pybel-tools Documentation

Release 0.7.0

Charles Tapley Hoyt

Sep 07, 2018

Getting Started

PyBEL-Tools is a suite of tools built on top of PyBEL to facilitate data management, integration, and analysis. For further examples, see the PyBEL-Notebooks repository. Installation is as easy as getting the code from PyPI with `python3 -m pip install pybel-tools`.

CHAPTER 1

Citation

If you use PyBEL and PyBEL Tools in your work, please cite[1]:

[1] Hoyt, C. T., *et al.* (2017). PyBEL: a Computational Framework for Biological Expression Language. Bioinformatics, 34(December), 1–2.

CHAPTER 2

Links

- Documented on Read the Docs
- Versioned on GitHub
- Tested on Travis CI
- Distributed by PyPI
- Chat on Gitter

2.1 Installation

PyBEL Tools is tested on Python3 installations on Mac OS and Linux on Travis CI.

Warning: Python2 and Windows are not thoroughly tested

2.1.1 Installation

Easiest

Download the latest stable code from PyPI with:

```
$ python3 -m pip install pybel_tools
```

Get the Latest

Download the most recent code from GitHub with:

```
$ python3 -m pip install git+https://github.com/pybel/pybel-tools.git@develop
```

For Developers

Clone the repository from GitHub and install in editable mode with:

```
$ git clone https://github.com/pybel/pybel-tools.git@develop
$ cd pybel-tools
$ python3 -m pip install -e .
```

2.1.2 Caveats

PyBEL Tools contains many dependencies, including the scientific Python Stack (numpy, scipy, etc.). This makes installation difficult for Windows users, for whom Python cannot easily build C extensions. We recommend using an Anaconda distribution of Python, which includes these precompiled.

2.2 Command Line Interface

2.2.1 Pickling lots of BEL scripts

All of the BEL scripts in the current working directory (and sub-directories) can be pickled in-place with the following command (add `-d` to specify a different directory)

```
$ pybel-tools io convert -d ~/bms/aetionomy/
```

2.2.2 Getting Data in to the Cache

Before running the service, some data can be pre-loaded in your cache.

Loading Selventa Corpra

The Selventa Small Corpus and Large Corpus are two example BEL documents distributed by the OpenBEL framework. They are good examples of many types of BEL statements and can be used immediately to begin exploring. Add `-v` for more logging information during compilation. This is highly suggested for the first run, since it takes a while to cache all of the namespaces and annotations. This only has to be done once, and will be much faster the second time!

Small Corpus

```
$ pybel-tools ensure small_corpus -v
```

Large Corpus

```
$ pybel-tools ensure large_corpus -v
```

Loading Other Resources

Gene Families

```
$ pybel-tools ensure gene_families -v
```

Named Protein Complexes

```
$ pybel-tools ensure named_complexes -v
```

Uploading Precompiled BEL

A single network stored as a PyBEL gpickle can quickly be uploaded using the following code:

```
$ pybel-tools io upload -p /path/to/my_network.gpickle
```

Uploading Multiple Networks

Multiple networks in a given directory and sub-directories can be uploaded by adding the `-r` tag.

```
$ pybel-tools io upload -p ~/bms/aetionomy/ -r
```

2.3 Summary

These scripts are designed to assist in the analysis of errors within BEL documents and provide some suggestions for fixes.

pybel_tools.summary.**count_relations**(*graph*)
Return a histogram over all relationships in a graph.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {relation type: frequency}

Return type collections.Counter

pybel_tools.summary.**get_edge_relations**(*graph*)
Builds a dictionary of {node pair: set of edge types}

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A dictionary of {(node, node): set of edge types}

Return type dict[tuple[tuple,tuple],set[str]]

pybel_tools.summary.**count_unique_relations**(*graph*)
Returns a histogram of the different types of relations present in a graph.

Note: this operation only counts each type of edge once for each pair of nodes

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns Counter from {relation type: frequency}

Return type collections.Counter

pybel_tools.summary.**count_annotations**(*graph*)
Counts how many times each annotation is used in the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {annotation key: frequency}

Return type collections.Counter

pybel_tools.summary.**get_annotations**(*graph*)
Gets the set of annotations used in the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of annotation keys

Return type set[str]

pybel_tools.summary.**get_annotations_containing_keyword**(*graph*, *keyword*)
Gets annotation/value pairs for values for whom the search string is a substring

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **keyword** (*str*) – Search for annotations whose values have this as a substring

Return type list[dict[str,str]

pybel_tools.summary.**count_annotation_values**(*graph*, *annotation*)
Counts in how many edges each annotation appears in a graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to count

Returns A Counter from {annotation value: frequency}

Return type collections.Counter

pybel_tools.summary.**count_annotation_values_filtered**(*graph*, *annotation*, *source_filter=None*, *target_filter=None*)
Counts in how many edges each annotation appears in a graph, but filter out source nodes and target nodes

See pybel_tools.utils.keep_node() for a basic filter.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to count
- **source_filter** (*types.FunctionType*) – A predicate (graph, node) -> bool for keeping source nodes
- **target_filter** (*types.FunctionType*) – A predicate (graph, node) -> bool for keeping target nodes

Returns A Counter from {annotation value: frequency}

Return type Counter

pybel_tools.summary.**pair_is_consistent**(*graph*, *u*, *v*)
Return if the edges between the given nodes are consistent, meaning they all have the same relation.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **u** (*tuple*) – The source BEL node
- **v** (*tuple*) – The target BEL node

Returns If the edges aren't consistent, return false, otherwise return the relation type

Return type bool or str

pybel_tools.summary.**get_consistent_edges**(*graph*)
Yields pairs of (source node, target node) for which all of their edges have the same type of relation.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterator over (source, target) node pairs corresponding to edges with many inconsistent relations

Return type iter[tuple]

pybel_tools.summary.**pair_has_contradiction**(*graph*, *u*, *v*)
Checks if a pair of nodes has any contradictions in their causal relationships.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **u** (*tuple*) – The source BEL node
- **v** (*tuple*) – The target BEL node

Returns Do the edges between these nodes have a contradiction?

Return type bool

pybel_tools.summary.**get_contradictory_pairs**(*graph*)
Iterates over contradictory node pairs in the graph based on their causal relationships

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterator over (source, target) node pairs that have contradictory causal edges

Return type iter

pybel_tools.summary.**count_pathologies**(*graph*)
Returns a counter of all of the mentions of pathologies in a network

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type Counter

pybel_tools.summary.**relation_set_has_contradictions**(*relations*)
Return if the set of relations contains a contradiction.

Parameters **relations** (*set[str]*) – A set of relations

Return type bool

pybel_tools.summary.**get_unused_annotations**(*graph*)
Gets the set of all annotations that are defined in a graph, but are never used.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of annotations

Return type set[str]

pybel_tools.summary.**get_unused_list_annotation_values**(*graph*)
Gets all of the unused values for list annotations

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A dictionary of {str annotation: set of str values that aren't used}

Return type dict[str, set[str]]

pybel_tools.summary.**count_error_types**(*graph*)
Counts the occurrence of each type of error in a graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter of {error type: frequency}

Return type collections.Counter

pybel_tools.summary.**count_naked_names**(*graph*)
Counts the frequency of each naked name (names without namespaces)

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {name: frequency}

Return type collections.Counter

pybel_tools.summary.**get_naked_names**(*graph*)
Gets the set of naked names in the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type set[str]

pybel_tools.summary.**get_incorrect_names_by_namespace**(*graph*, *namespace*)
Returns the set of all incorrect names from the given namespace in the graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespace** (*str*) – The namespace to filter by

Returns The set of all incorrect names from the given namespace in the graph

Return type set[str]

pybel_tools.summary.**get_incorrect_names**(*graph*)
Returns the dict of the sets of all incorrect names from the given namespace in the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The set of all incorrect names from the given namespace in the graph

Return type dict[str,set[str]]

pybel_tools.summary.**get_undefined_namespaces**(*graph*)
Gets all namespaces that aren't actually defined

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The set of all undefined namespaces

Return type set[str]

pybel_tools.summary.**get_undefined_namespace_names**(*graph*, *namespace*)
Gets the names from a namespace that wasn't actually defined

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespace** (*str*) – The namespace to filter by

Returns The set of all names from the undefined namespace

Return type set[str]

pybel_tools.summary.**calculate_incorrect_name_dict**(*graph*)
Groups all of the incorrect identifiers in a dict of {namespace: list of erroneous names}

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A dictionary of {namespace: list of erroneous names}

Return type dict[str, str]

pybel_tools.summary.**calculate_error_by_annotation**(*graph*, *annotation*)
Groups the graph by a given annotation and builds lists of errors for each

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group errors by

Returns A dictionary of {annotation value: list of errors}

Return type dict[str, list[str]]

pybel_tools.summary.**group_errors**(*graph*)
Groups the errors together for analysis of the most frequent error

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A dictionary of {error string: list of line numbers}

Return type dict[str, list[int]]

pybel_tools.summary.**get_names_including_errors**(*graph*)
Takes the names from the graph in a given namespace and the erroneous names from the same namespace and returns them together as a unioned set

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The dict of the sets of all correct and incorrect names from the given namespace in the graph

Return type dict[str,set[str]]

pybel_tools.summary.**get_names_including_errors_by_namespace**(*graph*, *namespace*)
Takes the names from the graph in a given namespace (pybel.struct.summary.get_names_by_namespace()) and the erroneous names from the same namespace (*get_incorrect_names_by_namespace()*) and returns them together as a unioned set

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespace** (*str*) – The namespace to filter by

Returns The set of all correct and incorrect names from the given namespace in the graph

Return type set[str]

pybel_tools.summary.**get_undefined_annotations**(*graph*)
Gets all annotations that aren't actually defined

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The set of all undefined annotations

Return type set[str]

pybel_tools.summary.**get_namespaces_with_incorrect_names**(*graph*)
Returns the set of all namespaces with incorrect names in the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type set[str]

pybel_tools.summary.**get_most_common_errors**(*graph*, *number=20*)
Gets the most common errors in a graph

Parameters

- **graph** (*pybel.BELGraph*) –
- **number** (*int*) –

Return type Counter

pybel_tools.summary.**plot_summary_axes**(*graph*, *lax*, *rax*, *logx=True*)
Plots your graph summary statistics on the given axes.

After, you should run `plt.tight_layout()` and you must run `plt.show()` to view.

Shows: 1. Count of nodes, grouped by function type 2. Count of edges, grouped by relation type

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **lax** – An axis object from matplotlib
- **rax** – An axis object from matplotlib

Example usage:

```
>>> import matplotlib.pyplot as plt
>>> from pybel import from_pickle
>>> from pybel_tools.summary import plot_summary_axes
>>> graph = from_pickle('~/dev/bms/aetionomy/parkinsons.gpickle')
>>> fig, axes = plt.subplots(1, 2, figsize=(10, 4))
>>> plot_summary_axes(graph, axes[0], axes[1])
>>> plt.tight_layout()
>>> plt.show()
```

pybel_tools.summary.**plot_summary**(*graph*, *plt*, *logx=True*, ***kwargs*)
Plots your graph summary statistics. This function is a thin wrapper around `plot_summary_axis()`. It automatically takes care of building figures given matplotlib's pyplot module as an argument. After, you need to run `plt.show()`.

`plt` is given as an argument to avoid needing matplotlib as a dependency for this function

Shows:

1. Count of nodes, grouped by function type
2. Count of edges, grouped by relation type

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **plt** – Give `matplotlib.pyplot` to this parameter
- **kwargs** – keyword arguments to give to `plt.subplots()`

Example usage:

```
>>> import matplotlib.pyplot as plt
>>> from pybel import from_pickle
>>> from pybel_tools.summary import plot_summary
>>> graph = from_pickle('~/dev/bms/aetionomy/parkinsons.gpickle')
>>> plot_summary(graph, plt, figsize=(10, 4))
>>> plt.show()
```

pybel_tools.summary.**info_list**(*graph*)
Returns useful information about the graph as a list of tuples

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type list

pybel_tools.summary.**info_str**(*graph*)
Puts useful information about the graph in a string

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type str

pybel_tools.summary.**info_json**(*graph*)
Returns useful information about the graph as a dictionary

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type dict

pybel_tools.summary.**print_summary**(*graph*, *file=None*)
Prints useful information about the graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **file** – A writeable file or file-like object. If None, defaults to `sys.stdout`

pybel_tools.summary.**is_causal_relation**(*data*)
Check if the given relation is causal.

Parameters **data** (*dict*) – The PyBEL edge data dictionary

Return type bool

pybel_tools.summary.**get_causal_out_edges**(*graph*, *nbunch*)
Gets the out-edges to the given node that are causal

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nbunch** (*tuple*) – A BEL node or iterable of BEL nodes

Returns A set of (source, target) pairs where the source is the given node

Return type set[tuple]

pybel_tools.summary.**get_causal_in_edges**(*graph*, *node*)
Gets the in-edges to the given node that are causal

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node

Returns A set of (source, target) pairs where the target is the given node

Return type set

pybel_tools.summary.**is_causal_source**(*graph*, *node*)
Return true of the node is a causal source.

- Doesn't have any causal in edge(s)
- Does have causal out edge(s)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node

Returns If the node is a causal source

Return type bool

pybel_tools.summary.**is_causal_central**(*graph*, *node*)
Return true if the node is neither a causal sink nor a causal source.

- Does have causal in edges(s)
- Does have causal out edge(s)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node

Returns If the node is neither a causal sink nor a causal source

Return type bool

pybel_tools.summary.**is_causal_sink**(*graph*, *node*)
Return true if the node is a causal sink.

- Does have causal in edge(s)
- Doesn't have any causal out edge(s)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node

Returns If the node is a causal source

Return type bool

pybel_tools.summary.**get_causal_source_nodes**(*graph*, *function*)
Returns a set of all nodes that have an in-degree of 0, which likely means that it is an external perturbagen and is not known to have any causal origin from within the biological system.

