pybel-tools Documentation

Release 0.4.3

Charles Tapley Hoyt

Oct 07, 2017

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.

CHAPTER 1

Links

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

Installation

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

Warning: Python2 and Windows are not thoroughly tested

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 .
```

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.

Cookbook

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)

```
$ python3 -m pybel_tools io convert -d ~/bms/aetionomy/
```

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

```
$ python3 -m pybel_tools ensure small_corpus -v
```

Large Corpus

```
$ python3 -m pybel_tools ensure large_corpus -v
```

Loading Other Resources

Gene Families

```
$ python3 -m pybel_tools ensure gene_families -v
```

Named Protein Complexes

```
$ python3 -m pybel_tools ensure named_complexes -v
```

Orthology Relations

Coming soon!

Uploading Precompiled BEL

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

```
$ python3 -m 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.

```
$ python3 -m pybel_tools io upload -p ~/bms/aetionomy/ -r
```

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*)
Returns 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

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_annotation_values_by_annotation**(*graph*)
Gets the set of values for each annotation used in a BEL graph

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

Returns A dictionary of {annotation key: set of annotation values}

Return type dict[str, 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.**get_annotation_values**(*graph*, *annotation*)
Get all values for the given annotation

Parameters

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

Returns A set of all annotation values

Return type set[str]

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.**get_all_relations**(*graph*, *u*, *v*)

Returns the set of all relations between a given pair of nodes

Parameters

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

Return type set[str]

pybel_tools.summary.**pair_is_consistent**(*graph*, *u*, *v*)

Returns 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_inconsistent_edges**(*graph*)
Returns an iterator over inconsistent edges

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

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.**get_tree_annotations**(*graph*)
Builds tree structure with annotation for a given graph

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

Returns The JSON structure necessary for building the tree box

Return type list[dict]

pybel_tools.summary.**relation_set_has_contradictions**(*relations*)
Returns 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 and the erroneous names from the same 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

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*)
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*, ***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**(*graph*, *u*, *v*, *k*, *d*)
Is the given relation causal?

pybel_tools.summary.**get_causal_out_edges**(*graph*, *node*)
Gets the out-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 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.**has_causal_out_edges**(*graph*, *node*)
Does the node have causal out edges?

Parameters

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

Returns If the node has causal out edges

Return type bool

pybel_tools.summary.**has_causal_in_edges**(*graph*, *node*)
Does the node have causal in edges?

Parameters

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

Returns If the node has causal in edges

Return type bool

pybel_tools.summary.**is_causal_source**(*graph*, *node*)
Is 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*)
Is the node 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*)
Is 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_variants**(*graph*)
Counts how many of each type of variant a graph has

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

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

Return type dict

pybel_tools.summary.**count_functions**(*graph*)
Counts the frequency of each function present in a graph

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

Returns A Counter from {function: frequency}

Return type collections.Counter

pybel_tools.summary.**get_functions**(*graph*)
Gets the set of all functions used in this graph

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

Returns A set of functions

Return type set[str]

pybel_tools.summary.**count_namespaces**(*graph*)
Counts the frequency of each namespace across all nodes (that have namespaces)

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

Returns A Counter from {namespace: frequency}

Return type collections.Counter

pybel_tools.summary.**get_namespaces**(*graph*)
Gets the set of all namespaces used in this graph

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

Returns A set of namespaces

Return type set[str]

pybel_tools.summary.**count_names**(*graph*)
Counts all names through the graph by the NAME tag in the nodes' data dictionaries.

This is useful to identify which nodes appear with the same name in multiple namespaces, or to identify variants

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

Returns A Counter from {names: frequency}

Return type collections.Counter

pybel_tools.summary.**get_names_by_namespace**(*graph*, *namespace*)
Get the set of all of the names in a given namespace that are in the graph

Parameters

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

Returns A set of names belonging to the given namespace that are in the given graph

Return type set[str]

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

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

Returns A set of namespaces that are included but not used

Return type set[str]

pybel_tools.summary.**get_names**(*graph*)
Get a dictionary of {namespace: set of names} present in the graph.

Roughly equivalent to:

```
>>> {namespace: get_names_by_namespace(graph, namespace) for namespace in get_
↪namespaces(graph)}
```

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

Returns A dictionary of {namespace: set of names}

Return type dict[str, set[str]]

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_pubmed_identifiers**(*graph*)

Gets the set of all PubMed identifiers cited in the construction of a graph

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

Returns A set of all PubMed identifiers cited in the construction of this graph

Return type set[str]

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='Subgraph'*)
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]]

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.

