ssbio Documentation

Release 1.0.0a0

Nathan Mih

Sep 22, 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 genes to their encoded proteins 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
4. Demonstrate fully-featured Python scientific analysis environments in Jupyter notebooks

Example questions you can (start to) 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?
- How can I zoom in and visualize the interactions happening in the cell at the molecular level?
- How do structural properties correlate with my experimental datasets?
- How can I improve the contents of my model with structural data?
- 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. Most of these additional programs are used to predict or calculate properties of proteins.

CHAPTER 3

Tutorials

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

CHAPTER 4

Citation

The manuscript for the `ssbio` package can be found and cited at[1].

[1] Mih, N. et al. ssbio: A Python Framework for Structural Systems Biology. bioRxiv 165506 (2017). doi:10.1101/165506

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 0x7f89f05bd0f0>
```

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

```
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

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
         2zya    Dimeric 6-phosphogluconate dehydrogenase compl...
```

```
2zyd    Dimeric 6-phosphogluconate dehydrogenase compl...

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

       mapped_chains  resolution chemicals     taxonomy_name structure_file
pdb_id
2zya             A;B         1.6       6PG  Escherichia coli       2zya.cif
2zyd             A;B         1.5       GLO  Escherichia coli       2zyd.cif
```

```
In [11]: # Set representative structures
         my_protein.set_representative_structure(engine='biopython', force_rerun=True)
```

```
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.protein.sequence.utils.alignment] WARNING: Gap penalties not implemented in
[2017-09-04 17:50] [ssbio.core.protein] WARNING: SRR1753782_00918: no structures meet quality checks
```

Loading and aligning new sequences

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

Methods

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

         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 = 'MSKSQIGVVGMAVMGRPLALNIESRGYTVSVFNRSREKTEEVIAENPGKKLVPYYTVKEFVESLETPI

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

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

```
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	notes
0	Domain	179	467	Note=6PGD;Ontology_term=ECO:0000259;evidence=E...
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	

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

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()
```

```
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

```
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_C
```

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
        import tempfile
        ROOT_DIR = tempfile.gettempdir()

        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
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: /tmp/ssbio_protein_properties: GEM-PRO project locat
[2017-09-20 01:07] [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()
```

```
[2017-09-20 01:07] [root] INFO: getUserAgent: Begin
[2017-09-20 01:07] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-09-20 01:07] [root] INFO: getUserAgent: End
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 2/2: number of genes mapped to UniProt
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro
```

```
Missing UniProt mapping:  []
```

```
Out[8]: uniprot  reviewed gene_name                 kegg  \
        gene
        b0118  P36683     False      acnB  ecj:JW0114;eco:b0118
        b1276  P25516     False      acnA  ecj:JW1268;eco:b1276

                                refseq  pdbs                     pfam  \
        gene
        b0118  NP_414660.1;WP_001307570.1  1L5J  PF00330;PF06434;PF11791
        b1276  NP_415792.1;WP_000099535.1   NaN          PF00330;PF00694

                        description  entry_date  entry_version    seq_date  \
        gene
        b0118  Aconitate hydratase B  2017-08-30            162  1997-11-01
        b1276  Aconitate hydratase A  2017-08-30            150  2008-01-15

               seq_version sequence_file metadata_file
        gene
        b0118            3  P36683.fasta    P36683.xml
        b1276            3  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()
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative sequence
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for
```

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

```
Out[9]: uniprot                 kegg  pdbs sequence_file metadata_file
        gene
        b0118  P36683  ecj:JW0114;eco:b0118  1L5J  P36683.fasta    P36683.xml
        b1276  P25516  ecj:JW1268;eco:b1276   NaN  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()
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-09-20 01:07] [root] INFO: getUserAgent: Begin
[2017-09-20 01:07] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-09-20 01:07] [root] INFO: getUserAgent: End

[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 1/2: number of genes with at least one experimental
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Completed UniProt --> best PDB mapping. See the "df_
```

Out[10]:

gene	pdb_id	pdb_chain_id	uniprot	experimental_method	resolution	coverage	start	end	unp_start	unp_end	rank
b0118	1l5j	A	P36683	X-ray diffraction	2.4	1	1	865	1	865	1
b0118	1l5j	B	P36683	X-ray diffraction	2.4	1	1	865	1	865	2

In [11]:
```
# Mapping using BLAST
my_gempro.blast_seqs_to_pdb(all_genes=True, seq_ident_cutoff=.7, evalue=0.00001)
my_gempro.df_pdb_blast.head(2)
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 0: number of genes with additional structures added
[2017-09-20 01:07] [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)
```