These nodes are useful to identify because they generally don't provide any mechanistic insight.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **function** (*str*) – The BEL function to filter by

Returns A set of source nodes

Return type set[tuple]

pybel_tools.summary.**get_causal_central_nodes**(*graph*, *function*)
Returns a set of all nodes that have both an in-degree > 0 and out-degree > 0. This means that they are an integral part of a pathway, since they are both produced and consumed.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **function** (*str*) – The BEL function to filter by

Returns A set of central ABUNDANCE nodes

Return type set

pybel_tools.summary.**get_causal_sink_nodes**(*graph*, *function*)
Returns a set of all ABUNDANCE nodes that have an causal out-degree of 0, which likely means that the knowledge assembly is incomplete, or there is a curation error.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **function** (*str*) – The BEL function to filter by

Returns A set of sink ABUNDANCE nodes

Return type set[tuple]

pybel_tools.summary.**get_degradations**(*graph*)
Gets all nodes that are degraded

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of nodes that are degraded

Return type set[tuple]

pybel_tools.summary.**get_activities**(*graph*)
Gets all nodes that have molecular activities

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of nodes that have molecular activities

Return type set[tuple]

pybel_tools.summary.**get_translocated**(*graph*)
Gets all nodes that are translocated

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of nodes that are translocated

Return type set[tuple]

pybel_tools.summary.**count_top_centrality**(*graph*, *number=30*)
Gets top centrality dictionary

Parameters

- **graph** –
- **number** (*int*) –

Return type dict[tuple,int]

pybel_tools.summary.**get_modifications_count**(*graph*)
Gets a modifications count dictionary

Parameters **graph** (*pybel.BELGraph*) –

Return type dict[str,int]

pybel_tools.summary.**count_subgraph_sizes**(*graph*, *annotation='Subgraph'*)
Counts the number of nodes in each subgraph induced by an anotation

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by and compare. Defaults to 'Subgraph'

Returns A dictionary from {annotation value: number of nodes}

Return type dict[str, int]

pybel_tools.summary.**calculate_subgraph_edge_overlap**(*graph*, *annotation='Subgraph'*)
Builds a dataframe to show the overlap between different subgraphs

Options: 1. Total number of edges overlap (intersection) 2. Percentage overlap (tanimoto similarity)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by and compare. Defaults to 'Subgraph'

Returns {subgraph: set of edges}, {(subgraph 1, subgraph2): set of intersecting edges}, {(subgraph 1, subgraph2): set of unioned edges}, {(subgraph 1, subgraph2): tanimoto similarity},

pybel_tools.summary.**summarize_subgraph_edge_overlap**(*graph*, *annotation='Subgraph'*)
Returns a similarity matrix between all subgraphs (or other given annotation)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by and compare. Defaults to `"Subgraph"`

Returns A similarity matrix in a dict of dicts

Return type dict

pybel_tools.summary.**rank_subgraph_by_node_filter**(*graph*, *node_filters*, *annotation='Subgraph'*, *reverse=True*)
Ranks subgraphs by which have the most nodes matching an given filter

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph

- **node_filters** (*types.FunctionType or iter[types.FunctionType]*) – A predicate or list of predicates (graph, node) -> bool
- **annotation** (*str*) –
- **reverse** (*bool*) –

Return type list

A use case for this function would be to identify which subgraphs contain the most differentially expressed genes.

```
>>> from pybel import from_pickle
>>> from pybel.constants import *
>>> from pybel_tools.integration import overlay_type_data
>>> from pybel_tools.summary import rank_subgraph_by_node_filter
>>> import pandas as pd
>>> graph = from_pickle('~/dev/bms/aetionomy/alzheimers.gpickle')
>>> df = pd.read_csv('~/dev/bananas/data/alzheimers_dgxp.csv', columns=['Gene',
↪'log2fc'])
>>> data = {gene: log2fc for _, gene, log2fc in df.itertuples()}
>>> overlay_type_data(graph, data, 'log2fc', GENE, 'HGNC', impute=0)
>>> results = rank_subgraph_by_node_filter(graph, lambda g, n: 1.3 < abs(g.
↪node[n]['log2fc']))
```

pybel_tools.summary.**summarize_subgraph_node_overlap**(*graph*, *node_filters=None*, *annotation='Subgraph'*)

Calculates the subgraph similarity tanimoto similarity in nodes passing the given filter

Provides an alternate view on subgraph similarity, from a more node-centric view

pybel_tools.summary.**count_pmids**(*graph*)

Counts the frequency of PubMed documents in a graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {(pmid, name): frequency}

Return type collections.Counter

pybel_tools.summary.**get_pmid_by_keyword**(*keyword*, *graph=None*, *pubmed_identifiers=None*)

Gets the set of PubMed identifiers beginning with the given keyword string

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **keyword** (*str*) – The beginning of a PubMed identifier
- **pubmed_identifiers** (*set[str]*) – A set of pre-cached PubMed identifiers

Returns A set of PubMed identifiers starting with the given string

Return type set[str]

pybel_tools.summary.**count_citations**(*graph*, ***annotations*)

Counts the citations in a graph based on a given filter

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotations** (*dict*) – The annotation filters to use

Returns A counter from {(citation type, citation reference): frequency}

Return type collections.Counter

pybel_tools.summary.**count_citations_by_annotation**(*graph*, *annotation*)
Groups the citation counters by subgraphs induced by the annotation

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to use to group the graph

Returns A dictionary of Counters {subgraph name: Counter from {citation: frequency}}

pybel_tools.summary.**count_authors**(*graph*)
Counts the contributions of each author to the given graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {author name: frequency}

Return type collections.Counter

pybel_tools.summary.**count_unique_authors**(*graph*)
Counts all authors in the given graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The number of unique authors whose publications contributed to the graph

Return type int

pybel_tools.summary.**count_author_publications**(*graph*)
Counts the number of publications of each author to the given graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter from {author name: frequency}

Return type collections.Counter

pybel_tools.summary.**count_unique_citations**(*graph*)
Returns the number of unique citations

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns The number of unique citations in the graph.

Return type int

pybel_tools.summary.**get_authors**(*graph*)
Gets the set of all authors in the given graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of author names

Return type set[str]

pybel_tools.summary.**get_authors_by_keyword**(*keyword*, *graph=None*, *authors=None*)
Gets authors for whom the search term is a substring

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **keyword** (*str*) – The keyword to search the author strings for
- **authors** (*set[str]*) – An optional set of pre-cached authors calculated from the graph

Returns A set of authors with the keyword as a substring

Return type set[str]

pybel_tools.summary.**count_authors_by_annotation**(*graph*, *annotation='Subgraph'*)
Groups the author counters by subgraphs induced by the annotation

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to use to group the graph

Returns A dictionary of Counters {subgraph name: Counter from {author: frequency}}

Return type dict

pybel_tools.summary.**get_evidences_by_pmid**(*graph*, *pmids*)
Gets a dictionary from the given PubMed identifiers to the sets of all evidence strings associated with each in the graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or iter[str] pmids** (*str*) – An iterable of PubMed identifiers, as strings. Is consumed and converted to a set.

Returns A dictionary of {pmid: set of all evidence strings}

Return type dict

pybel_tools.summary.**count_citation_years**(*graph*)
Counts the number of citations in each year

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A Counter of {int year: int frequency}

Return type collections.Counter

pybel_tools.summary.**create_timeline**(*year_counter*)
Completes the Counter timeline

Parameters **year_counter** (*Counter*) – counter dict for each year

Returns complete timeline

Return type list[tuple[int,int]]

pybel_tools.summary.**get_citation_years**(*graph*)
Creates a citation timeline counter

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type list[tuple[int,int]]

2.4 Filters

This module contains functions for filtering node and edge iterables. It relies heavily on the concepts of functional programming and the concept of predicates.

2.4.1 Node Filters

A node filter is a function that takes two arguments: a `pybel.BELGraph` and a node tuple. It returns a boolean representing whether the node passed the given test.

This module contains a set of default functions for filtering lists of nodes and building node filtering functions.

A general use for a node filter function is to use the built-in `filter()` in code like `filter(your_node_filter, graph)`

pybel_tools.filters.node_filters.**summarize_node_filter**(*graph*, *node_filters*)
Prints a summary of the number of nodes passing a given set of filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node_filters** (*types.FunctionType or iter[types.FunctionType]*) – A node filter or list/tuple of node filters

pybel_tools.filters.node_filters.**node_inclusion_filter_builder**(*nbunch*)
Builds a filter that only passes on nodes in the given list

Parameters **nbunch** (*iter[tuple]*) – An iterable of BEL nodes

Returns A node filter (graph, node) -> bool

Return type types.FunctionType

pybel_tools.filters.node_filters.**node_exclusion_filter_builder**(*nodes*)
Builds a filter that fails on nodes in the given list

Parameters **nodes** (*list*) – A list of nodes

Returns A node filter (graph, node) -> bool

Return type types.FunctionType

pybel_tools.filters.node_filters.**function_inclusion_filter_builder**(*func*)
Build a filter that only passes on nodes of the given function(s).

Parameters **func** (*str or iter[str]*) – A BEL Function or list/set/tuple of BEL functions

Returns A node predicate

Return type (pybel.BELGraph, tuple) -> bool

pybel_tools.filters.node_filters.**function_exclusion_filter_builder**(*func*)
Build a filter that fails on nodes of the given function(s).

Parameters **func** (*str or list[str] or tuple[str] or set[str]*) – A BEL Function or list/set/tuple of BEL functions

Returns A node predicate

Return type (pybel.BELGraph, tuple) -> bool

pybel_tools.filters.node_filters.**function_namespace_inclusion_builder**(*func*, *namespace*)
Build a filter function for matching the given BEL function with the given namespace or namespaces.

Parameters

- **func** (*str*) – A BEL function
- **or iter[str] namespace** (*str*) – The namespace to serach by

Returns A node predicate

Return type (pybel.BELGraph, tuple) -> bool

pybel_tools.filters.node_filters.**namespace_inclusion_builder**(*namespace*)
Build a predicate for namespace inclusion.

Parameters **or iter[str] namespace** (*str*) – A namespace or iter of namespaces

Returns A node predicate

Return type (pybel.BELGraph, tuple) -> bool

pybel_tools.filters.node_filters.**data_contains_key_builder**(*key*)
Build a filter that passes only on nodes that have the given key in their data dictionary.

Parameters **key** (*str*) – A key for the node's data dictionary

Returns A node predicate

Return type (pybel.BELGraph, tuple) -> bool

pybel_tools.filters.node_filters.**node_has_label**(*graph*, *node*)
Passes for nodes that have been annotated with a label

pybel_tools.filters.node_filters.**node_missing_label**(*graph*, *node*)
Fails for nodes that have been annotated with a label

pybel_tools.filters.node_filters.**include_pathology_filter**(*graph*, *node*)
A filter that passes for nodes that are pybel.constants.PATHOLOGY

pybel_tools.filters.node_filters.**exclude_pathology_filter**(*graph*, *node*)
A filter that fails for nodes that are pybel.constants.PATHOLOGY

pybel_tools.filters.node_filters.**data_missing_key_builder**(*key*)
Build a filter that passes only on nodes that don't have the given key in their data dictionary.

Parameters **key** (*str*) – A key for the node's data dictionary

Returns A node filter (graph, node) -> bool

Return type (pybel.BELGraph, BaseEntity) -> bool

pybel_tools.filters.node_filters.**build_node_data_search**(*key*, *data_predicate*)
Pass for nodes who have the given key in their data dictionaries and whose associated values pass the given filter function.

Parameters

- **key** (*str*) – The node data dictionary key to check
- **data_predicate** (*(Any) -> bool*) – The filter to apply to the node data dictionary

Returns A node predicate

Return type (pybel.BELGraph, BaseEntity) -> bool

pybel_tools.filters.node_filters.**build_node_key_search**(*query*, *key*)
Build a node filter that only passes for nodes whose values for the given key are superstrings of the query string(s).

Parameters

- **query** (*str or iter[str]*) – The query string or strings to check if they're in the node name

- **key** (*str*) – The key for the node data dictionary. Should refer only to entries that have str values

Returns A node predicate

Return type (pybel.BELGraph, BaseEntity) -> bool

2.4.2 Edge Filters

A edge filter is a function that takes five arguments: a `pybel.BELGraph`, a source node tuple, a target node tuple, a key, and a data dictionary. It returns a boolean representing whether the edge passed the given test.

This module contains a set of default functions for filtering lists of edges and building edge filtering functions.

A general use for an edge filter function is to use the built-in `filter()` in code like `filter(your_edge_filter, graph.edges_iter(keys=True, data=True))`

pybel_tools.filters.edge_filters.**summarize_edge_filter**(*graph*, *edge_filters*)
Prints a summary of the number of edges passing a given set of filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **edge_filters** (*(pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool or iter[(pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool]*) – A filter or list of filters

pybel_tools.filters.edge_filters.**build_edge_data_filter**(*annotations*, *partial_match=True*)
Build a filter that keeps edges whose data dictionaries are super-dictionaries to the given dictionary.

Parameters

- **annotations** (*dict*) – The annotation query dict to match
- **partial_match** (*bool*) – Should the query values be used as partial or exact matches? Defaults to `True`.

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_pmid_inclusion_filter**(*pmid*)
Pass for edges with citations whose references are one of the given PubMed identifiers.

Parameters **pmid** (*str or iter[str]*) – A PubMed identifier or list of PubMed identifiers to filter for

Returns An edge filter function (graph, node, node, key, data) -> bool

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_pmid_exclusion_filter**(*pmid*)
Fail for edges with citations whose references are one of the given PubMed identifiers.

Parameters **pmid** (*str or iter[str]*) – A PubMed identifier or list of PubMed identifiers to filter against

Returns An edge filter function (graph, node, node, key, data) -> bool

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_author_inclusion_filter**(*author*)
Only passes for edges with author information that matches one of the given authors

Parameters **author** (*str or iter[str]*) – The author or list of authors to filter by

Returns An edge filter

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_source_namespace_filter**(*namespaces*)
Only passes for edges whose source nodes have the given namespace or one of the given namespaces

Parameters **namespaces** (*str or iter[str]*) – The namespace or namespaces to filter by

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_target_namespace_filter**(*namespaces*)
Only passes for edges whose target nodes have the given namespace or one of the given namespaces

Parameters **namespaces** (*str or iter[str]*) – The namespace or namespaces to filter by

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_annotation_dict_all_filter**(*annotations*)
Build an edge predicate that passes for edges whose data dictionaries's annotations entry are super-dictionaries to the given dictionary.