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.nodes_iter())

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**(*function*)
Builds a filter that only passes on nodes of the given function(s)

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

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

Return type types.FunctionType

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

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

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

Return type types.FunctionType

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

Parameters

- **function** (*str*) – A BEL function
- **or list[str] or tuple[str] or set[str] namespace** (*str*) – The namespace to serach by

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

Return type types.FunctionType

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

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

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

Return type types.FunctionType

pybel_tools.filters.node_filters.**data_contains_key_builder**(*key*)
Builds 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 filter (graph, node) -> bool

Return type types.FunctionType

pybel_tools.filters.node_filters.**data_missing_key_builder**(*key*)
Builds 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 types.FunctionType

pybel_tools.filters.node_filters.**node_has_cname**(*graph*, *node*)
Passes for nodes that have been annotated with a canonical name

pybel_tools.filters.node_filters.**node_missing_cname**(*graph*, *node*)
Fails for nodes that have been annotated with a canonical name

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.**node_has_molecular_activity**(*graph*, *node*)
Returns true if over any of the node's edges it has a molecular activity

Parameters

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

Returns If the node has a known molecular activity

Return type bool

pybel_tools.filters.node_filters.**node_is_degraded**(*graph*, *node*)
Returns true if over any of the node's edges it is degraded

Parameters

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

Returns If the node has a known degradation

Return type bool

pybel_tools.filters.node_filters.**node_is_translocated**(*graph*, *node*)
Returns true if over any of the node's edges it is transloated

Parameters

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

Returns If the node has a known translocation

Return type bool

pybel_tools.filters.node_filters.**node_is_upstream_leaf**(*graph*, *node*)
Returns if the node is an upstream leaf. An upstream leaf is defined as a node that has no in-edges, and exactly 1 out-edge.

Parameters

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

Returns If the node is an upstream leaf

Return type bool

pybel_tools.filters.node_filters.**node_has_pmod**(*graph*, *node*)
Checks if a node has a protein modification

Parameters

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

- **node** (*tuple*) – A BEL node

Returns If the node has a protein modification

Return type bool

pybel_tools.filters.node_filters.**node_has_gmod**(*graph*, *node*)
Checks if a node has a gene modification

Parameters

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

Returns If the node has a gene modification

Return type bool

pybel_tools.filters.node_filters.**node_has_hgvs**(*graph*, *node*)
Checks if a node has an HGVS variant

Parameters

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

Returns If the node has an HGVS variant

Return type bool

pybel_tools.filters.node_filters.**build_node_data_search**(*key*, *data_filter*)
Passes 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_filter** (*types.FunctionType*) – The filter to apply to the node data dictionary

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

Return type types.FunctionType

pybel_tools.filters.node_filters.**build_node_key_search**(*query*, *key*)
Builds 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

pybel_tools.filters.node_filters.**build_node_name_search**(*query*)
Searches nodes' names. Is a thin wrapper around *build_node_key_search()* with pybel.constants.NAME

pybel_tools.filters.node_filters.**build_node_cname_search**(*query*)
Searches nodes' canonical names. Is a thin wrapper around *build_node_key_search()*

pybel_tools.filters.node_filters.**iter_undefined_families**(*graph*, *namespace*)
Finds protein families from a given namespace (Such as SFAM) that aren't qualified by members

Parameters

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

Returns An iterator over nodes that don't

Return type iter[tuple]

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** (*types.FunctionType or list[types.FunctionType] or tuple[types.FunctionType]*) – A filter or list of filters

pybel_tools.filters.edge_filters.**edge_is_causal**(*graph*, *u*, *v*, *k*, *d*)
Only passes on causal edges, belonging to the set pybel.constants.CAUSAL_RELATIONS

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

Returns If the edge is a causal edge

Return type bool

pybel_tools.filters.edge_filters.**edge_has_author_annotation**(*graph*, *u*, *v*, *k*, *d*)
Passes for edges that have citations with authors

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

Returns Does the edge's citation data dictionary have authors included?

