline.gempro.GEMPRO'> (id: mtubercul
```

Loading a saved GEM-PRO

```
In [22]: # Loading a pickle file
         import pickle
         with open(op.join(my_gempro.model_dir, '{}.pckl'.format(my_gempro.id)), 'rb') as f:
             my_saved_gempro = pickle.load(f)

In [23]: # Loading a JSON file
         import ssbio.core.io
         my_saved_gempro = ssbio.core.io.load_json(op.join(my_gempro.model_dir, '{}.json'.format(my_g
```

The ATLAS Pipeline

Introduction

The ATLAS pipeline is focused on the comparison of multiple strains or species.

Tutorials

Python API

class ssbio.core.protein.**Protein**(*ident*, *description=None*, *root_dir=None*, *pdb_file_type='cif'*)

Generic class to store information on a protein, which may consist of multiple sequences and structures.

The main utilities of this class are to:

1. **Load, parse, and store multiple sources of the same or similar (ie. from different strains) protein sequences** as `SeqProp` objects
2. Load, parse, and store multiple experimental or predicted protein structures as `StructProp` objects
3. Calculate, store, and access sequence alignments to stored sequences or structures
4. Provide summaries of alignments
5. Map between residue numbers of sequences and structures

id
str – Unique identifier for this protein

description
str – Optional description for this protein

sequences
DictList – Path to FASTA file

structures
DictList – Path to generic metadata file

representative_sequence
SeqProp – Sequence set to represent this protein

representative_structure
StructProp – Structure set to represent this protein, cleaned and monomeric

representative_chain
str – Chain ID in the representative structure which best represents a sequence

representative_chain_seq_coverage
float – Percent identity of sequence coverage for the representative chain

sequence_alignments
DictList – Pairwise or multiple sequence alignments stored as `Bio.Align.MultipleSeqAlignment` objects

structure_alignments
DictList – Pairwise or multiple structure alignments - incomplete implementation

root_dir
str – Path to where the folder named by this protein's ID will be created. Default is current working directory.

pdb_file_type
str – `pdb`, `pdb.gz`, `mmcif`, `cif`, `cif.gz`, `xml.gz`, `mmtf`, `mmtf.gz` - choose a file type for files downloaded from the PDB

align_seqprop_to_structprop(*seqprop*, *structprop*, *chains=None*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*, ***kwargs*)
Run and store alignments of a SeqProp to chains in the `mapped_chains` attribute of a StructProp.

Alignments are stored in the sequence_alignments attribute, with the IDs formatted as <SeqProp_ID>_<StructProp_ID>-<Chain_ID>. Although it is more intuitive to align to individual ChainProps, StructProps should be loaded as little as possible to reduce run times.

Parameters

- **seqprop** (`SeqProp`) – SeqProp object with a loaded sequence
- **structprop** (`StructProp`) – StructProp object with a loaded structure
- **chains** (*str, list*) – Chain ID or IDs to map to. If not specified, `mapped_chains` attribute is inspected for chains. If no chains there, all chains will be aligned to.
- **outdir** (*str*) – Directory to output sequence alignment files (only if running with needle)
- **engine** (*str*) – Which pairwise alignment tool to use ("needle" or "biopython")
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** (*bool*) – If alignments should be rerun
- ****kwargs** – Other alignment options

blast_representative_sequence_to_pdb(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)
BLAST repseq to PDB and return the list of new structures added, also saves df_pdb_blast

df_homology_models
Get a dataframe of homology models

df_pdb_blast
Get a dataframe of PDB BLAST results

df_pdb_metadata
Get a dataframe of PDB metadata (PDBs have to be downloaded first)

df_pdb_ranking
Get a dataframe of UniProt -> best structure in PDB results

filter_sequences(*seq_type*)
Return a DictList of only specified types in the sequences attribute.

Parameters **seq_type** (SeqProp) – Object type

Returns A filtered DictList of specified object type only

Return type DictList

find_representative_chain(*seqprop*, *structprop*, *chains_to_check=None*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*)
Set and return the representative chain based on sequence quality checks to a reference sequence.