If no annotations are given, will always evaluate to true.

Parameters **annotations** (*dict[str,iter[str]]*) – The annotation query dict to match

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**build_annotation_dict_any_filter**(*annotations*)
Build an edge predicate that passes for edges whose data dictionaries match the given dictionary.

If the given dictionary is empty, will always evaluate to true.

Parameters **annotations** (*dict[str,iter[str]]*) – The annotation query dict to match

Return type (pybel.BELGraph, BaseEntity, BaseEntity, str) -> bool

pybel_tools.filters.edge_filters.**has_pathology_causal**(*graph*, *u*, *v*, *k*)
Check if the subject is a pathology and has a causal relationship with a non bioprocess/pathology.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*BaseEntity*) – A BEL node
- **v** (*BaseEntity*) – A BEL node
- **k** (*str*) – The edge key between the given nodes

Returns If the subject of this edge is a pathology and it participates in a causal reaction.

Return type bool

2.5 Selection

This module contains functions to help select data from networks

pybel_tools.selection.**group_nodes_by_annotation**(*graph*, *annotation='Subgraph'*)
Groups the nodes occurring in edges by the given annotation

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – An annotation to use to group edges

Returns dict of sets of BELGraph nodes

Return type dict

pybel_tools.selection.**average_node_annotation**(*graph*, *key*, *annotation='Subgraph'*, *aggregator=None*)

Groups graph into subgraphs and assigns each subgraph a score based on the average of all nodes values for the given node key

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*str*) – The key in the node data dictionary representing the experimental data
- **annotation** (*str*) – A BEL annotation to use to group nodes
- **aggregator** (*lambda*) – A function from list of values -> aggregate value. Defaults to taking the average of a list of floats.

pybel_tools.selection.**group_nodes_by_annotation_filtered**(*graph*, *node_filters=None*, *annotation='Subgraph'*)

Groups the nodes occurring in edges by the given annotation, with a node filter applied

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node_filters** (*types.FunctionType or iter[types.FunctionType]*) – A predicate or list of predicates (graph, node) -> bool
- **annotation** – The annotation to use for grouping

Returns A dictionary of {annotation value: set of nodes}

Return type dict[str,set[tuple]]

pybel_tools.selection.**get_subgraph_by_induction**(*graph*, *nodes*)

Induce a sub-graph over the given nodes or return None if none of the nodes are in the given graph.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – A list of BEL nodes in the graph

Return type Optional[pybel.BELGraph]

pybel_tools.selection.**get_subgraph_by_node_filter**(*graph*, *node_filters*)

Induces a graph on the nodes that pass all filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node_filters** (*types.FunctionType or iter[types.FunctionType]*) – A node filter or list/tuple of node filters

Returns A subgraph induced over the nodes passing the given filters

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_neighborhood**(*graph*, *nodes*)

Get a BEL graph around the neighborhoods of the given nodes. Returns none if no nodes are in the graph.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – An iterable of BEL nodes

Returns A BEL graph induced around the neighborhoods of the given nodes

Return type Optional[pybel.BELGraph]

pybel_tools.selection.**get_subgraph_by_second_neighbors**(*graph*, *nodes*, *filter_pathologies=False*)

Get a graph around the neighborhoods of the given nodes and expand to the neighborhood of those nodes.

Returns none if none of the nodes are in the graph.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – An iterable of BEL nodes
- **filter_pathologies** (*bool*) – Should expansion take place around pathologies?

Returns A BEL graph induced around the neighborhoods of the given nodes

Return type Optional[pybel.BELGraph]

pybel_tools.selection.**get_subgraph_by_all_shortest_paths**(*graph*, *nodes*, *weight=None*, *remove_pathologies=False*)

Induce a subgraph over the nodes in the pairwise shortest paths between all of the nodes in the given list.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – A set of nodes over which to calculate shortest paths
- **weight** (*str*) – Edge data key corresponding to the edge weight. If None, performs unweighted search
- **remove_pathologies** (*bool*) – Should the pathology nodes be deleted before getting shortest paths?

Returns A BEL graph induced over the nodes appearing in the shortest paths between the given nodes

Return type Optional[pybel.BELGraph]

pybel_tools.selection.**get_subgraph_by_annotation_value**(*graph*, *annotation*, *values*)

Induce a sub-graph over all edges whose annotations match the given key and value.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to group by
- **values** (*str or iter[str]*) – The value(s) for the annotation

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_annotations**(*graph*, *annotations*, *or_=None*)

Induce a sub-graph given an annotations filter.

Parameters

- **graph** – pybel.BELGraph graph: A BEL graph

- **annotations** (*dict[str,iter[str]]*) – Annotation filters (match all with pybel.utils.subdict_matches())
- **or** (*boolean*) – if True any annotation should be present, if False all annotations should be present in the edge. Defaults to True.

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_pubmed**(*graph*, *pubmed_identifiers*)
Induce a sub-graph over the edges retrieved from the given PubMed identifier(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or list[str] pubmed_identifiers** (*str*) – A PubMed identifier or list of PubMed identifiers

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_authors**(*graph*, *authors*)
Induce a sub-graph over the edges retrieved publications by the given author(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or list[str] authors** (*str*) – An author or list of authors

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_node_search**(*graph*, *query*)
Gets a subgraph induced over all nodes matching the query string

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **or iter[str] query** (*str*) – A query string or iterable of query strings for node names

Returns A subgraph induced over the original BEL graph

Return type pybel.BELGraph

Thinly wraps *search_node_names()* and *get_subgraph_by_induction()*.

pybel_tools.selection.**get_causal_subgraph**(*graph*)
Builds a new subgraph induced over all edges that are causal

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph**(*graph*, *seed_method=None*, *seed_data=None*, *expand_nodes=None*, *remove_nodes=None*)
Run a pipeline query on graph with multiple sub-graph filters and expanders.

Order of Operations:

1. Seeding by given function name and data
2. Add nodes
3. Remove nodes

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **seed_method** (*str*) – The name of the get_subgraph_by_* function to use
- **seed_data** – The argument to pass to the get_subgraph function
- **expand_nodes** (*list[tuple]*) – Add the neighborhoods around all of these nodes
- **remove_nodes** (*list[tuple]*) – Remove these nodes and all of their in/out edges

Return type Optional[pybel.BELGraph]

pybel_tools.selection.**get_multi_causal_upstream**(*graph*, *nbunch*)

Get the union of all the 2-level deep causal upstream subgraphs from the nbunch.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nbunch** (*tuple or list[tuple]*) – A BEL node or list of BEL nodes

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_multi_causal_downstream**(*graph*, *nbunch*)

Get the union of all of the 2-level deep causal downstream subgraphs from the nbunch.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nbunch** (*tuple or list[tuple]*) – A BEL node or list of BEL nodes

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_random_subgraph**(*graph*, *number_edges=None*, *number_seed_edges=None*, *seed=None*, *invert_degrees=None*)

Generate a random subgraph based on weighted random walks from random seed edges.

Parameters

- **number_edges** (*Optional[int]*) – Maximum number of edges. Defaults to pybel_tools.constants.SAMPLE_RANDOM_EDGE_COUNT (250).
- **number_seed_edges** (*Optional[int]*) – Number of nodes to start with (which likely results in different components in large graphs). Defaults to SAMPLE_RANDOM_EDGE_SEED_COUNT (5).
- **seed** (*Optional[int]*) – A seed for the random state
- **invert_degrees** (*Optional[bool]*) – Should the degrees be inverted? Defaults to true.

Return type pybel.BELGraph

pybel_tools.selection.**get_leaves_by_type**(*graph*, *func=None*, *prune_threshold=1*)

Returns an iterable over all nodes in graph (in-place) with only a connection to one node. Useful for gene and RNA. Allows for optional filter by function type.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **func** (*str*) – If set, filters by the node's function from `pybel.constants` like `pybel.constants.GENE`, `pybel.constants.RNA`, `pybel.constants.PROTEIN`, or `pybel.constants.BIOPROCESS`
- **prune_threshold** (*int*) – Removes nodes with less than or equal to this number of connections. Defaults to `1`

Returns An iterable over nodes with only a connection to one node

Return type iter[tuple]

pybel_tools.selection.**get_nodes_in_all_shortest_paths**(*graph*, *nodes*, *weight=None*, *remove_pathologies=False*)

Get a set of nodes in all shortest paths between the given nodes.

Thinly wraps `networkx.all_shortest_paths()`.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – The list of nodes to use to use to find all shortest paths
- **weight** (*Optional[str]*) – Edge data key corresponding to the edge weight. If none, uses unweighted search.
- **remove_pathologies** (*bool*) – Should pathology nodes be removed first?

Returns A set of nodes appearing in the shortest paths between nodes in the BEL graph

Return type set[tuple]

Note: This can be trivially parallelized using `networkx.single_source_shortest_path()`

---

pybel_tools.selection.**get_shortest_directed_path_between_subgraphs**(*graph*, *a*, *b*)

Calculate the shortest path that occurs between two disconnected subgraphs A and B going through nodes in the source graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **a** (*pybel.BELGraph*) – A subgraph of `graph`, disjoint from `b`
- **b** (*pybel.BELGraph*) – A subgraph of `graph`, disjoint from `a`

Returns A list of the shortest paths between the two subgraphs

Return type list

pybel_tools.selection.**get_shortest_undirected_path_between_subgraphs**(*graph*, *a*, *b*)

Get the shortest path between two disconnected subgraphs A and B, disregarding directionality of edges in graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **a** (*pybel.BELGraph*) – A subgraph of `graph`, disjoint from `b`
- **b** (*pybel.BELGraph*) – A subgraph of `graph`, disjoint from `a`

Returns A list of the shortest paths between the two subgraphs

Return type list

pybel_tools.selection.**search_node_names**(*graph*, *query*)
Search for nodes containing a given string(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **query** (*str or iter[str]*) – The search query

Returns An iterator over nodes whose names match the search query

Return type iter

Example:

```
>>> from pybel.examples import sialic_acid_graph
>>> from pybel_tools.selection import search_node_names
>>> list(search_node_names(sialic_acid_graph, 'CD33'))
[('Protein', 'HGNC', 'CD33'), ('Protein', 'HGNC', 'CD33', ('pmod', ('bel', 'Ph
↪'')))]
```

pybel_tools.selection.**search_node_namespace_names**(*graph*, *query*, *namespace*)
Search for nodes with the given namespace(s) and whose names containing a given string(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **query** (*str or iter[str]*) – The search query
- **namespace** (*str or iter[str]*) – The namespace(s) to filter

Returns An iterator over nodes whose names match the search query

Return type iter

pybel_tools.selection.**search_node_hgnc_names**(*graph*, *query*)
Search for nodes with the HGNC namespace and whose names containing a given string(s).

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **query** (*str or iter[str]*) – The search query

Returns An iterator over nodes whose names match the search query

Return type iter

pybel_tools.selection.**convert_path_to_metapath**(*graph*, *nodes*)
Converts a list of nodes to their corresponding functions

Parameters **nodes** (*list[tuple]*) – A list of BEL node tuples

Return type list[str]

pybel_tools.selection.**get_walks_exhaustive**
Gets all walks under a given length starting at a given node

Parameters

- **graph** (*networkx.Graph*) – A graph
- **node** – Starting node
- **length** (*int*) – The length of walks to get

Returns A list of paths

Return type list[tuple]

pybel_tools.selection.**match_simple_metapath**(*graph*, *node*, *simple_metapath*)
Matches a simple metapath starting at the given node

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node
- **simple_metapath** (*list[str]*) – A list of BEL Functions

Returns An iterable over paths from the node matching the metapath

Return type iter[tuple]

2.6 Integration

This module contains functions that help add more data to the network

pybel_tools.integration.**overlay_data**(*graph*, *data*, *label=None*, *overwrite=False*)
Overlays tabular data on the network

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **data** (*dict*) – A dictionary of {tuple node: data for that node}
- **label** (*Optional[str]*) – The annotation label to put in the node dictionary
- **overwrite** (*bool*) – Should old annotations be overwritten?

pybel_tools.integration.**overlay_type_data**(*graph*, *data*, *func*, *namespace*, *label=None*, *overwrite=False*, *impute=None*)
Overlay tabular data on the network for data that comes from an data set with identifiers that lack namespaces.

For example, if you want to overlay differential gene expression data from a table, that table probably has HGNC identifiers, but no specific annotations that they are in the HGNC namespace or that the entities to which they refer are RNA.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **dict data** (*dict[str,float]*) – A dictionary of {name: data}
- **func** (*str*) – The function of the keys in the data dictionary
- **namespace** (*str*) – The namespace of the keys in the data dictionary
- **label** (*Optional[str]*) – The annotation label to put in the node dictionary
- **overwrite** (*bool*) – Should old annotations be overwritten?
- **impute** (*Optional[float]*) – The value to use for missing data

pybel_tools.integration.**load_differential_gene_expression**(*path*, *gene_symbol_column='Gene.symbol'*, *logfc_column='logFC'*, *aggregator=None*)
Load and preprocess a differential gene expression data.

Parameters

- **path** (*str*) – The path to the CSV
- **gene_symbol_column** (*str*) – The header of the gene symbol column in the data frame
- **logfc_column** (*str*) – The header of the log-fold-change column in the data frame
- **aggregator** (*Optional[list[float] -> float]*) – A function that aggregates a list of differential gene expression values. Defaults to `numpy.median()`. Could also use: `numpy.mean()`, `numpy.average()`, `numpy.min()`, or `numpy.max()`

Returns A dictionary of {gene symbol: log fold change}

Return type dict[str,float]

2.7 Mutation

This module contains functions that mutate or make transformations on a network

pybel_tools.mutation.**collapse_nodes**(*graph*, *survivor_mapping*)
Collapse all nodes in values to the key nodes, in place.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **survivor_mapping** (*dict[tuple, set[tuple]]*) – A dictionary with survivors as their keys, and iterables of the corresponding victims as values.

pybel_tools.mutation.**rewire_variants_to_genes**(*graph*)
Finds all protein variants that are pointing to a gene and not a protein and fixes them by changing their function to be `pybel.constants.GENE`, in place

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

A use case is after running `collapse_to_genes()`.

pybel_tools.mutation.**collapse_gene_variants**(*graph*)
Collapses all gene's variants' edges to their parents, in-place

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

pybel_tools.mutation.**collapse_protein_variants**(*graph*)
Collapses all protein's variants' edges to their parents, in-place

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

pybel_tools.mutation.**collapse_consistent_edges**(*graph*)
Collapse consistent edges together.