Return type bool

pybel_tools.filters.edge_filters.**edge_has_pubmed_citation**(*graph*, *u*, *v*, *k*, *data*)
Passes for edges that have PubMed citations

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **data** (*dict*) – The edge data dictionary

Returns Is the edge's citation from PUBMED?

Return type bool

pybel_tools.filters.edge_filters.**build_inverse_filter**(*edge_filter*)
Builds a filter that is the inverse of the given filter

Parameters **edge_filter** (*types.FunctionType*) – An edge filter function (graph, node, node, key, data) -> bool

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

Return type types.FunctionType

pybel_tools.filters.edge_filters.**build_annotation_value_filter**(*annotation*, *value*)
Builds a filter that only passes for edges that contain the given annotation and have the given value(s)

Parameters

- **annotation** (*str*) – The annotation to filter on
- **or iter[str] value** (*str*) – The value or values for the annotation to filter on

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

Return type types.FunctionType

pybel_tools.filters.edge_filters.**build_edge_data_filter**(*annotations*, *partial_match=True*)
Builds a filter that keeps edges whose data dictionaries are superdictionaries 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.

pybel_tools.filters.edge_filters.**build_annotation_dict_all_filter**(*annotations*, *partial_match=True*)
Builds a filter that keeps edges whose data dictionaries's annotations entry are superdictionaries 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`.

pybel_tools.filters.edge_filters.**build_annotation_dict_any_filter**(*annotations*)
Builds a filter that keeps edges whose data dictionaries's annotations entry contain any match to the target dictionary

Parameters **annotations** (*dict*) – The annotation query dict to match

pybel_tools.filters.edge_filters.**build_relation_filter**(*relations*)
Builds a filter that passes only for edges with the given relation

Parameters **relations** (*str or iter[str]*) – A relation or iterable of relations

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

Return type types.FunctionType

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

Parameters **pmids** (*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 types.FunctionType

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

Parameters **pmids** (*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 types.FunctionType

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

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

Returns An edge filter

Return type types.FunctionType

pybel_tools.filters.edge_filters.**edge_has_activity**(*graph*, *u*, *v*, *k*, *d*)
Checks if the edge contains an activity in either the subject or object

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

Returns If the edge contains an activity in either the subject or object

Return type bool

pybel_tools.filters.edge_filters.**edge_has_translocation**(*graph*, *u*, *v*, *k*, *d*)
Checks if the edge has a translocation in either the subject or object

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

Returns If the edge has a translocation in either the subject or object

Return type bool

pybel_tools.filters.edge_filters.**edge_has_degradation**(*graph*, *u*, *v*, *k*, *d*)

Checks if the edge contains a degradation in either the subject or object

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

Returns If the edge contains a degradation in either the subject or object

Return type bool

pybel_tools.filters.edge_filters.**edge_has_pathology_causal**(*graph*, *u*, *v*, *k*, *d*)

Returns if the subject of this edge is a pathology and participates in a causal relation where the object is not a pathology. These relations are generally nonsense.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key between the given nodes
- **d** (*dict*) – The edge data dictionary

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

Return type bool

Comparison Functions

pybel_tools.comparison.**graph_entities_equal**(*g*, *h*)

Tests that two graphs have the same nodes

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns Do the two graphs share the same set of nodes?

Return type bool

pybel_tools.comparison.**graph_topologically_equal**(*g*, *h*)
Tests that two graphs are topologically equal, defined by having the same nodes, and same connectivity

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns Do the graphs share the same set of nodes, and the same connectivity?

Return type bool

pybel_tools.comparison.**graph_relations_equal**(*g*, *h*)
Tests that two graphs are equal, defined by having the same nodes, same connections, and existence of same connection types

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns Do the two graphs share the same set of nodes, same connectivity, and types of connections?

Return type bool

pybel_tools.comparison.**graph_provenance_equal**(*g*, *h*)
Tests that two graphs are equal, defined by having the same nodes, same connections with same types, and same attributes exactly.

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns Do the two graphs share the same set of nodes, same connectivity, types of connections, evidences, and annotations?