Parameters

- **seqprop** (SeqProp) – SeqProp object to compare to chain sequences
- **structprop** (StructProp) – StructProp object with chains to compare to in the mapped_chains attribute. If there are none present, chains_to_check can be specified, otherwise all chains are checked.
- **chains_to_check** (*str, list*) – Chain ID or IDs to check for sequence coverage quality
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for

Returns the best chain ID, if any

Return type str

get_disulfide_bridges(*representative_only=True*)
Run Biopython's disulfide bridge finder and store found bridges. Annotations are stored in the protein structure's chain sequence at: seq_record.annotations['SSBOND-biopython']

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative structure

get_dssp_annotations(*representative_only=True*, *force_rerun=False*)
Run DSSP on structures and store calculations. Annotations are stored in the protein structure's chain sequence at: seq_record.letter_annotations['*-dssp']

Todo

•Some errors arise from storing annotations for nonstandard amino acids, need to run DSSP separately for those

---

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_experimental_structures()
DictList: Return a DictList of all experimental structures in self.structures

get_freesasa_annotations(*include_hetatms=False*, *representative_only=True*, *force_rerun=False*)
Run freesasa on structures and store calculations. Annotations are stored in the protein structure's chain sequence at: `seq_record.letter_annotations['*-freesasa']`

Parameters

- **include_hetatms** (*bool*) – If HETATMs should be included in calculations. Defaults to `False`.
- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_homology_models()
DictList: Return a DictList of all homology models in self.structures

get_msms_annotations(*representative_only=True*, *force_rerun=False*)
Run MSMS on structures and store calculations. Annotations are stored in the protein structure's chain sequence at: `seq_record.letter_annotations['*-msms']`

Parameters

- **representative_only** (*bool*) – If analysis should only be run on the representative structure
- **force_rerun** (*bool*) – If calculations should be rerun even if an output file exists

get_sequence_properties(*representative_only=True*)
Run Biopython ProteinAnalysis and EMBOSS pepstats to summarize basic statistics of the protein sequences. Annotations are stored in the protein's respective seqprop objects at: `.seq_record.annotations`

Parameters **representative_only** (*bool*) – If analysis should only be run on the representative sequence

load_itasser_folder(*ident*, *itasser_folder*, *organize=False*, *outdir=None*, *organize_name=None*, *set_as_representative=False*, *create_dfs=False*, *force_rerun=False*)
Load the results folder from I-TASSER, copy structure files over, and create summary dataframes.

Parameters

- **ident** – I-TASSER ID
- **itasser_folder** – Path to results folder
- **organize** (*bool*) – If select files from modeling should be copied to the Protein directory
- **outdir** (*str*) – Path to directory where files will be copied and organized to
- **organize_name** (*str*) – Basename of files to rename results to. If not provided, will use id attribute.
- **set_as_representative** – If this structure should be set as the representative structure

- **create_dfs** – If summary dataframes should be created
- **parse** – If the structure's 3D coordinates and chains should be parsed
- **force_rerun** –

Returns StructProp object stored in the structures list.

Return type ITASSERProp

load_kegg(*kegg_id*, *kegg_organism_code=None*, *kegg_seq_file=None*, *kegg_metadata_file=None*, *set_as_representative=False*, *download=False*, *outdir=None*, *force_rerun=False*)

Load a KEGG ID, sequence, and metadata files into the sequences attribute.