```
[2017-09-20 01:07] [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)
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Updated homology model information for 2 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 [15]: # Download all mapped PDBs and gather the metadata
         my_gempro.pdb_downloader_and_metadata()
         my_gempro.df_pdb_metadata.head(2)
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: Saved 1 structures total
```

```
Out[15]: 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     taxonomy_name structure_file
         gene
         b0118         2.4   F3S;TRA  Escherichia coli       1l5j.pdb
```

```
In [16]: # Set representative structures
         my_gempro.set_representative_structure()
         my_gempro.df_representative_structures.head()
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[16]: id  is_experimental file_type  \
         gene
         b0118         REP-1l5j             True       pdb
         b1276  REP-ACON1_ECOLI            False       pdb

                                  structure_file
         gene
         b0118                       1l5j-A_clean.pdb
         b1276  ACON1_ECOLI_model1_clean-X_clean.pdb
```

Computing sequence and structure properties

```
In [17]: # 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 [18]: # 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/
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: /tmp/ssbio_protein_properties/data/ssbio_protein_pro
```

```
[2017-09-20 01:07] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with SCRATCH predictions loaded
```

```
In [19]: my_gempro.get_disulfide_bridges()
```

```
In [20]: # Requires DSSP installation
         # See the ssbio wiki for more information: https://github.com/SBRG/ssbio/wiki/Software-Insta
         my_gempro.get_dssp_annotations()
```

```
In [21]: # 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 [22]: from Bio.SeqFeature import SeqFeature, FeatureLocation
```

Looking at one protein

```
In [23]: my_protein = my_gempro.genes.b0118.protein
```

```
In [24]: # Here is a feature from UniProt
         a_feature = my_protein.representative_sequence.features[11]
         a_feature
```

```
Out[24]: SeqFeature(FeatureLocation(ExactPosition(1), ExactPosition(14)), type='helix')
```

```
In [25]: for f_resnum in a_feature:
             my_protein.get_residue_annotations(f_resnum, use_representatives=True)
```

```
Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 25,
          'seq_SS-sspro': 'C',
          'seq_SS-sspro8': 'C',
          'seq_residue': 'M',
          'seq_resnum': 1,
          'struct_ASA-dssp': 57.00000000000001,
          'struct_CA_DEPTH-msms': 1.99948489059817,
```

```
          'struct_PHI-dssp': 360.0,
          'struct_PSI-dssp': -50.0,
          'struct_RES_DEPTH-msms': 2.3006200496065223,
          'struct_RSA-dssp': 0.3031914893617021,
          'struct_SS-dssp': '-',
          'struct_residue': 'M',
          'struct_resnum': 1}

Out[25]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 10,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'L',
          'seq_resnum': 2,
          'struct_ASA-dssp': 17.0,
          'struct_CA_DEPTH-msms': 2.3599760610113107,
          'struct_PHI-dssp': -50.4,
          'struct_PSI-dssp': -49.5,
          'struct_RES_DEPTH-msms': 2.251669422562806,
          'struct_RSA-dssp': 0.10365853658536583,
          'struct_SS-dssp': 'H',
          'struct_residue': 'L',
          'struct_resnum': 2}

Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 75,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'E',
          'seq_resnum': 3,
          'struct_ASA-dssp': 169.0,
          'struct_CA_DEPTH-msms': 1.9998071344844712,
          'struct_PHI-dssp': -62.3,
          'struct_PSI-dssp': -51.4,
          'struct_RES_DEPTH-msms': 1.7084874299626664,
          'struct_RSA-dssp': 0.8711340206185567,
          'struct_SS-dssp': 'H',
          'struct_residue': 'E',
          'struct_resnum': 3}

Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 75,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'E',
          'seq_resnum': 4,
          'struct_ASA-dssp': 159.0,
          'struct_CA_DEPTH-msms': 1.9999139000634236,
          'struct_PHI-dssp': -63.0,
          'struct_PSI-dssp': -40.0,
          'struct_RES_DEPTH-msms': 1.7263995450061262,
          'struct_RSA-dssp': 0.8195876288659794,
          'struct_SS-dssp': 'H',
          'struct_residue': 'E',
          'struct_resnum': 4}