> **Warning:** This operation doesn't preserve evidences or other annotations

Parameters **graph** (*pybel.BELGraph*) – A BEL Graph

pybel_tools.mutation.**collapse_equivalencies_by_namespace**(*graph*, *victim_namespace*, *survivor_namespace*)
Collapse pairs of nodes with the given namespaces that have equivalence relationships.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or iter[str] victim_namespace** (*str*) – The namespace(s) of the node to collapse
- **survivor_namespace** (*str*) – The namespace of the node to keep

To convert all ChEBI names to InChI keys, assuming there are appropriate equivalence relations between nodes with those namespaces:

```
>>> collapse_equivalencies_by_namespace(graph, 'CHEBI', 'CHEBIID')
>>> collapse_equivalencies_by_namespace(graph, 'CHEBIID', 'INCHI')
```

pybel_tools.mutation.**collapse_orthologies_by_namespace**(*graph*, *victim_namespace*, *survivor_namespace*)

Collapse pairs of nodes with the given namespaces that have orthology relationships.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **or iter[str] victim_namespace** (*str*) – The namespace(s) of the node to collapse
- **survivor_namespace** (*str*) – The namespace of the node to keep

To collapse all MGI nodes to their HGNC orthologs, use: >>> collapse_orthologies_by_namespace('MGI', 'HGNC')

To collapse collapse both MGI and RGD nodes to their HGNC orthologs, use: >>> collapse_orthologies_by_namespace(['MGI', 'RGD'], 'HGNC')

pybel_tools.mutation.**collapse_to_protein_interactions**(*graph*)

Collapse to a graph made of only causal gene/protein edges.

Parameters **graph** (*pybel.BELGraph*) – A BEL Graph

Return type pybel.BELGraph

pybel_tools.mutation.**remove_inconsistent_edges**(*graph*)

Remove all edges between node paris with consistent edges.

This is the all-or-nothing approach. It would be better to do more careful investigation of the evidences during curation.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

pybel_tools.mutation.**get_peripheral_successor_edges**(*graph*, *subgraph*)

Gets the set of possible successor edges peripheral to the subgraph. The source nodes in this iterable are all inside the subgraph, while the targets are outside.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns An iterable of possible successor edges (4-tuples of node, node, key, data)

Return type iter[tuple]

pybel_tools.mutation.**get_peripheral_predecessor_edges**(*graph*, *subgraph*)

Gets the set of possible predecessor edges peripheral to the subgraph. The target nodes in this iterable are all inside the subgraph, while the sources are outside.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Return type iter[tuple]

pybel_tools.mutation.**count_sources**(*edge_iter*)

Counts the source nodes in an edge iterator with keys and data

Parameters **edge_iter** (*iter[tuple]*) – An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Returns A counter of source nodes in the iterable

Return type collections.Counter

pybel_tools.mutation.**count_targets**(*edge_iter*)

Counts the target nodes in an edge iterator with keys and data

Parameters **edge_iter** (*iter[tuple]*) – An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Returns A counter of target nodes in the iterable

Return type collections.Counter

pybel_tools.mutation.**count_possible_successors**(*graph*, *subgraph*)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns A counter of possible successor nodes

Return type collections.Counter

pybel_tools.mutation.**count_possible_predecessors**(*graph*, *subgraph*)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns A counter of possible predecessor nodes

Return type collections.Counter

pybel_tools.mutation.**get_subgraph_edges**(*graph*, *annotation*, *value*, *source_filter=None*, *target_filter=None*)

Gets all edges from a given subgraph whose source and target nodes pass all of the given filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to search
- **value** (*str*) – The annotation value to search by
- **source_filter** – Optional filter for source nodes (graph, node) -> bool
- **target_filter** – Optional filter for target nodes (graph, node) -> bool

Returns An iterable of (source node, target node, key, data) for all edges that match the annotation/value and node filters

Return type iter[tuple]

pybel_tools.mutation.**get_subgraph_peripheral_nodes**(*graph*, *subgraph*, *node_filters=None*, *edge_filters=None*)

Gets a summary dictionary of all peripheral nodes to a given subgraph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** (*iter[tuple]*) – A set of nodes
- **node_filters** (*lambda*) – Optional. A list of node filter predicates with the interface (graph, node) -> bool. See *pybel_tools.filters.node_filters* for more information
- **edge_filters** (*lambda*) – Optional. A list of edge filter predicates with the interface (graph, node, node, key, data) -> bool. See *pybel_tools.filters.edge_filters* for more information

Returns A dictionary of {external node: {'successor': {internal node: list of (key, dict)}, 'predecessor': {internal node: list of (key, dict)}}}

Return type dict

For example, it might be useful to quantify the number of predecessors and successors

```
>>> import pybel_tools as pbt
>>> sgn = 'Blood vessel dilation subgraph'
>>> sg = pbt.selection.get_subgraph_by_annotation_value(graph, annotation=
↪'Subgraph', value=sgn)
>>> p = pbt.mutation.get_subgraph_peripheral_nodes(graph, sg, node_filters=pbt.
↪filters.exclude_pathology_filter)
>>> for node in sorted(p, key=lambda n: len(set(p[n]['successor']) | set(p[n][
↪'predecessor'])), reverse=True):
>>>     if 1 == len(p[sgn][node]['successor']) or 1 == len(p[sgn][node][
↪'predecessor']):
>>>         continue
>>>     print(node,
>>>           len(p[node]['successor']),
>>>           len(p[node]['predecessor']),
>>>           len(set(p[node]['successor']) | set(p[node]['predecessor'])))
```

pybel_tools.mutation.**expand_periphery**(*universe*, *graph*, *node_filters=None*, *edge_filters=None*, *threshold=2*)

Iterates over all possible edges, peripheral to a given subgraph, that could be added from the given graph. Edges could be added if they go to nodes that are involved in relationships that occur with more than the threshold (default 2) number of nodes in the subgraph.

Parameters

- **universe** (*pybel.BELGraph*) – The universe of BEL knowledge
- **graph** (*pybel.BELGraph*) – The (sub)graph to expand
- **node_filters** (*lambda*) – Optional. A list of node filter predicates with the interface (graph, node) -> bool. See *pybel_tools.filters.node_filters* for more information

- **edge_filters** (*lambda*) – Optional. A list of edge filter predicates with the interface (graph, node, node, key, data) -> bool. See *pybel_tools.filters.edge_filters* for more information
- **threshold** – Minimum frequency of betweenness occurrence to add a gap node

A reasonable edge filter to use is `pybel_tools.filters.keep_causal_edges()` because this function can allow for huge expansions if there happen to be hub nodes.

pybel_tools.mutation.**enrich_grouping**(*universe*, *graph*, *function*, *relation*)

Adds all of the grouped elements. See *enrich_complexes()*, *enrich_composites()*, and *enrich_reactions()*

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich
- **function** (*str*) – The function by which the subject of each triple is filtered
- **relation** (*str*) – The relationship by which the predicate of each triple is filtered

pybel_tools.mutation.**enrich_complexes**(*universe*, *graph*)

Add all of the members of the complex abundances to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.**enrich_composites**(*universe*, *graph*)

Adds all of the members of the composite abundances to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.**enrich_reactions**(*universe*, *graph*)

Adds all of the reactants and products of reactions to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.**enrich_variants**(*universe*, *graph*, *func=None*)

Add the reference nodes for all variants of the given function.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich
- **or iter[str] func** (*str*) – The function by which the subject of each triple is filtered. Defaults to the set of protein, rna, mirna, and gene.

pybel_tools.mutation.**enrich_unqualified**(*universe*, *graph*)
Enrich the subgraph with the unqualified edges from the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

The reason you might want to do this is you induce a subgraph from the original graph based on an annotation filter, but the unqualified edges that don't have annotations that most likely connect elements within your graph are not included.

See also:

This function thinly wraps the successive application of the following functions:

- *enrich_complexes()*
- *enrich_composites()*
- *enrich_reactions()*
- *enrich_variants()*

Equivalent to:

```
>>> enrich_complexes(universe, graph)
>>> enrich_composites(universe, graph)
>>> enrich_reactions(universe, graph)
>>> enrich_variants(universe, graph)
```

pybel_tools.mutation.**expand_internal**(*universe*, *graph*, *edge_filters=None*)
Edges between entities in the subgraph that pass the given filters

Parameters

- **universe** (*pybel.BELGraph*) – The full graph
- **graph** (*pybel.BELGraph*) – A subgraph to find the upstream information
- **edge_filters** (*list or lambda*) – Optional list of edge filter functions (graph, node, node, key, data) -> bool

pybel_tools.mutation.**expand_internal_causal**(*universe*, *graph*)
Adds causal edges between entities in the subgraph.

Is an extremely thin wrapper around *expand_internal()*.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich with causal relations between contained nodes

Equivalent to:

```
>>> import pybel_tools as pbt
>>> from pybel.struct.filters.edge_predicates import is_causal_relation
>>> pbt.mutation.expand_internal(universe, graph, edge_filters=is_causal_relation)
```

pybel_tools.mutation.**is_node_highlighted**(*graph*, *node*)
Returns if the given node is highlighted.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node

Returns Does the node contain highlight information?

Return type bool

pybel_tools.mutation.**highlight_nodes**(*graph*, *nodes=None*, *color=None*)
Adds a highlight tag to the given nodes.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*iter[tuple]*) – The nodes to add a highlight tag on
- **color** (*str*) – The color to highlight (use something that works with CSS)

pybel_tools.mutation.**remove_highlight_nodes**(*graph*, *nodes=None*)
Removes the highlight from the given nodes, or all nodes if none given.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **nodes** (*list*) – The list of nodes to un-highlight

pybel_tools.mutation.**is_edge_highlighted**(*graph*, *u*, *v*, *k*, *d*)
Returns if the given edge is highlighted.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns Does the edge contain highlight information?

Return type bool

pybel_tools.mutation.**highlight_edges**(*graph*, *edges=None*, *color=None*)
Adds a highlight tag to the given edges.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **edges** (*iter[tuple]*) – The edges (4-tuples of u, v, k, d) to add a highlight tag on
- **color** (*str*) – The color to highlight (use something that works with CSS)

pybel_tools.mutation.**remove_highlight_edges**(*graph*, *edges=None*)
Removes the highlight from the given edges, or all edges if none given.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **edges** (*iter[tuple]*) – The edges (4-tuple of u,v,k,d) to remove the highlight from)

pybel_tools.mutation.**highlight_subgraph**(*universe*, *graph*)
Highlights all nodes/edges in the universe that in the given graph.

Parameters **universe** (*pybel.BELGraph*) – The universe of knowledge

pybel_tools.mutation.**remove_highlight_subgraph**(*graph*, *subgraph*)
Removes the highlight from all nodes/edges in the graph that are in the subgraph.

Parameters

- **graph** (*pybel.BELGraph*) – The BEL graph to mutate
- **subgraph** (*pybel.BELGraph*) – The subgraph from which to remove the highlighting

pybel_tools.mutation.**enrich_protein_and_rna_origins**(*graph*)
Add the corresponding RNA for each protein then the corresponding gene for each RNA/miRNA.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

pybel_tools.mutation.**infer_missing_two_way_edges**(*graph*)
Add edges to the graph when a two way edge exists, and the opposite direction doesn't exist.

Use: two way edges from BEL definition and/or axiomatic inverses of membership relations

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

pybel_tools.mutation.**infer_missing_backwards_edge**(*graph*, *u*, *v*, *k*)
Add the same edge, but in the opposite direction if not already present.

pybel_tools.mutation.**enrich_internal_unqualified_edges**(*graph*, *subgraph*)
Add the missing unqualified edges between entities in the subgraph that are contained within the full graph.

Parameters

- **graph** (*pybel.BELGraph*) – The full BEL graph
- **subgraph** (*pybel.BELGraph*) – The query BEL subgraph

pybel_tools.mutation.**parse_authors**(*graph*, *force_parse=False*)
Parses all of the citation author strings to lists by splitting on the pipe character "|"

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **force_parse** (*bool*) – Forces serialization without checking the tag

Returns A set of all authors in this graph

Return type set[str]

pybel_tools.mutation.**serialize_authors**(*graph*, *force_serialize=False*)
Recombines all authors with the pipe character "|".

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **force_serialize** (*bool*) – Forces serialization without checking the tag

pybel_tools.mutation.**enrich_pubmed_citations**(*graph*, *stringify_authors=False*, *manager=None*)
Overwrites all PubMed citations with values from NCBI's eUtils lookup service.

Sets authors as list, so probably a good idea to run *pybel_tools.mutation.serialize_authors()* before exporting.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **stringify_authors** (*bool*) – Converts all author lists to author strings using *pybel_tools.mutation.serialize_authors()*. Defaults to `False`.

- **manager** (*None or str or Manager*) – An RFC-1738 database connection string, a pre-built `pybel.manager.Manager`, or `None` for default connection

Returns A set of PMIDs for which the eUtils service crashed

Return type set[str]

`pybel_tools.mutation.`**`random_by_nodes`**(*graph*, *percentage=None*)
Gets a random graph by inducing over a percentage of the original nodes

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **percentage** (*float*) – The percentage of edges to keep

Return type pybel.BELGraph

`pybel_tools.mutation.`**`random_by_edges`**(*graph*, *percentage=None*)
Gets a random graph by keeping a certain percentage of original edges

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **percentage** (*float*) – The percentage of edges to keep

Return type pybel.BELGraph

`pybel_tools.mutation.`**`shuffle_node_data`**(*graph*, *key*, *percentage=None*)
Shuffles the node's data. Useful for permutation testing.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*str*) – The node data dictionary key
- **percentage** (*float*) – What percentage of possible swaps to make

Return type pybel.BELGraph

`pybel_tools.mutation.`**`shuffle_relations`**(*graph*, *percentage=None*)
Shuffles the relations. Useful for permutation testing.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **percentage** (*float*) – What percentage of possible swaps to make

Return type pybel.BELGraph

`pybel_tools.mutation.`**`build_expand_node_neighborhood_by_hash`**(*manager: pybel.manager.cache_manager.Manager*)
Make an expand function that's bound to the manager.