Parameters

- **kegg_id** (*str*) – KEGG ID
- **kegg_organism_code** (*str*) – KEGG organism code to prepend to the kegg_id if not part of it already. Example: `eco:b1244`, eco is the organism code
- **kegg_seq_file** (*str*) – Path to KEGG FASTA file
- **kegg_metadata_file** (*str*) – Path to KEGG metadata file (raw KEGG format)
- **set_as_representative** (*bool*) – If this KEGG ID should be set as the representative sequence
- **download** (*bool*) – If the KEGG sequence and metadata files should be downloaded if not provided
- **outdir** (*str*) – Where the sequence and metadata files should be downloaded to
- **force_rerun** (*bool*) – If ID should be reloaded and files redownloaded

Returns object contained in the sequences attribute

Return type KEGGProp

load_manual_sequence(*seq*, *ident=None*, *write_fasta_file=False*, *outname=None*, *outdir=None*, *set_as_representative=False*, *force_rewrite=False*)

Load a manual sequence given as a string and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **seq** (*str, Seq, SeqRecord*) – Sequence string, Biopython Seq or SeqRecord object
- **ident** (*str*) – Optional identifier for the sequence, required if seq is a string. Also will override existing IDs in Seq or SeqRecord objects if set.
- **write_fasta_file** –
- **outname** –
- **outdir** –
- **set_as_representative** –
- **force_rewrite** –

Returns:

load_manual_sequence_file(*ident*, *seq_file*, *copy_file=False*, *outdir=None*, *set_as_representative=False*)

Load a manual sequence given as a FASTA file and optionally set it as the representative sequence. Also store it in the sequences attribute.

Parameters

- **ident** –
- **seq_file** –
- **copy_file** –
- **outdir** –
- **set_as_representative** –

load_pdb(*pdb_id*, *mapped_chains=None*, *pdb_file=None*, *file_type=None*, *set_as_representative=False*, *force_rerun=False*)

Load a PDB ID into the structures attribute.

Parameters

- **pdb_id** (*str*) – PDB ID
- **mapped_chains** (*str, list*) – Chain ID or list of IDs which you are interested in
- **pdb_file** (*str*) – Path to PDB file
- **file_type** (*str*) – Type of PDB file
- **set_as_representative** (*bool*) – If this structure should be set as the representative structure
- **parse** (*bool*) – If the structure's 3D coordinates and chains should be parsed

Returns The object that is now contained in the structures attribute

Return type PDBProp

load_uniprot(*uniprot_id*, *uniprot_seq_file=None*, *uniprot_xml_file=None*, *set_as_representative=False*, *download=False*, *download_metadata_file_type='xml'*, *simple_parse=False*, *outdir=None*, *force_rerun=False*)

Load a UniProt ID and associated sequence/metadata files into the sequences attribute.

Parameters

- **uniprot_id** –
- **uniprot_seq_file** –
- **uniprot_xml_file** –
- **set_as_representative** –
- **download** –
- **outdir** –
- **force_rerun** –

Returns

Return type UniProtProp

map_seqprop_resnums_to_structprop_chain_index(*resnums*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Map a residue number in the seqprop to the mapping index in the structprop/chain_id

Use this to get the indices of the chain to then get structure residue number.

Parameters

- **resnums** (*int, list*) – Residue numbers in the sequence
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – Chain ID to map to index
- **use_representatives** (*bool*) – If representative sequence/structure/chain should be used in mapping

Returns Mapping of resnums to indices

Return type dict

map_seqprop_resnums_to_structprop_resnums(*resnums*, *seqprop=None*, *structprop=None*, *chain_id=None*, *use_representatives=False*)

Map a residue number in the seqprop to the structure's residue number for a specified chain.

Parameters

- **resnums** (*int, list*) – Residue numbers in the sequence
- **seqprop** (SeqProp) – SeqProp object
- **structprop** (StructProp) – StructProp object
- **chain_id** (*str*) – Chain ID to map to

Returns Mapping of resnums to structure residue IDs

Return type dict

num_sequences

int – Return the total number of sequences

num_structures

int – Return the total number of structures

num_structures_experimental

int – Return the total number of experimental structures

num_structures_homology

int – Return the total number of homology models

pairwise_align_sequences_to_representative(*gapopen=10*, *gapextend=0.5*, *outdir=None*, *engine='needle'*, *parse=True*, *force_rerun=False*)

Align all sequences in the sequences attribute to the representative sequence. Stores the alignments the `sequence_alignments` DictList

Parameters

- **gapopen** –
- **gapextend** –
- **outdir** –
- **engine** –
- **parse** (*bool*) – Store locations of mutations, insertions, and deletions in the alignment object (as an annotation)
- **force_rerun** –

Returns:

pdb_downloader_and_metadata(*outdir=None*, *pdb_file_type=None*, *force_rerun=False*)
Download experimental structures

prep_itasser_modeling(*itasser_installation*, *itlib_folder*, *runtype*, *create_in_dir=None*, *execute_from_dir=None*, *print_exec=False*, ***kwargs*)
Prepare to run I-TASSER homology modeling for a sequence.