Out[25]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 0,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'Y',
```

```
          'seq_resnum': 5,
          'struct_ASA-dssp': 6.0,
          'struct_CA_DEPTH-msms': 1.999848478689486,
          'struct_PHI-dssp': -61.3,
          'struct_PSI-dssp': -43.4,
          'struct_RES_DEPTH-msms': 2.3493110252243294,
          'struct_RSA-dssp': 0.027027027027032,
          'struct_SS-dssp': 'H',
          'struct_residue': 'Y',
          'struct_resnum': 5}

Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 35,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'R',
          'seq_resnum': 6,
          'struct_ASA-dssp': 93.0,
          'struct_CA_DEPTH-msms': 2.5040841578768798,
          'struct_PHI-dssp': -81.4,
          'struct_PSI-dssp': -17.7,
          'struct_RES_DEPTH-msms': 1.9430019321682137,
          'struct_RSA-dssp': 0.375,
          'struct_SS-dssp': 'H',
          'struct_residue': 'R',
          'struct_resnum': 6}

Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 65,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'K',
          'seq_resnum': 7,
          'struct_ASA-dssp': 151.0,
          'struct_CA_DEPTH-msms': 1.9997938559545687,
          'struct_PHI-dssp': -63.6,
          'struct_PSI-dssp': -40.2,
          'struct_RES_DEPTH-msms': 1.902182757987616,
          'struct_RSA-dssp': 0.7365853658536585,
          'struct_SS-dssp': 'H',
          'struct_residue': 'K',
          'struct_resnum': 7}

Out[25]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 25,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'H',
          'seq_resnum': 8,
          'struct_ASA-dssp': 50.0,
          'struct_CA_DEPTH-msms': 1.9997835567091418,
          'struct_PHI-dssp': -63.3,
          'struct_PSI-dssp': -43.3,
          'struct_RES_DEPTH-msms': 2.010413265831258,
          'struct_RSA-dssp': 0.2717391304347826,
          'struct_SS-dssp': 'H',
          'struct_residue': 'H',
          'struct_resnum': 8}

Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 30,
```

```
           'seq_SS-sspro': 'H',
           'seq_SS-sspro8': 'H',
           'seq_residue': 'V',
           'seq_resnum': 9,
           'struct_ASA-dssp': 48.0,
           'struct_CA_DEPTH-msms': 2.5094559198025883,
           'struct_PHI-dssp': -58.1,
           'struct_PSI-dssp': -44.7,
           'struct_RES_DEPTH-msms': 2.3061489061311873,
           'struct_RSA-dssp': 0.3380281690140845,
           'struct_SS-dssp': 'H',
           'struct_residue': 'V',
           'struct_resnum': 9}
Out[25]: {'seq_RSA-accpro': 'e',
           'seq_RSA-accpro20': 55,
           'seq_SS-sspro': 'H',
           'seq_SS-sspro8': 'H',
           'seq_residue': 'A',
           'seq_resnum': 10,
           'struct_ASA-dssp': 59.0,
           'struct_CA_DEPTH-msms': 1.9996920575010475,
           'struct_PHI-dssp': -67.7,
           'struct_PSI-dssp': -42.4,
           'struct_RES_DEPTH-msms': 1.8414853111827647,
           'struct_RSA-dssp': 0.5566037735849056,
           'struct_SS-dssp': 'H',
           'struct_residue': 'A',
           'struct_resnum': 10}
Out[25]: {'seq_RSA-accpro': 'e',
           'seq_RSA-accpro20': 65,
           'seq_SS-sspro': 'H',
           'seq_SS-sspro8': 'H',
           'seq_residue': 'E',
           'seq_resnum': 11,
           'struct_ASA-dssp': 125.0,
           'struct_CA_DEPTH-msms': 1.999389970046125,
           'struct_PHI-dssp': -57.2,
           'struct_PSI-dssp': -43.4,
           'struct_RES_DEPTH-msms': 1.7213727717369471,
           'struct_RSA-dssp': 0.6443298969072165,
           'struct_SS-dssp': 'H',
           'struct_residue': 'E',
           'struct_resnum': 11}
Out[25]: {'seq_RSA-accpro': '-',
           'seq_RSA-accpro20': 5,
           'seq_SS-sspro': 'H',
           'seq_SS-sspro8': 'H',
           'seq_residue': 'R',
           'seq_resnum': 12,
           'struct_ASA-dssp': 20.0,
           'struct_CA_DEPTH-msms': 2.000033003939616,
           'struct_PHI-dssp': -76.1,
           'struct_PSI-dssp': -38.3,
           'struct_RES_DEPTH-msms': 2.245048277244616,
           'struct_RSA-dssp': 0.08064516129032258,
           'struct_SS-dssp': 'H',
           'struct_residue': 'R',
```