Return type (pybel.BELGraph, pybel.BELGraph, str) -> None

`pybel_tools.mutation.`**`build_delete_node_by_hash`**(*manager: pybel.manager.cache_manager.Manager*)
Make a delete function that's bound to the manager.

Return type (pybel.BELGraph, str) -> None

2.8 Visualization

A wrapper around PyBEL-Jupyter.

This functionality has been moved to: https://github.com/pybel/pybel-jupyter.

2.9 Query Builder

A wrapper around the PyBEL query builder.

exception pybel_tools.query.**QueryMissingNetworksError**
Raised if a query is created from json but doesn't have a listing of network identifiers.

class pybel_tools.query.**Query**(*network_ids=None*, *seeding=None*, *pipeline=None*)
Represents a query over a network store.

Build a query.

Parameters **network_ids** (*None or int or iter[int]*) – Database network identifiers identifiers

append_network(*network_id*)
Add a network to this query.

Parameters **network_id** (*int*) – The database identifier of the network

Returns self for fluid API

Return type *Query*

append_seeding_induction(*nodes*)
Add a seed induction method.

Parameters **or Node or BaseEntity] nodes** (*list[tuple*) – A list of PyBEL node tuples

Returns seeding container for fluid API

Return type Seeding

append_seeding_neighbors(*nodes*)
Add a seed by neighbors.

Parameters **nodes** (*BaseEntity or iter[BaseEntity]*) – A list of PyBEL node tuples

append_seeding_annotation(*annotation*, *values*)
Add a seed induction method for single annotation's values.

Parameters

- **annotation** (*str*) – The annotation to filter by
- **values** (*set[str]*) – The values of the annotation to keep

append_seeding_sample(***kwargs*)
Add seed induction methods.

Kwargs can have `number_edges` or `number_seed_nodes`.

append_pipeline(*name*, **args*, ***kwargs*)
Add an entry to the pipeline. Defers to `pybel_tools.pipeline.Pipeline.append()`.

Parameters **name** (*str or types.FunctionType*) – The name of the function

Returns This pipeline for fluid query building

Return type Pipeline

run(*manager*)
Run this query and returns the resulting BEL graph.

Parameters **manager** – A cache manager

Return type Optional[pybel.BELGraph]

to_json()
Return this query as a JSON object.

Return type dict

dump(*file*, ***kwargs*)
Dump this query to a file as JSON.

dumps(***kwargs*)
Dump this query to a string as JSON

Return type str

static from_json(*data*)
Load a query from a JSON dictionary.

Parameters **data** (*dict*) – A JSON dictionary

Return type *Query*

Raises QueryMissingNetworksError

static load(*file*)
Load a query from a JSON file.

Parameters **file** – A file or file-like

Return type *Query*

Raises QueryMissingNetworksError

static loads(*s*)
Load a query from a JSON string

Parameters **s** (*str*) – A stringified JSON query

Return type *Query*

Raises QueryMissingNetworksError

2.10 Stability Analysis

pybel_tools.analysis.stability.**get_contradiction_summary**(*graph*)
Yield triplets of (source node, target node, set of relations) for (source node, target node) pairs that have multiple, contradictory relations.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type iter[tuple]

pybel_tools.analysis.stability.**get_regulatory_pairs**(*graph*)
Finds pairs of nodes that have mutual causal edges that are regulating each other such that `A -> B` and `B -| A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of pairs of nodes with mutual causal edges

Return type set

pybel_tools.analysis.stability.**get_chaotic_pairs**(*graph*)
Finds pairs of nodes that have mutual causal edges that are increasing each other such that `A -> B` and `B -> A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of pairs of nodes with mutual causal edges

Return type set

pybel_tools.analysis.stability.**get_dampened_pairs**(*graph*)
Finds pairs of nodes that have mutual causal edges that are decreasing each other such that `A -| B` and `B -| A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns A set of pairs of nodes with mutual causal edges

Return type set

pybel_tools.analysis.stability.**get_correlation_graph**(*graph*)
Extracts a graph of only correlative relationships

Parameters **graph** (*pybel.BELGraph*) – A BEL Graph

Return type networkx.Graph

pybel_tools.analysis.stability.**get_correlation_triangles**(*graph*)
Returns a set of all triangles pointed by the given node

Parameters **graph** (*networkx.Graph*) – A non-directional graph

Return type set[tuple]

pybel_tools.analysis.stability.**get_triangles**(*graph*)
Gets a set of triples representing the 3-cycles from a directional graph. Each 3-cycle is returned once, with nodes in sorted order.

Parameters **graph** (*networkx.DiGraph*) – A directional graph

Return type set[tuple]

pybel_tools.analysis.stability.**get_separate_unstable_correlation_triples**(*graph*)
Yields all triples of nodes A, B, C such that `A positiveCorrelation B`, `A positiveCorrelation C`, and `B negativeCorrelation C`

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterator over triples of unstable graphs, where the second two are negative

Return type iter[tuple]

pybel_tools.analysis.stability.**get_mutually_unstable_correlation_triples**(*graph*)
Yields all triples of nodes A, B, C such that `A negativeCorrelation B`, `B negativeCorrelation C`, and `C negativeCorrelation A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type iter[tuple]

pybel_tools.analysis.stability.**jens_transformation_alpha**(*graph*)
Applies Jens' transformation (Type 1) to the graph

1. Induce a subgraph over causal + correlative edges
2. **Transform edges by the following rules:**
   - increases => increases
   - decreases => backwards increases
   - positive correlation => two way increases
   - negative correlation => delete

The resulting graph can be used to search for 3-cycles, which now symbolize unstable triplets where `A -> B`, `A -| C` and `B positiveCorrelation C`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type networkx.DiGraph

pybel_tools.analysis.stability.**jens_transformation_beta**(*graph*)
Applies Jens' Transformation (Type 2) to the graph

1. Induce a subgraph over causal and correlative relations
2. **Transform edges with the following rules:**
   - increases => backwards decreases
   - decreases => decreases
   - positive correlation => delete
   - negative correlation => two way decreases

The resulting graph can be used to search for 3-cycles, which now symbolize stable triples where `A -> B`, `A -| C` and `B negativeCorrelation C`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type networkx.DiGraph

pybel_tools.analysis.stability.**get_jens_unstable**(*graph*)
Yields triples of nodes where `A -> B`, `A -| C`, and `C positiveCorrelation A`. Calculated efficiently using the Jens Transformation.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterable of triplets of nodes

Return type iter[tuple]

pybel_tools.analysis.stability.**get_increase_mismatch_triplets**(*graph*)
Iterates over triples of nodes where `A -> B`, `A -> C`, and `C negativeCorrelation A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterable of triplets of nodes

Return type iter[tuple]

pybel_tools.analysis.stability.**get_decrease_mismatch_triplets**(*graph*)
Iterates over triplets of nodes where `A -| B`, `A -| C`, and `C negativeCorrelation A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterable of triplets of nodes

Return type iter[tuple]

pybel_tools.analysis.stability.**get_chaotic_triplets**(*graph*)
Iterates over triples of nodes that mutually increase each other, such as when `A -> B`, `B -> C`, and `C -> A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterable of triplets of nodes

Return type iter[tuple]

pybel_tools.analysis.stability.**get_dampened_triplets**(*graph*)
Iterates over triples of nodes that mutually decreases each other, such as when `A -| B`, `B -| C`, and `C -| A`.

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Returns An iterable of triplets of nodes

Return type iter[tuple]

pybel_tools.analysis.stability.**summarize_stability**(*graph*)
Summarize the stability of the graph

Parameters **graph** (*pybel.BELGraph*) – A BEL graph

Return type dict

2.11 Subgraph Expansion Workflow

pybel_tools.mutation.expansion.**get_peripheral_successor_edges**(*graph*, *subgraph*)
Gets the set of possible successor edges peripheral to the subgraph. The source nodes in this iterable are all inside the subgraph, while the targets are outside.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns An iterable of possible successor edges (4-tuples of node, node, key, data)

Return type iter[tuple]

pybel_tools.mutation.expansion.**get_peripheral_predecessor_edges**(*graph*, *subgraph*)
Gets the set of possible predecessor edges peripheral to the subgraph. The target nodes in this iterable are all inside the subgraph, while the sources are outside.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Return type iter[tuple]

pybel_tools.mutation.expansion.**count_sources**(*edge_iter*)
Counts the source nodes in an edge iterator with keys and data

Parameters **edge_iter** (*iter[tuple]*) – An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Returns A counter of source nodes in the iterable

Return type collections.Counter

pybel_tools.mutation.expansion.**count_targets**(*edge_iter*)
Counts the target nodes in an edge iterator with keys and data

Parameters **edge_iter** (*iter[tuple]*) – An iterable on possible predecessor edges (4-tuples of node, node, key, data)

Returns A counter of target nodes in the iterable

Return type collections.Counter

pybel_tools.mutation.expansion.**count_possible_successors**(*graph*, *subgraph*)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns A counter of possible successor nodes

Return type collections.Counter

pybel_tools.mutation.expansion.**count_possible_predecessors**(*graph*, *subgraph*)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** – An iterator of BEL nodes

Returns A counter of possible predecessor nodes

Return type collections.Counter

pybel_tools.mutation.expansion.**get_subgraph_edges**(*graph*, *annotation*, *value*, *source_filter=None*, *target_filter=None*)
Gets all edges from a given subgraph whose source and target nodes pass all of the given filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – The annotation to search
- **value** (*str*) – The annotation value to search by
- **source_filter** – Optional filter for source nodes (graph, node) -> bool
- **target_filter** – Optional filter for target nodes (graph, node) -> bool

Returns An iterable of (source node, target node, key, data) for all edges that match the annotation/value and node filters

Return type iter[tuple]

pybel_tools.mutation.expansion.**get_subgraph_peripheral_nodes**(*graph*, *subgraph*, *node_filters=None*, *edge_filters=None*)
Gets a summary dictionary of all peripheral nodes to a given subgraph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **subgraph** (*iter[tuple]*) – A set of nodes
- **node_filters** (*lambda*) – Optional. A list of node filter predicates with the interface (graph, node) -> bool. See *pybel_tools.filters.node_filters* for more information
- **edge_filters** (*lambda*) – Optional. A list of edge filter predicates with the interface (graph, node, node, key, data) -> bool. See *pybel_tools.filters.edge_filters* for more information

Returns A dictionary of {external node: {'successor': {internal node: list of (key, dict)}, 'predecessor': {internal node: list of (key, dict)}}}

Return type dict

For example, it might be useful to quantify the number of predecessors and successors

```
>>> import pybel_tools as pbt
>>> sgn = 'Blood vessel dilation subgraph'
>>> sg = pbt.selection.get_subgraph_by_annotation_value(graph, annotation=
↪'Subgraph', value=sgn)
>>> p = pbt.mutation.get_subgraph_peripheral_nodes(graph, sg, node_filters=pbt.
↪filters.exclude_pathology_filter)
>>> for node in sorted(p, key=lambda n: len(set(p[n]['successor']) | set(p[n][
↪'predecessor'])), reverse=True):
>>>     if 1 == len(p[sgn][node]['successor']) or 1 == len(p[sgn][node][
↪'predecessor']):
>>>         continue
>>>     print(node,
>>>           len(p[node]['successor']),
>>>           len(p[node]['predecessor']),
>>>           len(set(p[node]['successor']) | set(p[node]['predecessor'])))
```

pybel_tools.mutation.expansion.**expand_periphery**(*universe*, *graph*, *node_filters=None*, *edge_filters=None*, *threshold=2*)

Iterates over all possible edges, peripheral to a given subgraph, that could be added from the given graph. Edges could be added if they go to nodes that are involved in relationships that occur with more than the threshold (default 2) number of nodes in the subgraph.

Parameters

- **universe** (*pybel.BELGraph*) – The universe of BEL knowledge
- **graph** (*pybel.BELGraph*) – The (sub)graph to expand
- **node_filters** (*lambda*) – Optional. A list of node filter predicates with the interface (graph, node) -> bool. See *pybel_tools.filters.node_filters* for more information
- **edge_filters** (*lambda*) – Optional. A list of edge filter predicates with the interface (graph, node, node, key, data) -> bool. See *pybel_tools.filters.edge_filters* for more information
- **threshold** – Minimum frequency of betweenness occurrence to add a gap node

A reasonable edge filter to use is `pybel_tools.filters.keep_causal_edges()` because this function can allow for huge expansions if there happen to be hub nodes.

pybel_tools.mutation.expansion.**enrich_grouping**(*universe*, *graph*, *function*, *relation*)

Adds all of the grouped elements. See *enrich_complexes()*, *enrich_composites()*, and *enrich_reactions()*

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich
- **function** (*str*) – The function by which the subject of each triple is filtered
- **relation** (*str*) – The relationship by which the predicate of each triple is filtered

pybel_tools.mutation.expansion.**enrich_complexes**(*universe*, *graph*)
Add all of the members of the complex abundances to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.expansion.**enrich_composites**(*universe*, *graph*)
Adds all of the members of the composite abundances to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.expansion.**enrich_reactions**(*universe*, *graph*)
Adds all of the reactants and products of reactions to the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

pybel_tools.mutation.expansion.**enrich_variants**(*universe*, *graph*, *func=None*)
Add the reference nodes for all variants of the given function.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich
- **or iter[str] func** (*str*) – The function by which the subject of each triple is filtered. Defaults to the set of protein, rna, mirna, and gene.

pybel_tools.mutation.expansion.**enrich_unqualified**(*universe*, *graph*)
Enrich the subgraph with the unqualified edges from the graph.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich

The reason you might want to do this is you induce a subgraph from the original graph based on an annotation filter, but the unqualified edges that don't have annotations that most likely connect elements within your graph are not included.