Parameters

- **itasser_installation** (*str*) – Path to I-TASSER folder, i.e. `~/software/I-TASSER4.4`
- **itlib_folder** (*str*) – Path to ITLIB folder, i.e. `~/software/ITLIB`
- **runtype** – How you will be running I-TASSER - local, slurm, or torque
- **create_in_dir** (*str*) – Local directory where folders will be created
- **execute_from_dir** (*str*) – Optional path to execution directory - use this if you are copying the homology models to another location such as a supercomputer for running
- **all_genes** (*bool*) – If all genes should be prepped, or only those without any mapped structures
- **print_exec** (*bool*) – If the execution statement should be printed to run modelling
- ****kwargs** –

Todo

•kwargs - extra options for SLURM or Torque execution

---

protein_dir
str – Protein folder

protein_statistics
Get a dictionary of basic statistics describing this protein

root_dir
str – Directory where Protein project folder is located

sequence_dir
str – Directory where sequence related files are stored

sequence_mutation_summary(*sequence_ids=None*, *alignment_type=None*)
Summarize all mutations found in the sequence_alignments attribute.

Returns 2 dictionaries, single_counter and fingerprint_counter.

single_counter: Dictionary of {point mutation: list of genes/strains} Example: {('A', 24, 'V'): ['Strain1', 'Strain2', 'Strain4'],

('R', 33, 'T'): ['Strain2']}

Here, we report which genes/strains have the single point mutation.

fingerprint_counter: Dictionary of {mutation group: list of genes/strains} Example: {(('A', 24, 'V'), ('R', 33, 'T')): ['Strain2'],

(('A', 24, 'V')): ['Strain1', 'Strain4']}

Here, we report which genes/strains have the specific combinations (or "fingerprints") of point mutations

Parameters

- **sequence_ids** (*str, list*) – Specified alignment ID or IDs to use
- **alignment_type** (*str*) – Specified alignment type contained in the `annotation` field of an alignment object

Returns single_counter, fingerprint_counter

Return type dict, dict

set_representative_sequence(*force_rerun=False*)

Consolidate sequences that were loaded and set a single representative sequence.

set_representative_structure(*seq_outdir=None*, *struct_outdir=None*, *pdb_file_type=None*, *engine='needle'*, *always_use_homology=False*, *rez_cutoff=0.0*, *seq_ident_cutoff=0.5*, *allow_missing_on_termini=0.2*, *allow_mutants=True*, *allow_deletions=False*, *allow_insertions=False*, *allow_unresolved=True*, *clean=True*, *keep_chemicals=None*, *force_rerun=False*)

Set a representative structure from a structure in self.structures

Parameters

- **seq_outdir** (*str*) – Path to output directory of sequence alignment files
- **struct_outdir** (*str*) – Path to output directory of structure files
- **pdb_file_type** (*str*) – pdb, pdb.gz, mmcif, cif, cif.gz, xml.gz, mmtf, mmtf.gz - PDB structure file type that should be downloaded
- **engine** (*str*) – "needle" or "biopython" - which pairwise sequence alignment engine should be used needle is the standard EMBOSS tool to run pairwise alignments biopython is Biopython's implementation of needle. Results can differ!