```
          'struct_resnum': 12}
```

```
Out[25]: {'seq_RSA-accpro': 'e',
          'seq_RSA-accpro20': 55,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'H',
          'seq_residue': 'A',
          'seq_resnum': 13,
          'struct_ASA-dssp': 64.0,
          'struct_CA_DEPTH-msms': 1.99947401906432,
          'struct_PHI-dssp': -64.8,
          'struct_PSI-dssp': -33.2,
          'struct_RES_DEPTH-msms': 1.7845496230695679,
          'struct_RSA-dssp': 0.6037735849056604,
          'struct_SS-dssp': 'H',
          'struct_residue': 'A',
          'struct_resnum': 13}
```

```
In [40]: for f in my_protein.representative_sequence.features:
             if 'metal' in f.type.lower():
                 print(f)
                 my_protein.get_residue_annotations(f.location.end, use_representatives=True)
                 print('**********************************************************************')
```

```
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
```

```
Out[40]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 10,
          'seq_SS-sspro': 'C',
          'seq_SS-sspro8': 'T',
          'seq_residue': 'C',
          'seq_resnum': ExactPosition(710),
          'struct_ASA-dssp': 16.0,
          'struct_CA_DEPTH-msms': 10.148959936792412,
          'struct_PHI-dssp': -67.1,
          'struct_PSI-dssp': -7.2,
          'struct_RES_DEPTH-msms': 10.009108769044623,
          'struct_RSA-dssp': 0.11851851851851852,
          'struct_SS-dssp': 'T',
          'struct_residue': 'C',
          'struct_resnum': 710}
```

```
**********************************************************************
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
```

```
Out[40]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 5,
          'seq_SS-sspro': 'C',
          'seq_SS-sspro8': 'C',
```

```
          'seq_residue': 'C',
          'seq_resnum': ExactPosition(769),
          'struct_ASA-dssp': 12.0,
          'struct_CA_DEPTH-msms': 8.296585114953835,
          'struct_PHI-dssp': -67.8,
          'struct_PSI-dssp': -28.3,
          'struct_RES_DEPTH-msms': 8.049832103234303,
          'struct_RSA-dssp': 0.08888888888888889,
          'struct_SS-dssp': '-',
          'struct_residue': 'C',
          'struct_resnum': 769}

**********************************************************************
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


Out[40]: {'seq_RSA-accpro': '-',
          'seq_RSA-accpro20': 5,
          'seq_SS-sspro': 'H',
          'seq_SS-sspro8': 'G',
          'seq_residue': 'C',
          'seq_resnum': ExactPosition(772),
          'struct_ASA-dssp': 11.0,
          'struct_CA_DEPTH-msms': 8.282291996377394,
          'struct_PHI-dssp': -50.2,
          'struct_PSI-dssp': -38.0,
          'struct_RES_DEPTH-msms': 8.23936884323092,
          'struct_RSA-dssp': 0.08148148148148149,
          'struct_SS-dssp': 'G',
          'struct_residue': 'C',
          'struct_resnum': 772}

**********************************************************************

In [34]: # 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

**Residue-level properties calculated from structure:
S
'structure_resnums': [434], 'CA_DEPTH-msms': [1.999802354873969], 'PHI-dssp': [93.2], 'SS-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

**Residue-level properties calculated from structure:
G
```

```
'structure_resnums': [500], 'CA_DEPTH-msms': [4.872765262426097], 'PHI-dssp': [-171.3], 'SS-dssp': [
--------------------------------
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

**Residue-level properties calculated from structure:
T
'structure_resnums': [503], 'CA_DEPTH-msms': [3.1546567153990472], 'PHI-dssp': [-64.7], 'SS-dssp': [
--------------------------------
********************************

********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

**Residue-level properties calculated from structure:
S
'structure_resnums': [709], 'CA_DEPTH-msms': [10.02414716892187], 'PHI-dssp': [-109.6], 'SS-dssp': [
--------------------------------
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