See also:

This function thinly wraps the successive application of the following functions:

- *enrich_complexes()*
- *enrich_composites()*
- *enrich_reactions()*
- *enrich_variants()*

Equivalent to:

```
>>> enrich_complexes(universe, graph)
>>> enrich_composites(universe, graph)
>>> enrich_reactions(universe, graph)
>>> enrich_variants(universe, graph)
```

pybel_tools.mutation.expansion.**expand_internal**(*universe*, *graph*, *edge_filters=None*)

Edges between entities in the subgraph that pass the given filters

Parameters

- **universe** (*pybel.BELGraph*) – The full graph
- **graph** (*pybel.BELGraph*) – A subgraph to find the upstream information
- **edge_filters** (*list or lambda*) – Optional list of edge filter functions (graph, node, node, key, data) -> bool

pybel_tools.mutation.expansion.**expand_internal_causal**(*universe*, *graph*)

Adds causal edges between entities in the subgraph.

Is an extremely thin wrapper around *expand_internal()*.

Parameters

- **universe** (*pybel.BELGraph*) – A BEL graph representing the universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph to enrich with causal relations between contained nodes

Equivalent to:

```
>>> import pybel_tools as pbt
>>> from pybel.struct.filters.edge_predicates import is_causal_relation
>>> pbt.mutation.expand_internal(universe, graph, edge_filters=is_causal_relation)
```

2.12 Unbiased Candidate Mechanism Generation

The Unbiased Candidate Mechanism Generation workflow addresses the inconsistency in the definitions of the boundaries of pathways, mechanisms, subgraphs, etc. in networks and systems biology that are introduced during curation due to a variety of reasons.

A simple approach for generating unbiased candidate mechanisms is to take the upstream controlles

This module provides functions for generating subgraphs based around a single node, most likely a biological process.

Subgraphs induced around biological processes should prove to be subgraphs of the NeuroMMSig/canonical mechanisms and provide an even more rich mechanism inventory.

2.12.1 Examples

This method has been applied in the following Jupyter Notebooks:

- Generating Unbiased Candidate Mechanisms

pybel_tools.generation.**remove_unweighted_leaves**(*graph*, *key=None*)
Remove nodes that are leaves and that don't have a weight (or other key) attribute set.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT.

pybel_tools.generation.**is_unweighted_source**(*graph*, *node*, *key*)
Check if the node is both a source and also has an annotation.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – A BEL node
- **key** (*str*) – The key in the node data dictionary representing the experimental data

pybel_tools.generation.**get_unweighted_sources**(*graph*, *key=None*)
Get nodes on the periphery of the subgraph that do not have a annotation for the given key.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*str*) – The key in the node data dictionary representing the experimental data

Returns An iterator over BEL nodes that are unannotated and on the periphery of this subgraph

Return type iter[tuple]

pybel_tools.generation.**remove_unweighted_sources**(*graph*, *key=None*)
Prunes unannotated nodes on the periphery of the subgraph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT.

pybel_tools.generation.**prune_mechanism_by_data**(*graph*, *key=None*)
Removes all leaves and source nodes that don't have weights. Is a thin wrapper around *remove_unweighted_leaves()* and *remove_unweighted_sources()*

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT.

Equivalent to:

```
>>> remove_unweighted_leaves(graph)
>>> remove_unweighted_sources(graph)
```

pybel_tools.generation.**generate_mechanism**(*graph*, *node*, *key=None*)
Generates a mechanistic subgraph upstream of the given node

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **node** (*tuple*) – The target BEL node for generation
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT.

Returns A subgraph grown around the target BEL node

Return type pybel.BELGraph

pybel_tools.generation.**generate_bioprocess_mechanisms**(*graph*, *key=None*)
Generate a mechanistic subgraph for each biological process in the graph using *generate_mechanism()*

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT

Returns A dictionary from {tuple bioprocess node: BELGraph candidate mechanism}

Return type dict[tuple, pybel.BELGraph]

2.13 Heat Diffusion Workflow

This module describes a heat diffusion workflow for analyzing BEL networks with differential gene expression[0].

It has four parts:

1. Assembling a network, pre-processing, and overlaying data
2. Generating unbiased candidate mechanisms from the network
3. Generating random subgraphs from each unbiased candidate mechanism
4. Applying standard heat diffusion to each subgraph and calculating scores for each unbiased candidate mechanism based on the distribution of scores for its subgraph

In this algorithm, heat is applied to the nodes based on the data set. For the differential gene expression experiment, the log-fold-change values are used instead of the corrected p-values to allow for the effects of up- and down-regulation to be admitted in the analysis. Finally, heat diffusion inspired by previous algorithms publsihed in systems and networks biology[1][2] is run with the constraint that decreases edges cause the sign of the heat to be flipped. Because of the construction of unbiased candidate mechanisms, all heat will flow towards their seed biological process nodes. The amount of heat on the biological process node after heat diffusion stops becomes the score for the whole unbiased candidate mechanism.

[0] Hoyt, C. T., Konotopez, A., Ebeling, C., & Wren, J. (2018). PyBEL: a computational framework for Biological Expression Language. Bioinformatics (Oxford, England), 34(4), 703–704.

[1] Bernabo N., *et al.* (2014). The biological networks in studying cell signal transduction complexity: The examples of sperm capacitation and of endocannabinoid system. Computational and Structural Biotechnology Journal, 11 (18), 11–21.

[2] Leiserson, M. D. M., *et al.* (2015). Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. Nature Genetics, 47 (2), 106–14.

The issue of inconsistent causal networks addressed by SST[3] does not affect heat diffusion algorithms since it can quantify multiple conflicting pathways. However, it does not address the possibility of contradictory edges, for example, when `A increases B` and `A decreases B` are both true. A random sampling approach is used on networks with contradictory edges and aggregate statistics over multiple trials are used to assess the robustness of the scores as a function of the topology of the underlying unbiases candidate mechanisms.

2.13.1 Invariants

- Because heat always flows towards the biological process node, it is possible to remove leaf nodes (nodes with no incoming edges) after each step, since their heat will never change.

2.13.2 Examples

This workflow has been applied in several Jupyter notebooks:

- Heat Diffusion Workflow
- Time Series Heat Diffusion

2.13.3 Future Work

This algorithm can be tuned to allow the use of correlative relationships. Because many multi-scale and multi-modal data are often measured with correlations to molecular features, this enables experiments to be run using SNP or brain imaging features, whose experiments often measure their correlation with the activity of gene products.

pybel_tools.analysis.heat.**RESULT_LABELS = ['avg', 'stddev', 'normality', 'median', 'neighbo**
The columns in the score tuples

pybel_tools.analysis.heat.**calculate_average_scores_on_graph**(*graph*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *use_tqdm=False*)

Calculates the scores over all biological processes in the sub-graph.

As an implementation, it simply computes the sub-graphs then calls *calculate_average_scores_on_subgraphs()* as described in that function's documentation.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph with heats already on the nodes
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*Optional[str]*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*Optional[float]*) – The initial score for all nodes. This number can go up or down.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **use_tqdm** (*bool*) – Should there be a progress bar for runners?

[3] Vasilyev, D. M., *et al.* (2014). An algorithm for score aggregation over causal biological networks based on random walk sampling. BMC Research Notes, 7, 516.

Returns A dictionary of {pybel node tuple: results tuple}

Return type dict[tuple, tuple]

Suggested usage with `pandas`:

```
>>> import pandas as pd
>>> from pybel_tools.analysis.heat import calculate_average_scores_on_graph
>>> graph = ...  # load graph and data
>>> scores = calculate_average_scores_on_graph(graph)
>>> pd.DataFrame.from_items(scores.items(), orient='index', columns=RESULT_LABELS)
```

pybel_tools.analysis.heat.**calculate_average_scores_on_subgraphs**(*subgraphs*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *use_tqdm=False*)

Calculate the scores over precomputed candidate mechanisms.

Parameters

- **subgraphs** (*dict[tuple,pybel.BELGraph]*) – A dictionary of {tuple node: pybel.BELGraph candidate mechanism}
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*Optional[str]*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*Optional[float]*) – The initial score for all nodes. This number can go up or down.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **use_tqdm** (*bool*) – Should there be a progress bar for runners?

Returns A dictionary of {pybel node tuple: results tuple}

Return type dict[tuple, tuple]

Example Usage:

```
>>> import pandas as pd
>>> from pybel_tools.generation import generate_bioprocess_mechanisms
>>> from pybel_tools.analysis.heat import calculate_average_scores_on_subgraphs
>>> # load graph and data
>>> graph = ...
>>> candidate_mechanisms = generate_bioprocess_mechanisms(graph)
>>> scores = calculate_average_scores_on_subgraphs(candidate_mechanisms)
>>> pd.DataFrame.from_items(scores.items(), orient='index', columns=RESULT_LABELS)
```

pybel_tools.analysis.heat.**workflow**(*graph*, *node*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *minimum_nodes=1*)

Generate candidate mechanisms and run the heat diffusion workflow.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – The BEL node that is the focus of this analysis

- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*Optional[str]*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*Optional[float]*) – The initial score for all nodes. This number can go up or down.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **minimum_nodes** (*int*) – The minimum number of nodes a sub-graph needs to try running heat diffusion

Returns A list of runners

Return type list[*Runner*]

pybel_tools.analysis.heat.**multirun**(*graph*, *node*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *use_tqdm=False*)

Run the heat diffusion workflow multiple times, each time yielding a *Runner* object upon completion.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – The BEL node that is the focus of this analysis
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*str*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **use_tqdm** (*bool*) – Should there be a progress bar for runners?

Returns An iterable over the runners after each iteration

Return type iter[*Runner*]

class pybel_tools.analysis.heat.**Runner**(*graph*, *target_node*, *key=None*, *tag=None*, *default_score=None*)

This class houses the data related to a single run of the heat diffusion workflow.

Initializes the heat diffusion runner class

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **target_node** (*tuple*) – The BEL node that is the focus of this analysis
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*str*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial score for all nodes. This number can go up or down.

iter_leaves()
Returns an iterable over all nodes that are leaves. A node is a leaf if either:

- it doesn't have any predecessors, OR
- all of its predecessors have a score in their data dictionaries

Returns An iterable over all leaf nodes

Return type iter

has_leaves()
Returns if the current graph has any leaves.

Implementation is not that smart currently, and does a full sweep.

Returns Does the current graph have any leaves?

Return type bool

in_out_ratio(*node*)
Calculates the ratio of in-degree / out-degree of a node

Parameters **node** (*tuple*) – A BEL node

Returns The in-degree / out-degree ratio for the given node

Return type float

unscored_nodes_iter()
Iterates over all nodes without a score

get_random_edge()
This function should be run when there are no leaves, but there are still unscored nodes. It will introduce a probabilistic element to the algorithm, where some edges are disregarded randomly to eventually get a score for the network. This means that the score can be averaged over many runs for a given graph, and a better data structure will have to be later developed that doesn't destroy the graph (instead, annotates which edges have been disregarded, later)

1. get all unscored
2. rank by in-degree
3. weighted probability over all in-edges where lower in-degree means higher probability
4. pick randomly which edge

Returns A random in-edge to the lowest in/out degree ratio node. This is a 3-tuple of (node, node, key)

Return type tuple

remove_random_edge()
Remove a random in-edge from the node with the lowest in/out degree ratio.

remove_random_edge_until_has_leaves()
Remove random edges until there is at least one leaf node.

score_leaves()
Calculate the score for all leaves.

Returns The set of leaf nodes that were scored

Return type set

run()
Calculate scores for all leaves until there are none, removes edges until there are, and repeats until all nodes have been scored.

run_with_graph_transformation()
Calculate scores for all leaves until there are none, removes edges until there are, and repeats until all nodes have been scored. Also, yields the current graph at every step so you can make a cool animation of how the graph changes throughout the course of the algorithm

Returns An iterable of BEL graphs

Return type iter[pybel.BELGraph]

done_chomping()
Determines if the algorithm is complete by checking if the target node of this analysis has been scored yet. Because the algorithm removes edges when it gets stuck until it is un-stuck, it is always guaranteed to finish.

Returns Is the algorithm done running?

Return type bool

get_final_score()
Return the final score for the target node.

Returns The final score for the target node

Return type float

calculate_score(*node*)
Calculate the score of the given node.

Parameters **node** (*tuple*) – A node in the BEL graph

Returns The new score of the node

Return type float

get_remaining_graph()
Allows for introspection on the algorithm at a given point by returning the sub-graph induced by all unscored nodes

Returns The remaining unscored BEL graph

Return type pybel.BELGraph

pybel_tools.analysis.heat.**workflow_aggregate**(*graph*, *node*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *aggregator=None*)
Get the average score over multiple runs.

This function is very simple, and can be copied to do more interesting statistics over the *Runner* instances. To iterate over the runners themselves, see *workflow()*

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – The BEL node that is the focus of this analysis
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to pybel_tools.constants.WEIGHT.
- **tag** (*Optional[str]*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'

- **default_score** (*Optional[float]*) – The initial score for all nodes. This number can go up or down.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **aggregator** (*Optional[list[float] -> float]*) – A function that aggregates a list of scores. Defaults to `numpy.average()`. Could also use: `numpy.mean()`, `numpy.median()`, `numpy.min()`, `numpy.max()`

Returns The average score for the target node

Return type float

pybel_tools.analysis.heat.**workflow_all**(*graph*, *key=None*, *tag=None*, *default_score=None*, *runs=None*)

Run the heat diffusion workflow and get runners for every possible candidate mechanism

1. Get all biological processes
2. Get candidate mechanism induced two level back from each biological process
3. Heat diffusion workflow for each candidate mechanism for multiple runs
4. Return all runner results

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **tag** (*str*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the heat diffusion workflow. Defaults to 100.

Returns A dictionary of {node: list of runners}

Return type dict[tuple,list[*Runner*]]

pybel_tools.analysis.heat.**workflow_all_aggregate**(*graph*, *key=None*, *tag=None*, *default_score=None*, *runs=None*, *aggregator=None*)

Run the heat diffusion workflow to get average score for every possible candidate mechanism.