- **always_use_homology** (*bool*) – If homology models should always be set as the representative structure
- **rez_cutoff** (*float*) – Resolution cutoff, in Angstroms (only if experimental structure)
- **seq_ident_cutoff** (*float*) – Percent sequence identity cutoff, in decimal form
- **allow_missing_on_termini** (*float*) – Percentage of the total length of the reference sequence which will be ignored when checking for modifications. Example: if 0.1, and reference sequence is 100 AA, then only residues 5 to 95 will be checked for modifications.
- **allow_mutants** (*bool*) – If mutations should be allowed or checked for
- **allow_deletions** (*bool*) – If deletions should be allowed or checked for
- **allow_insertions** (*bool*) – If insertions should be allowed or checked for
- **allow_unresolved** (*bool*) – If unresolved residues should be allowed or checked for
- **clean** (*bool*) – If structure should be cleaned
- **keep_chemicals** (*str, list*) – Keep specified chemical names if structure is to be cleaned
- **force_rerun** (*bool*) – If sequence to structure alignment should be rerun

Returns

Representative structure from the list of structures.

This is a not a map to the original structure, it is copied from its reference.

Return type *StructProp*

structure_dir
str – Directory where structure related files are stored

summarize_protein()
Gather all possible attributes in the sequences and structures and summarize everything.

Returns

Return type dict

view_all_mutations(*grouped=False*, *color='red'*, *unique_colors=True*, *structure_opacity=0.5*, *opacity_range=(0.8*, *1)*, *scale_range=(1*, *5)*, *gui=False*)
Map all sequence alignment mutations to the structure.

Parameters

- **grouped** (*bool*) – If groups of mutations should be colored and sized together
- **color** (*str*) – Color of the mutations (overridden if unique_colors=True)
- **unique_colors** (*bool*) – If each mutation/mutation group should be colored uniquely
- **structure_opacity** (*float*) – Opacity of the protein structure cartoon representation
- **opacity_range** (*tuple*) – Min/max opacity values (mutations that show up more will be opaque)
- **scale_range** (*tuple*) – Min/max size values (mutations that show up more will be bigger)
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

ssbio.core.protein.**log = <logging.Logger object>**
protein.py

Todo

•Implement structural alignment objects

---

class ssbio.protein.sequence.seqprop.**SeqProp**(*ident*, *sequence_path=None*, *metadata_path=None*, *seq=None*, *description='<unknown description>'*, *write_fasta_file=False*, *outfile=None*, *force_rewrite=False*)
Generic class to store information on a protein sequence.

The main utilities of this class are to:

1.Provide database identifier mappings as top level attributes

2.Manipulate a sequence as a Biopython SeqRecord

3.Calculate, store, and access sequence features and annotations

4.File I/O (sequence and feature files)

bigg
str – BiGG ID for this protein

kegg
str – KEGG ID for this protein

refseq
str – RefSeq ID for this protein

uniprot
str – UniProt ID for this protein

gene_name
str – Gene name encoding this protein

pdbs
list – List of PDB IDs mapped to this protein

seq_record
SeqRecord – Biopython `SeqRecord` representation of sequence

sequence_file
str – Path to FASTA file

metadata_file
str – Path to generic metadata file

annotations
dict – Freeform dictionary of annotations, copied from any SeqRecord `annotations`

letter_annotations
dict – Per-residue annotations, copied from any SeqRecord `letter_annotations`

features
list – Sequence features, copied from any SeqRecord `features`

blast_pdb(*seq_ident_cutoff=0*, *evalue=0.0001*, *display_link=False*, *outdir=None*, *force_rerun=False*)
BLAST this sequence to the PDB

equal_to(*seq_prop*)
Test if the sequence is equal to another SeqProp object's sequence

Parameters **seq_prop** – SeqProp object

Returns If the sequences are the same

Return type bool

equal_to_fasta(*seq_file*)
Test if this sequence is equal to another sequence file.

Parameters **seq_file** – Path to another sequence file

Returns If the sequences are the same

Return type bool

get_biopython_pepstats()
Run Biopython's built in ProteinAnalysis module.