**Residue-level properties calculated from structure:
G
'structure_resnums': [768], 'CA_DEPTH-msms': [6.680980525988339], 'PHI-dssp': [158.5], 'SS-dssp': ['S
--------------------------------
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

**Residue-level properties calculated from structure:
L
```

```
'structure_resnums': [771], 'CA_DEPTH-msms': [6.8059331457785985], 'PHI-dssp': [-52.6], 'SS-dssp': [
--------------------------------
*******************************
```

```
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
1l5j
```

```
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
```

```
115j
[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

115j
[8.2393688432309204]
ACON2_ECOLI
[8.2393688432309204]
E00113
[8.2393688432309204]
```

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
```

```
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: /tmp/genes_and_sequences_GP: GEM-PRO project locatio
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: Loaded in 2 sequences
[2017-08-29 14:06] [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)
```

```
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-08-29 14:06] [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   4rjy            D     1696.0         0.0             0.993994
        b0870   4rjy            B     1696.0         0.0             0.993994

               hit_percent_ident  hit_num_ident  hit_num_similar
        gene
        b0870           0.987988            329              331
        b0870           0.987988            329              331
```

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

```
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: Saved 11 structures total
```

```
Out[9]: pdb_id                                          pdb_title  \
        gene
        b0870   3wlx  Crystal structure of low-specificity L-threoni...
        b0870   4lnj  Structure of Escherichia coli Threonine Aldola...

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

              mapped_chains  resolution   chemicals     taxonomy_name structure_file
        gene
        b0870           A;B        2.51         PLG  Escherichia coli       3wlx.cif
        b0870           A;B        2.10  EPE;MG;PLR  Escherichia coli       4lnj.cif
```

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

```
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: 2/2: number of genes with a representative structure
[2017-08-29 14:06] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[10]: id  is_experimental file_type    structure_file
         gene
         b0870  3wlx-A             True       pdb  3wlx-A_clean.pdb
         b3041  1iez-A             True       pdb  1iez-A_clean.pdb
```

```
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 0x7f468f1dd470>
```

```
Out[11]: {'_structure_dir': '/tmp/genes_and_sequences_GP/genes/b0870/b0870_protein/structures',
          'chains': [<ChainProp A at 0x7f46871dd6a0>],
          'date': None,
          'description': 'Low specificity L-threonine aldolase (E.C.4.1.2.48)',
          'file_type': 'pdb',
          'id': '3wlx-A',
          'is_experimental': True,
          'mapped_chains': ['A'],
          'notes': {},
          'original_pdb_id': '3wlx',
          '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
```

```
[2017-08-29 14:07] [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
```

```
[2017-08-29 14:07] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-08-29 14:07] [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)
```

```
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: /tmp/genes_GP: GEM-PRO project location
[2017-08-29 14:07] [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()
```

```
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to KEGG
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> KEGG. See the "df_kegg_meta
```

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

Out[8]:

gene	kegg	refseq	uniprot	num_pdbs	pdbs	seq_len	sequence_file	metadata_file
b0761	eco:b0761	NP_415282	P0A9G8	5	1B9M;1H9S;1B9N;1O7L;1H9R	262	eco-b0761.faa	eco-b0761.kegg
b0889	eco:b0889	NP_415409	P0ACJ0	2	2GQQ;2L4A	164	eco-b0889.faa	eco-b0889.kegg
b0995	eco:b0995	NP_415515	P38684	1	1ZGZ	230	eco-b0995.faa	eco-b0995.kegg
b1013	eco:b1013	NP_415533	P0ACU2	4	4JYK;4XK4;4X1E;3LOC	212	eco-b1013.faa	eco-b1013.kegg
b1014	eco:b1014	NP_415534	P09546	16	3E2Q;4JNZ;3E2R;4JNY;2GPE;4O8A;3E2S;2FZN;1TJ1;1...	1320	eco-b1014.faa	eco-b1014.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()
```

```
[2017-08-29 14:07] [root] INFO: getUserAgent: Begin
[2017-08-29 14:07] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-08-29 14:07] [root] INFO: getUserAgent: End
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: 10/10: number of genes mapped to UniProt
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: Completed ID mapping --> UniProt. See the "df_unipro
```