1. Get all biological processes
2. Get candidate mechanism induced two level back from each biological process
3. Heat diffusion workflow on each candidate mechanism for multiple runs
4. Report average scores for each candidate mechanism

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.

- **tag** (*Optional[str]*) – The key for the nodes' data dictionaries where the scores will be put. Defaults to 'score'
- **default_score** (*Optional[float]*) – The initial score for all nodes. This number can go up or down.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **aggregator** (*Optional[list[float] -> float]*) – A function that aggregates a list of scores. Defaults to `numpy.average()`. Could also use: `numpy.mean()`, `numpy.median()`, `numpy.min()`, `numpy.max()`

Returns A dictionary of {node: upstream causal subgraph}

Return type dict

pybel_tools.analysis.heat.**calculate_average_score_by_annotation**(*graph*, *annotation*, *key=None*, *runs=None*, *use_tdqm=False*)

For each sub-graph induced over the edges matching the annotation, calculate the average score for all of the contained biological processes

Assumes you haven't done anything yet

1. Generates biological process upstream candidate mechanistic sub-graphs with `generate_bioprocess_mechanisms()`
2. Calculates scores for each sub-graph with `calculate_average_scores_on_sub-graphs()`
3. Overlays data with pbt.integration.overlay_data
4. Calculates averages with pbt.selection.group_nodes.average_node_annotation

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotation** (*str*) – A BEL annotation
- **key** (*Optional[str]*) – The key in the node data dictionary representing the experimental data. Defaults to `pybel_tools.constants.WEIGHT`.
- **runs** (*Optional[int]*) – The number of times to run the heat diffusion workflow. Defaults to 100.
- **use_tqdm** (*bool*) – Should there be a progress bar for runners?

Returns A dictionary from {str annotation value: tuple scores}

Return type dict[str,tuple]

Example Usage:

```
>>> import pybel
>>> from pybel_tools.integration import overlay_data
>>> from pybel_tools.analysis.heat import calculate_average_score_by_annotation
>>> graph = pybel.from_path(...)
>>> scores = calculate_average_score_by_annotation(graph, 'subgraph')
```

2.14 Algorithms

2.14.1 NeuroMMSig

An implementation of the NeuroMMSig mechanism enrichment algorithm *[DomingoFernandez2017]*.

pybel_tools.analysis.neurommsig.algorithm.**get_neurommsig_scores_prestratified**(*it*, *genes*, *ora_weight=None*, *hub_weight=None*, *top=None*, *topology_weight=None*

Takes a graph stratification and runs neurommsig on each

Parameters

- **it** (*iter[tuple[str,pybel.BELGraph]]*) – A pre-stratified set of graphs
- **genes** (*list[tuple]*) – A list of gene nodes
- **ora_weight** (*Optional[float]*) – The relative weight of the over-enrichment analysis score from `neurommsig_gene_ora()`. Defaults to 1.0.
- **hub_weight** (*Optional[float]*) – The relative weight of the hub analysis score from `neurommsig_hubs()`. Defaults to 1.0.
- **top** (*Optional[float]*) – The percentage of top genes to use as hubs. Defaults to 5% (0.05).
- **topology_weight** (*Optional[float]*) – The relative weight of the topolgical analysis core from `neurommsig_topology()`. Defaults to 1.0.
- **preprocess** (*bool*) – If true, preprocess the graph.

Returns A dictionary from {annotation value: NeuroMMSig composite score}

Return type Optional[dict[str, float]]

Pre-processing steps:

1. Infer the central dogma with :func:‘‘
2. Collapse all proteins, RNAs and miRNAs to genes with :func:‘‘
3. Collapse variants to genes with :func:‘‘

pybel_tools.analysis.neurommsig.algorithm.**get_neurommsig_scores**(*graph*, *genes*, *annotation='Subgraph'*, *ora_weight=None*, *hub_weight=None*, *top=None*, *topology_weight=None*, *preprocess=False*)

Preprocesses the graph, stratifies by the given annotation, then runs the NeuroMMSig algorithm on each.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph

- **genes** (*list[tuple]*) – A list of gene nodes
- **annotation** (*str*) – The annotation to use to stratify the graph to subgraphs
- **ora_weight** (*Optional[float]*) – The relative weight of the over-enrichment analysis score from `neurommsig_gene_ora()`. Defaults to 1.0.
- **hub_weight** (*Optional[float]*) – The relative weight of the hub analysis score from `neurommsig_hubs()`. Defaults to 1.0.
- **top** (*Optional[float]*) – The percentage of top genes to use as hubs. Defaults to 5% (0.05).
- **topology_weight** (*Optional[float]*) – The relative weight of the topolgical analysis core from `neurommsig_topology()`. Defaults to 1.0.
- **preprocess** (*bool*) – If true, preprocess the graph.

Returns A dictionary from {annotation value: NeuroMMSig composite score}

Return type Optional[dict[str, float]]

Pre-processing steps:

1. Infer the central dogma with :func:‘‘
2. Collapse all proteins, RNAs and miRNAs to genes with :func:‘‘
3. Collapse variants to genes with :func:‘‘

`pybel_tools.analysis.neurommsig.algorithm.`**`get_neurommsig_score`**(*graph*, *target_genes*, *ora_weight=None*, *hub_weight=None*, *top=None*, *topology_weight=None*)

Calculates the composite NeuroMMSig Score for a given list of genes.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **target_genes** (*list[tuple]*) – A list of gene nodes
- **ora_weight** (*Optional[float]*) – The relative weight of the over-enrichment analysis score from `neurommsig_gene_ora()`. Defaults to 1.0.
- **hub_weight** (*Optional[float]*) – The relative weight of the hub analysis score from `neurommsig_hubs()`. Defaults to 1.0.
- **top** (*Optional[float]*) – The percentage of top genes to use as hubs. Defaults to 5% (0.05).
- **topology_weight** (*Optional[float]*) – The relative weight of the topolgical analysis core from `neurommsig_topology()`. Defaults to 1.0.

Returns The NeuroMMSig composite score

Return type float

2.14.2 EpiCom

An implementation of chemical-based mechanism enrichment with NeuroMMSig *[HoytDomingoFernandez2018]*.

This algorithm has multiple steps:

1. Select NeuroMMSig networks for AD, PD, and epilepsy
2. Select drugs from DrugBank, and their targets
3. Run NeuroMMSig algorithm on target list for each network and each mechanism
4. Store in database

pybel_tools.analysis.epicom.algorithm.**epicom_on_graph**(*graph*, *dtis*, *preprocess=True*)

Parameters

- **graph** (*pybel.BELGraph*) –
- **dtis** (*dict[str,list[tuple]]*) –
- **preprocess** (*bool*) – If true, preprocess the graph with `neurommsig_graph_preprocessor()`.

Return type iter[tuple[str,str,float]]

pybel_tools.analysis.epicom.algorithm.**multi_run_epicom**(*graphs*, *path*)

Run EpiCom analysis on many graphs

Parameters

- **graphs** (*iter[pybel.BELGraph]*) –
- **or file path** (*str*) – output file path

pybel_tools.analysis.epicom.algorithm.**run_epicom**(*graph*, *directory*)

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **directory** (*str*) – The directory in which the algorithm is run

Return type iter[tuple[str,str,str,float]]

2.14.3 Reverse Causal Reasoning

An implementation of Reverse Causal Reasoning (RCR) *[Catlett2013]*.

pybel_tools.analysis.rcr.**run_rcr**(*graph*, *tag='dgxp'*)

Runs the reverse causal reasoning algorithm on a graph.

Steps:

1. Get all downstream controlled things into map (that have at least 4 downstream things)
2. calculate population of all things that are downstream controlled

Note: Assumes all nodes have been pre-tagged with data

Parameters

- **graph** (*pybel.BELGraph*) –
- **tag** (*str*) – The key for the nodes' data dictionaries that corresponds to the integer value for its differential expression.

2.14.4 Sample of Spanning Trees

An implementation of the sampling of spanning trees (SST) algorithm [Vasilyev2014]_.

pybel_tools.analysis.sst.**rank_edges**(*edges*, *edge_ranking=None*)
Returns the highest ranked edge from a multiedge

Parameters

- **edges** (*dict*) – dictionary with all edges between two nodes
- **edge_ranking** (*dict*) – A dictionary of {relationship: score}

Returns Highest ranked edge

Return type tuple: (edge id, relation, score given ranking)

class pybel_tools.analysis.sst.**Effect**
Represents the possible effect of a root node on a target

pybel_tools.analysis.sst.**get_path_effect**(*graph*, *path*, *relationship_dict*)
Calculates the final effect of the root node to the sink node in the path

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **path** (*list*) – Path from root to sink node
- **relationship_dict** (*dict*) – dictionary with relationship effects

Return type *Effect*

pybel_tools.analysis.sst.**run_cna**(*graph*, *root*, *targets*, *relationship_dict=None*)
Returns the effect from the root to the target nodes represented as {-1,1}

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **root** (*BaseEntity*) – The root node
- **targets** (*iter*) – The targets nodes
- **relationship_dict** (*dict*) – dictionary with relationship effects

Return list[tuple]

pybel_tools.analysis.sst.**get_random_walk_spanning_tree**(*graph*)
Generates a spanning tree from the directed graph using the random walk appropach proposed independently by by Broder (1989) and Aldous (1990). It simply generates random walks until all nodes have been covered.

Algorithm:

1. Choose a starting vertex s arbitrarily. Set T_V ← {s} and T_E ← .
2. **Do a simple random walk starting at s. Whenever we cross an edge e = {u, v} with v V ,** add v to TV and add e to TE.
3. Stop the random walk when TV = V . Output T = (T_V , T_E) as our spanning tree

Parameters **graph** (*networkx.DiGraph*) – The input graph

Return type networkx.DiGraph

See also:

- https://math.dartmouth.edu/~pw/math100w13/kothari.pdf
- http://keyulux.com/pdf/spanning_tree.pdf

pybel_tools.analysis.sst.**rank_causalr_hypothesis**(*graph*, *node_to_regulation*, *regulator_node*)

Returns the results of testing the regulator hypothesis of the given node on the input data. Note: this method returns both +/- signed hypotheses evaluated The algorithm is described in G Bradley et al. (2017)

Algorithm

1. Calculate the shortest path between the regulator node and each node in observed_regulation

2. Calculate the concordance of the causal network and the observed regulation when there is path between target node and regulator node

Parameters

- **graph** (*networkx.DiGraph*) – A causal graph
- **node_to_regulation** (*dict*) – Nodes to score (1,-1,0)
- **graph** – A causal graph

Return Dictionaries with hypothesis results (keys score, correct, incorrect, ambiguous)

Return type dict

See also:

- https://doi.org/10.1093/bioinformatics/btx425

2.15 IO Utilities

Utilities for loading and exporting BEL graphs

pybel_tools.ioutils.**get_paths_recursive**(*directory*, *extension='.bel'*, *exclude_directory_pattern=None*)

Gets all file paths in a given directory to BEL documents

Parameters

- **directory** (*str*) – The base directory to walk
- **extension** (*str*) – Extensions of files to keep
- **exclude_directory_pattern** (*str*) – Any directory names to exclude

pybel_tools.ioutils.**convert_paths**(*paths*, *manager=None*, *upload=False*, *do_enrich_protein_and_rna_origins=True*, *do_enrich_pubmed_citations=False*, *do_to_web=False*, ***kwargs*)

Parse and either uploads/pickles graphs in a given set of files, recursively.

Parameters

- **paths** (*iter[str]*) – The paths to convert
- **upload** (*bool*) – Should the networks be uploaded to the cache?
- **do_enrich_protein_and_rna_origins** (*bool*) – Should the RNA and gene be inferred for each protein?

- **do_enrich_pubmed_citations** (*bool*) – Should the citations be enriched using Entrez Utils?
- **do_to_web** (*bool*) – Send to BEL Commons?
- **kwargs** – Parameters to pass to `pybel.from_path()`

Returns A pair of a dictionary {path: bel graph} and list of failed paths

Return type tuple[dict[str,pybel.BELGraph],list[str]]

pybel_tools.ioutils.**convert_directory**(*directory*, *manager=None*, *upload=False*, *pickle=False*, *canonicalize=True*, *do_enrich_protein_and_rna_origins=True*, *enrich_citations=False*, *enrich_genes=False*, *enrich_go=False*, *send=False*, *exclude_directory_pattern=None*, *version_in_path=False*, ***kwargs*)

Parse and either uploads/pickles graphs in a given directory and recursively for sub-directories.

Parameters

- **directory** (*str*) – The directory to look through
- **upload** (*bool*) – Should the networks be uploaded to the cache?
- **pickle** (*bool*) – Should the networks be saved as pickles?
- **do_enrich_protein_and_rna_origins** (*bool*) – Should the central dogma be inferred for all proteins, RNAs, and miRNAs
- **enrich_citations** (*bool*) – Should the citations be enriched using Entrez Utils?
- **enrich_genes** (*bool*) – Should the genes' descriptions be downloaded from Gene Cards?
- **enrich_go** (*bool*) – Should the biological processes' descriptions be downloaded from Gene Ontology?
- **send** (*bool*) – Send to PyBEL Web?
- **exclude_directory_pattern** (*str*) – A pattern to use to skip directories
- **version_in_path** (*bool*) – Add the current pybel version to the pathname
- **kwargs** – Parameters to pass to `pybel.from_path()`

pybel_tools.ioutils.**upload_recursive**(*directory*, *manager=None*, *exclude_directory_pattern=None*)

Recursively uploads all gpickles in a given directory and sub-directories

Parameters

- **directory** (*str*) – the directory to traverse
- **exclude_directory_pattern** (*Optional[str]*) – Any directory names to exclude

pybel_tools.ioutils.**subgraphs_to_pickles**(*network*, *annotation*, *directory=None*)

Groups the given graph into subgraphs by the given annotation with `get_subgraph_by_annotation()` and outputs them as gpickle files to the given directory with `pybel.to_pickle()`

Parameters

- **network** (*pybel.BELGraph*) – A BEL network
- **annotation** (*str*) – An annotation to split by. Suggestion: `Subgraph`

- **directory** (*Optional[str]*) – A directory to output the pickles

2.16 Document Utilities

2.16.1 Creating Definition Documents

pybel_tools.definition_utils.**get_merged_namespace_names**(*locations*, *check_keywords=True*)

Loads many namespaces and combines their names.