Stores statistics in the seq_record.annotations attribute.

get_dict(*only_keys=None*, *exclude_attributes=None*, *df_format=False*)
Get a copy of all attributes as a dictionary, including object properties

Parameters

- **only_keys** –
- **exclude_attributes** –
- **df_format** –

Returns:

get_emboss_pepstats()
Run the EMBOSS pepstats program on the protein sequence.

Stores statistics in the seq_record.annotations attribute. Saves a .pepstats file of the results where the sequence file is located.

load_feature_file(*gff_path*)
Load a GFF file and store features

Parameters **gff_path** – Path to GFF file.

load_letter_annotations(*annotation_name*, *letter_annotation*)
Add per-residue annotations to the letter_annotations attribute

Parameters

- **annotation_name** – Name of the key that will be used to access this annotation
- **letter_annotation** (*str, list, tuple*) – Sequence that contains information on the sequence, with the length being equal to the protein sequence length

load_metadata_path(*metadata_path*)
Load a metadata file and provide pointers to its location

Parameters **metadata_path** – Path to metadata file

load_sequence_path(*sequence_path*)
Load a sequence file and provide pointers to its location

Parameters **sequence_path** – Path to sequence file

num_pdbs
Report the number of PDB IDs mapped

seq_len
Get the sequence length

seq_str
Get the sequence formatted as a string

summary(*as_df*)
Summarize this sequence.

Returns

Return type dict

write_fasta_file(*outfile*, *force_rerun=False*)
Write a FASTA file for the protein sequence.

Parameters

- **outfile** (*str*) – Path to new FASTA file to be written to

- **force_rerun** (*bool*) – If an existing file should be overwritten

ssbio.protein.sequence.seqprop.**log = <logging.Logger object>**

seqprop.py

Todo

•Include methods to read and write GFF files so features don't need to be stored in memory

---

class ssbio.protein.structure.structprop.**StructProp**(*ident*, *description=None*, *chains=None*, *mapped_chains=None*, *structure_path=None*, *file_type=None*, *representative_chain=None*, *is_experimental=False*)

Class for protein structural properties.

add_chain_ids(*chains*)

Add chains by ID into the chains attribute

Parameters **chains** (*str, list*) – Chain ID or list of IDs

add_mapped_chain_ids(*mapped_chains*)

Add chains by ID into the mapped_chains attribute

Parameters **mapped_chains** (*str, list*) – Chain ID or list of IDs

clean_structure(*out_suffix='_clean'*, *outdir=None*, *force_rerun=False*, *remove_atom_alt=True*, *keep_atom_alt_id='A'*, *remove_atom_hydrogen=True*, *add_atom_occ=True*, *remove_res_hetero=True*, *keep_chemicals=None*, *keep_res_only=None*, *add_chain_id_if_empty='X'*, *keep_chains=None*)

Clean the structure file associated with this structure, and save it as a new file. Returns the file path.

Parameters

- **out_suffix** (*str*) – Suffix to append to original filename
- **outdir** (*str*) – Path to output directory
- **force_rerun** (*bool*) – If structure should be re-cleaned if a clean file exists already
- **remove_atom_alt** (*bool*) – Remove alternate positions
- **keep_atom_alt_id** (*str*) – If removing alternate positions, which alternate ID to keep
- **remove_atom_hydrogen** (*bool*) – Remove hydrogen atoms
- **add_atom_occ** (*bool*) – Add atom occupancy fields if not present
- **remove_res_hetero** (*bool*) – Remove all HETATMs
- **keep_chemicals** (*str, list*) – If removing HETATMs, keep specified chemical names
- **keep_res_only** (*str, list*) – Keep ONLY specified resnames, deletes everything else!