```
Missing UniProt mapping:  []
```

Out[9]:

gene	uniprot	reviewed	gene_name	kegg	refseq	num_pdbs
b0761	P0A9G8	False	modE	ecj:JW0744;eco:b0761	NP_415282.1;WP_001147439.1	5
b0889	P0ACJ0	False	lrp	ecj:JW0872;eco:b0889	NP_415409.1;WP_000228473.1	2
b0995	P38684	False	torR	ecj:JW0980;eco:b0995		
b1013	P0ACU2	False	rutR	ecj:JW0998;eco:b1013		
b1014	P09546	False	putA	ecj:JW0999;eco:b1014		

```
b0995  NP_415515.1;WP_001120125.1         1
b1013  NP_415533.1;WP_000191701.1         4
b1014  NP_415534.1;WP_001326840.1        16

                                                    pdbs  \
gene
b0761                          1B9M;1B9N;1H9R;1H9S;1O7L
b0889                                       2GQQ;2L4A
b0995                                            1ZGZ
b1013                                 3LOC;4JYK;4X1E;4XK4
b1014  1TIW;1TJ0;1TJ1;1TJ2;2AY0;2FZM;2FZN;2GPE;2RBF;3...

                         pfam  seq_len  \
gene
b0761          PF00126;PF03459      262
b0889                   PF01037      164
b0995          PF00072;PF00486      230
b1013          PF00440;PF08362      212
b1014  PF00171;PF01619;PF14850     1320

                                            description  entry_date  \
gene
b0761                     Transcriptional regulator ModE  2017-07-05
b0889               Leucine-responsive regulatory protein  2017-07-05
b0995  TorCAD operon transcriptional regulatory prote...  2017-07-05
b1013            HTH-type transcriptional regulator RutR  2017-07-05
b1014                          Bifunctional protein PutA  2017-07-05

       entry_version    seq_date  seq_version sequence_file metadata_file
gene
b0761            104  2005-07-19            1  P0A9G8.fasta    P0A9G8.xml
b0889            104  2007-01-23            2  P0ACJ0.fasta    P0ACJ0.xml
b0995            146  1997-11-01            2  P38684.fasta    P38684.xml
b1013             98  2005-11-22            1  P0ACU2.fasta    P0ACU2.xml
b1014            177  1997-11-01            3  P09546.fasta    P09546.xml
```

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()
```

```
Widget Javascript not detected.  It may not be installed or enabled properly.

[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: 10/10: number of genes with a representative sequenc
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: See the "df_representative_sequences" attribute for


Missing a representative sequence:  []
```

Out[10]:
```
       uniprot                 kegg  num_pdbs  \
gene
b0761  P0A9G8  ecj:JW0744;eco:b0761         5
b0889  P0ACJ0  ecj:JW0872;eco:b0889         2
b0995  P38684  ecj:JW0980;eco:b0995         1
b1013  P0ACU2  ecj:JW0998;eco:b1013         4
b1014  P09546  ecj:JW0999;eco:b1014        16

                                                  pdbs  seq_len  \
gene
b0761                          1B9M;1B9N;1H9R;1H9S;1O7L      262
b0889                                       2GQQ;2L4A      164
```

```
b0995                                                 1ZGZ      230
b1013                                  3LOC;4JYK;4X1E;4XK4      212
b1014  1TIW;1TJ0;1TJ1;1TJ2;2AY0;2FZM;2FZN;2GPE;2RBF;3...     1320

      sequence_file metadata_file
gene
b0761  P0A9G8.fasta    P0A9G8.xml
b0889  P0ACJ0.fasta    P0ACJ0.xml
b0995  P38684.fasta    P38684.xml
b1013  P0ACU2.fasta    P0ACU2.xml
b1014  P09546.fasta    P09546.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 [11]: # Mapping using the PDBe best_structures service
         my_gempro.map_uniprot_to_pdb(seq_ident_cutoff=.3)
         my_gempro.df_pdb_ranking.head()
```

```
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: Mapping UniProt IDs --> PDB IDs...
[2017-08-29 14:07] [root] INFO: getUserAgent: Begin
[2017-08-29 14:07] [root] INFO: getUserAgent: user_agent: EBI-Sample-Client/ (services.py; Python 3.5
[2017-08-29 14:07] [root] INFO: getUserAgent: End
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:07] [ssbio.pipeline.gempro] INFO: 8/10: number of genes with at least one experimental
[2017-08-29 14:07] [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
         b0761   1h9s            A  P0A9G8   X-ray diffraction        1.82     0.534
         b0761   1b9m            A  P0A9G8   X-ray diffraction        1.75     1.000
         b0761   1b9m            B  P0A9G8   X-ray diffraction        1.75     1.000
         b0761   1o7l            D  P0A9G8   X-ray diffraction        2.75     1.000
         b0761   1o7l            C  P0A9G8   X-ray diffraction        2.75     1.000

                start  end  unp_start  unp_end  rank
         gene
         b0761      1  140        123      262     9
         b0761      4  265          1      262     1
         b0761      4  265          1      262     2
```

```
b0761      1  262          1      262     8
b0761      1  262          1      262     7
```

```
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)
```

```
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:08] [ssbio.pipeline.gempro] INFO: Completed sequence --> PDB BLAST. See the "df_pdb_bl
[2017-08-29 14:08] [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  1.372630e-104             0.910377
         b1013   4x1e            B      966.0  1.372630e-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)
```

```
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:08] [ssbio.pipeline.gempro] INFO: Updated PDB metadata dataframe. See the "df_pdb_meta
[2017-08-29 14:08] [ssbio.pipeline.gempro] INFO: Saved 40 structures total
```

```
Out[13]: chemicals                       description experimental_method  \
         gene
         b0761  CA;CL;MOO     TRANSCRIPTIONAL REGULATOR MODE   X-ray diffraction
         b0761        MOO  MOLYBDENUM TRANSPORT PROTEIN MODE   X-ray diffraction

               mapped_chains pdb_id                                          pdb_title  \
         gene
         b0761       A;B;C;D   1o7l  Molybdate-activated form of ModE from Escheric...
         b0761           A;B   1h9s  Molybdate bound complex of Dimop domain of Mod...

                resolution structure_file                      taxonomy_name
         gene
         b0761        2.75       1o7l.cif                   ESCHERICHIA COLI
         b0761        1.82       1h9s.cif  ESCHERICHIA COLI;ESCHERICHIA COLI
```