Parameters

- **locations** (*iter[str]*) – An iterable of URLs or file paths pointing to BEL namespaces.
- **check_keywords** (*bool*) – Should all the keywords be the same? Defaults to `True`

Returns A dictionary of {names: labels}

Return type dict[str, str]

Example Usage

```
>>> from pybel.resources.definitions import write_namespace
>>> from pybel_tools.definition_utils import export_namespace, get_merged_
↪namespace_names
>>> graph = ...
>>> original_ns_url = ...
>>> export_namespace(graph, 'MBS') # Outputs in current directory to MBS.belns
>>> value_dict = get_merged_namespace_names([original_ns_url, 'MBS.belns'])
>>> with open('merged_namespace.belns', 'w') as f:
>>> ...  write_namespace('MyBrokenNamespace', 'MBS', 'Other', 'Charles Hoyt',
↪'PyBEL Citation', value_dict, file=f)
```

pybel_tools.definition_utils.**merge_namespaces**(*input_locations*, *output_path*, *namespace_name*, *namespace_keyword*, *namespace_domain*, *author_name*, *citation_name*, *namespace_description=None*, *namespace_species=None*, *namespace_version=None*, *namespace_query_url=None*, *namespace_created=None*, *author_contact=None*, *author_copyright=None*, *citation_description=None*, *citation_url=None*, *citation_version=None*, *citation_date=None*, *case_sensitive=True*, *delimiter='|'*, *cacheable=True*, *functions=None*, *value_prefix=''*, *sort_key=None*, *check_keywords=True*)

Merges namespaces from multiple locations to one.

Parameters

- **input_locations** (*iter*) – An iterable of URLs or file paths pointing to BEL namespaces.
- **output_path** (*str*) – The path to the file to write the merged namespace
- **namespace_name** (*str*) – The namespace name
- **namespace_keyword** (*str*) – Preferred BEL Keyword, maximum length of 8
- **namespace_domain** (*str*) – One of: `pybel.constants.NAMESPACE_DOMAIN_BIOPROCESS`, `pybel.constants.NAMESPACE_DOMAIN_CHEMICAL`, `pybel.constants.NAMESPACE_DOMAIN_GENE`, or `pybel.constants.NAMESPACE_DOMAIN_OTHER`
- **author_name** (*str*) – The namespace's authors
- **citation_name** (*str*) – The name of the citation
- **namespace_query_url** (*str*) – HTTP URL to query for details on namespace values (must be valid URL)
- **namespace_description** (*str*) – Namespace description
- **namespace_species** (*str*) – Comma-separated list of species taxonomy id's
- **namespace_version** (*str*) – Namespace version
- **namespace_created** (*str*) – Namespace public timestamp, ISO 8601 datetime
- **author_contact** (*str*) – Namespace author's contact info/email address
- **author_copyright** (*str*) – Namespace's copyright/license information
- **citation_description** (*str*) – Citation description
- **citation_url** (*str*) – URL to more citation information
- **citation_version** (*str*) – Citation version
- **citation_date** (*str*) – Citation publish timestamp, ISO 8601 Date
- **case_sensitive** (*bool*) – Should this config file be interpreted as case-sensitive?
- **delimiter** (*str*) – The delimiter between names and labels in this config file
- **cacheable** (*bool*) – Should this config file be cached?
- **functions** (*iterable of characters*) – The encoding for the elements in this namespace
- **value_prefix** (*str*) – a prefix for each name
- **sort_key** – A function to sort the values with `sorted()`
- **check_keywords** (*bool*) – Should all the keywords be the same? Defaults to `True`

pybel_tools.definition_utils.**export_namespace**(*graph*, *namespace*, *directory=None*, *cacheable=False*)

Exports all names and missing names from the given namespace to its own BEL Namespace files in the given directory.

Could be useful during quick and dirty curation, where planned namespace building is not a priority.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespace** (*str*) – The namespace to process

- **directory** (*str*) – The path to the directory where to output the namespace. Defaults to the current working directory returned by `os.getcwd()`
- **cacheable** (*bool*) – Should the namespace be cacheable? Defaults to `False` because, in general, this operation will probably be used for evil, and users won't want to reload their entire cache after each iteration of curation.

pybel_tools.definition_utils.**export_namespaces**(*graph*, *namespaces*, *directory=None*, *cacheable=False*)

Thinly wraps *export_namespace()* for an iterable of namespaces.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **namespaces** (*iter[str]*) – An iterable of strings for the namespaces to process
- **directory** (*str*) – The path to the directory where to output the namespaces. Defaults to the current working directory returned by `os.getcwd()`
- **cacheable** (*bool*) – Should the namespaces be cacheable? Defaults to `False` because, in general, this operation will probably be used for evil, and users won't want to reload their entire cache after each iteration of curation.

2.16.2 Creating Knowledge Documents

pybel_tools.document_utils.**write_boilerplate**(*name*, *version=None*, *description=None*, *authors=None*, *contact=None*, *copyright=None*, *licenses=None*, *disclaimer=None*, *namespace_url=None*, *namespace_patterns=None*, *annotation_url=None*, *annotation_patterns=None*, *annotation_list=None*, *pmids=None*, *entrez_ids=None*, *file=None*)

Writes a boilerplate BEL document, with standard document metadata, definitions.

Parameters

- **name** (*str*) – The unique name for this BEL document
- **contact** (*str*) – The email address of the maintainer
- **description** (*str*) – A description of the contents of this document
- **authors** (*str*) – The authors of this document
- **version** (*str*) – The version. Defaults to current date in format `YYYYMMDD`.
- **copyright** (*str*) – Copyright information about this document
- **licenses** (*str*) – The license applied to this document
- **disclaimer** (*str*) – The disclaimer for this document
- **namespace_url** (*dict[str,str]*) – an optional dictionary of {str name: str URL} of namespaces
- **namespace_patterns** (*dict[str,str]*) – An optional dictionary of {str name: str regex} namespaces
- **annotation_url** (*dict[str,str]*) – An optional dictionary of {str name: str URL} of annotations

- **annotation_patterns** (*dict[str,str]*) – An optional dictionary of {str name: str regex} of regex annotations
- **annotation_list** (*dict[str,set[str]]*) – An optional dictionary of {str name: set of names} of list annotations
- **or iter[int] pmids** (*iter[str]*) – A list of PubMed identifiers to auto-populate with citation and abstract
- **or iter[int] entrez_ids** (*iter[str]*) – A list of Entrez identifiers to autopopulate the gene summary as evidence
- **file** (*file*) – A writable file or file-like. If None, defaults to `sys.stdout`

pybel_tools.document_utils.**lint_file**(*in_file*, *out_file=None*)
Helps remove extraneous whitespace from the lines of a file

Parameters

- **in_file** (*file*) – A readable file or file-like
- **out_file** (*file*) – A writable file or file-like

pybel_tools.document_utils.**lint_directory**(*source*, *target*)
Adds a linted version of each document in the source directory to the target directory

Parameters

- **source** (*str*) – Path to directory to lint
- **target** (*str*) – Path to directory to output

2.17 Utilities

This module contains functions useful throughout PyBEL Tools

pybel_tools.utils.**pairwise**(*iterable*)
Iterate over pairs in list s -> (s0,s1), (s1,s2), (s2, s3), ...

pybel_tools.utils.**graph_edge_data_iter**(*graph*)
Iterate over the edge data dictionaries.

Returns An iterator over the edge dictionaries in the graph

Return type iter[dict]

pybel_tools.utils.**count_defaultdict**(*dict_of_lists*)
Takes a dictionary and applies a counter to each list

Parameters **dict_of_lists** (*dict or collections.defaultdict*) – A dictionary of lists

Returns A dictionary of {key: Counter(values)}

Return type dict

pybel_tools.utils.**count_dict_values**(*dict_of_counters*)
Counts the number of elements in each value (can be list, Counter, etc)

Parameters **dict_of_counters** (*dict or collections.defaultdict*) – A dictionary of things whose lengths can be measured (lists, Counters, dicts)

Returns A Counter with the same keys as the input but the count of the length of the values list/tuple/set/Counter

Return type collections.Counter

pybel_tools.utils.**set_percentage**(*x*, *y*)
What percentage of x is contained within y?

Parameters

- **x** (*set*) – A set
- **y** (*set*) – Another set

Returns The percentage of x contained within y

Return type float

pybel_tools.utils.**tanimoto_set_similarity**(*x*, *y*)
Calculates the tanimoto set similarity

Parameters

- **x** (*set*) – A set
- **y** (*set*) – Another set

Returns The similarity between

Return type float

pybel_tools.utils.**min_tanimoto_set_similarity**(*x*, *y*)
Calculates the tanimoto set similarity using the minimum size

Parameters

- **x** (*set*) – A set
- **y** (*set*) – Another set

Returns The similarity between

Return type float

pybel_tools.utils.**calculate_single_tanimoto_set_distances**(*target*, *dict_of_sets*)
Returns a dictionary of distances keyed by the keys in the given dict. Distances are calculated based on pairwise tanimoto similarity of the sets contained

Parameters

- **target** (*set*) – A set
- **dict_of_sets** (*dict*) – A dict of {x: set of y}

Returns A similarity dicationary based on the set overlap (tanimoto) score between the target set and the sets in dos

Return type dict

pybel_tools.utils.**calculate_tanimoto_set_distances**(*dict_of_sets*)
Returns a distance matrix keyed by the keys in the given dict. Distances are calculated based on pairwise tanimoto similarity of the sets contained

Parameters **dict_of_sets** (*dict*) – A dict of {x: set of y}

Returns A similarity matrix based on the set overlap (tanimoto) score between each x as a dict of dicts

Return type dict

pybel_tools.utils.**calculate_global_tanimoto_set_distances**(*dict_of_sets*)
Calculates an alternative distance matrix based on the following equation:

$$distance(A, B) = 1 - \|A \cup B\| / \| \cup_{s \in S} s\|$$

Parameters **dict_of_sets** (*dict*) – A dict of {x: set of y}

Returns A similarity matrix based on the alternative tanimoto distance as a dict of dicts

Return type dict

pybel_tools.utils.**barh**(*d*, *plt*, *title=None*)
A convenience function for plotting a horizontal bar plot from a Counter

pybel_tools.utils.**barv**(*d*, *plt*, *title=None*, *rotation='vertical'*)
A convenience function for plotting a vertical bar plot from a Counter

pybel_tools.utils.**safe_add_edge**(*graph*, *u*, *v*, *key*, *attr_dict*, ***attr*)
Adds an edge while preserving negative keys, and paying no respect to positive ones

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – The source BEL node
- **v** (*tuple*) – The target BEL node
- **key** (*int*) – The edge key. If less than zero, corresponds to an unqualified edge, else is disregarded
- **attr_dict** (*dict*) – The edge data dictionary
- **attr** (*dict*) – Edge data to assign via keyword arguments

pybel_tools.utils.**prepare_c3**(*data*, *y_axis_label='y'*, *x_axis_label='x'*)
Prepares C3 JSON for making a bar chart from a Counter

Parameters

- **data** (*Counter or dict or collections.defaultdict*) – A dictionary of {str: int} to display as bar chart
- **y_axis_label** (*str*) – The Y axis label
- **x_axis_label** (*str*) – X axis internal label. Should be left as default 'x')

Returns A JSON dictionary for making a C3 bar chart

Return type dict

pybel_tools.utils.**prepare_c3_time_series**(*data*, *y_axis_label='y'*, *x_axis_label='x'*)
Prepares C3 JSON for making a time series

Parameters

- **data** (*list*) – A list of tuples [(year, count)]
- **y_axis_label** (*str*) – The Y axis label
- **x_axis_label** (*str*) – X axis internal label. Should be left as default 'x')

Returns A JSON dictionary for making a C3 bar chart

Return type dict

pybel_tools.utils.**build_template_environment**(*here*)
Builds a custom templating enviroment so Flask apps can get data from lots of different places

Parameters **here** (*str*) – Give this the result of `os.path.dirname(os.path.abspath(__file__))`

Return type jinja2.Environment

`pybel_tools.utils.`**`build_template_renderer`**(*file*)
In your file, give this function the current file

Parameters **file** (*str*) – The location of the current file. Pass it `__file__` like in the example below.

```
>>> render_template = build_template_renderer(__file__)
```

`pybel_tools.utils.`**`calculate_betweenness_centality`**(*graph*, *k=200*)
Calculates the betweenness centrality over nodes in the graph. Tries to do it with a certain number of samples, but then tries a complete approach if it fails.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **k** (*int*) – The number of samples to use

Return type collections.Counter[tuple,float]

`pybel_tools.utils.`**`get_circulations`**(*t*)
Iterate over all possible circulations of an ordered collection (tuple or list)

Parameters **or list t** (*tuple*) –

Return type iter

`pybel_tools.utils.`**`canonical_circulation`**(*t*, *key=None*)
Get get a canonical representation of the ordered collection by finding its minimum circulation with the given sort key

Parameters

- **or list t** (*tuple*) –
- **key** – A function for sort

Returns The

`pybel_tools.utils.`**`get_version`**()
Gets the current PyBEL Tools version

Returns The current PyBEL Tools version

Return type str

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Bibliography

[DomingoFernandez2017] Domingo-Fernández, D., *et al* (2017). Multimodal mechanistic signatures for neurodegenerative diseases (NeuroMMSig): A web server for mechanism enrichment. Bioinformatics, 33(22), 3679–3681.

[HoytDomingoFernandez2018] Charles Tapley Hoyt, Daniel Domingo-Fernández, Nora Balzer, Anka Güldenpfennig, Martin Hofmann-Apitius; A systematic approach for identifying shared mechanisms in epilepsy and its comorbidities, Database, Volume 2018, 1 January 2018, bay050

[Catlett2013] Catlett, N. L., *et al* (2013). Reverse causal reasoning: applying qualitative causal knowledge to the interpretation of high-throughput data. BMC Bioinformatics, 14(1), 340.

Python Module Index

p

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W