- **add_chain_id_if_empty** (*str*) – Add a chain ID if not present
- **keep_chains** (*str, list*) – Keep only these chains

Returns Path to cleaned PDB file

Return type str

get_dict_with_chain(*chain*, *only_keys=None*, *chain_keys=None*, *exclude_attributes=None*, *df_format=False*)

get_dict method which incorporates attributes found in a specific chain. Does not overwrite any attributes in the original StructProp.

Parameters

- **chain** –
- **only_keys** –
- **chain_keys** –
- **exclude_attributes** –
- **df_format** –

Returns attributes of StructProp + the chain specified

Return type dict

get_disulfide_bridges(*threshold=3.0*)
Run Biopython's search_ss_bonds to find potential disulfide bridges for each chain and store in ChainProp.

get_dssp_annotations(*outdir*, *force_rerun=False*)
Run DSSP on this structure and store the DSSP annotations in the corresponding ChainProp SeqRecords

Parameters

- **outdir** (*str*) – Path to where DSSP dataframe will be stored.
- **force_rerun** (*bool*) – If DSSP results should be recalculated

get_freesasa_annotations(*outdir*, *include_hetatms=False*, *force_rerun=False*)
Run freesasa on this structure and store the calculated properties in the corresponding ChainProp SeqRecords

get_residue_depths(*outdir*, *force_rerun=False*)
Run MSMS on this structure and store the residue depths/ca depths in the corresponding ChainProp SeqRecords

get_structure_seqs(*model*)
Store chain sequences in the corresponding ChainProp objects in the chains attribute.

load_structure_path(*structure_path*, *file_type*)
Load a structure file and provide pointers to its location

Parameters

- **structure_path** – Path to structure file
- **file_type** – Type of structure file

parse_structure()
Read the 3D coordinates of a structure file and return it as a Biopython Structure object

Also create ChainProp objects in the chains attribute

Returns Biopython Structure object

Return type Structure

view_structure(*opacity=1.0*, *recolor=True*, *gui=False*)
Use NGLviewer to display a structure in a Jupyter notebook

Parameters

- **opacity** (*float*) – Opacity of the structure
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

view_structure_and_highlight_residues(*structure_resnums*, *chain=None*, *color='red'*, *structure_opacity=0.5*, *gui=False*)

Input a residue number or numbers to view on the structure.

Parameters

- **structure_resnums** (*int, list*) – Residue number(s) to highlight, structure numbering
- **chain** (*str, list*) – Chain ID or IDs of which residues are a part of. If not provided, all chains in the mapped_chains attribute will be used. IMPORTANT: if that is also empty, all residues in all chains matching the residue numbers will be shown, which may not always be correct.
- **color** (*str*) – Color to highlight with
- **structure_opacity** (*float*) – Opacity of the protein structure cartoon representation
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

view_structure_and_highlight_residues_scaled(*structure_resnums*, *chain=None*, *color='red'*, *unique_colors=False*, *structure_opacity=0.5*, *opacity_range=(0.5, 1)*, *scale_range=(0.7, 10)*, *gui=False*)

Input a list of residue numbers to view on the structure. Or input a dictionary of residue numbers to counts to scale residues by counts (useful to view mutations).

Parameters

- **structure_resnums** (*int, list, dict*) – Residue number(s) to highlight, or a dictionary of residue number to frequency count
- **chain** (*str, list*) – Chain ID or IDs of which residues are a part of. If not provided, all chains in the mapped_chains attribute will be used. PLEASE NOTE: if that is also empty, all residues in all chains matching the residue numbers will be shown.
- **color** (*str*) – Color to highlight with
- **unique_colors** (*bool*) – If each mutation should be colored uniquely (will override color argument)
- **structure_opacity** (*float*) – Opacity of the protein structure cartoon representation
- **opacity_range** (*tuple*) – Min/max opacity values (residues that have higher frequency counts will be opaque)
- **scale_range** (*tuple*) – Min/max size values (residues that have higher frequency counts will be bigger)
- **gui** (*bool*) – If the NGLview GUI should show up

Returns NGLviewer object

- genindex

Python Module Index

S

Index

U

V

W