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

```
Widget Javascript not detected.  It may not be installed or enabled properly.
```

```
[2017-08-29 14:09] [ssbio.core.protein] WARNING: b1014: no structures meet quality checks
[2017-08-29 14:09] [ssbio.core.protein] WARNING: b0995: no structures meet quality checks
[2017-08-29 14:09] [ssbio.core.protein] WARNING: b1130: no structures meet quality checks
[2017-08-29 14:09] [ssbio.pipeline.gempro] INFO: 5/10: number of genes with a representative structur
[2017-08-29 14:09] [ssbio.pipeline.gempro] INFO: See the "df_representative_structures" attribute for
```

```
Out[14]: id  is_experimental file_type    structure_file
         gene
         b0761  1b9m-A             True       pdb  1b9m-A_clean.pdb
         b0889  2gqq-A             True       pdb  2gqq-A_clean.pdb
         b1013  4jyk-A             True       pdb  4jyk-A_clean.pdb
         b1187  1hw1-A             True       pdb  1hw1-A_clean.pdb
         b1221  1a04-A             True       pdb  1a04-A_clean.pdb
```

```
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 0x7f35fed36cf8>
```

```
Out[15]: {'_structure_dir': '/tmp/genes_GP/genes/b1187/b1187_protein/structures',
          'chains': [<ChainProp A at 0x7f35fee7f748>],
          'date': None,
          'description': 'FATTY ACID METABOLISM REGULATOR PROTEIN',
          'file_type': 'pdb',
          'id': '1hw1-A',
          'is_experimental': True,
          'mapped_chains': ['A'],
          'notes': {},
          'original_pdb_id': '1hw1',
          '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 [16]: import os.path as op
         my_gempro.save_json(op.join(my_gempro.model_dir, '{}.json'.format(my_gempro.id)), compressio
```

```
[2017-08-29 14:09] [root] WARNING: json-tricks: numpy scalar serialization is experimental and may wo
[2017-08-29 14:09] [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()

        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 transforme
```

```
[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]:

gene	uniprot	kegg	num_pdbs	pdbs	seq_len	sequence_file
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]:

gene	pdb_id	pdb_chain_id	uniprot	experimental_method	resolution	coverage	\
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	

gene	start	end	unp_start	unp_end	rank
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 w
[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.protein.structure.structprop.**StructProp**(*ident*, *description=None*, *chains=None*, *mapped_chains=None*, *is_experimental=False*, *structure_path=None*, *file_type=None*)

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

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

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_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