Return type bool

pybel_tools.comparison.**graph_edges_intersection**(*g*, *h*)
Returns the set of node pairs that appear in both graphs

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns The set of edges shared between the two graphs

Return type set

pybel_tools.comparison.**graph_edges_difference**(*g*, *h*)
Returns the set of node pairs that appear in g but not h

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns The asymmetric difference between the edges in g and h

Return type set

pybel_tools.comparison.**graph_edges_symmetric_difference**(*g*, *h*)
Returns the set of edges that appear in only one of g or h

Parameters

- **g** (*pybel.BELGraph*) – A BEL Graph
- **h** (*pybel.BELGraph*) – Another BEL Graph

Returns The symmetric difference between the edges in g and h

Return type set

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*)
Induces a graph over the given nodes

Parameters

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

Returns A subgraph induced over the given nodes

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_edge_filter**(*graph*, *edge_filters*)
Induces a subgraph on all edges that pass the given filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **edge_filters** (*list or tuple or lambda*) – A predicate or list of predicates (graph, node, node, key, data) -> bool

Returns A BEL subgraph induced over the edges passing the given filters

Return type 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*)
Gets a BEL graph around the neighborhoods of the given nodes

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 pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_second_neighbors**(*graph*, *nodes*, *filter_pathologies=False*)
Gets a BEL graph around the neighborhoods of the given nodes, and expands to the neighborhood of those nodes

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 pybel.BELGraph

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

Induces 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** (*set[tuple]*) – A set of nodes over which to calculate shortest paths
- **cutoff** (*int*) – Depth to stop the shortest path search. Only paths of length <= cutoff are returned.
- **weight** (*str*) – Edge data key corresponding to the edge weight. If None, performs unweighted search

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

Return type pybel.BELGraph

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

Builds a new subgraph induced 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
- **value** (*str*) – The value 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*)

Returns the subgraph given an annotations filter.

Parameters

- **graph** – pybel.BELGraph graph: A BEL graph
- **annotations** (*dict*) – 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

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_data**(*graph*, *annotations*)

Returns the subgraph filtering for Citation, Evidence or Annotation in the edges.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **annotations** (*dict*) – Annotation filters (match all with pybel.utils.subdict_matches())

Returns A subgraph of the original BEL graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_pubmed**(*graph*, *pmids*)
Induces a subgraph over the edges retrieved from the given PubMed identifier(s)

Parameters

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

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraph_by_authors**(*graph*, *authors*)
Induces a subgraph 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*)
Runs pipeline query on graph with multiple subgraph 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

Returns A BEL Graph

Return type pybel.BELGraph

pybel_tools.selection.**get_subgraphs_by_annotation**(*graph*, *annotation*)
Stratifies the given graph into subgraphs based on the values for edges' annotations

Parameters

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

Returns A dictionary of {str value: BELGraph subgraph}

Return type dict[str, pybel.BELGraph]

pybel_tools.selection.**get_multi_causal_upstream**(*graph*, *nbunch*)
Gets the union of all the 2-level deep causal upstream subgraphs from the nbunch

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or list[tuple] nbunch** (*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*)
Gets the union of all of the 2-level deep causal downstream subgraphs from the nbunch

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **or list[tuple] nbunch** (*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=250*, *number_seed_nodes=5*)
Randomly picks a node from the graph, and performs a weighted random walk to sample the given number of edges around it

Parameters

- **graph** (*pybel.BELGraph*) –
- **number_edges** (*int*) – Maximum number of edges
- **number_seed_nodes** (*int*) – Number of nodes to start with (which likely results in different components in large graphs)

Return type pybel.BELGraph

pybel_tools.selection.**get_upstream_leaves**(*graph*)
Gets all leaves of the graph (with no incoming edges and only one outgoing edge)

See also:

`upstream_leaf_predicate()`

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

Returns An iterator over nodes that are upstream leaves

Return type iter

pybel_tools.selection.**get_unweighted_upstream_leaves**(*graph*, *key*)
Gets all leaves of the graph with no incoming edges, one outgoing edge, and without the given key in its data dictionary

See also:

`data_does_not_contain_key_builder()`

Parameters

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

Returns An iterable over leaves (nodes with an in-degree of 0) that don't have the given annotation

Return type iter

pybel_tools.selection.**get_gene_leaves**(*graph*)
Find all genes who have only one connection, that's a transcription to its RNA

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

pybel_tools.selection.**get_rna_leaves**(*graph*)
Find all RNAs who have only one connection, that's a translation to its protein

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

pybel_tools.selection.**get_leaves_by_type**(*graph*, *function=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
- **function** (*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

pybel_tools.selection.**get_nodes_in_all_shortest_paths**(*graph*, *nodes*, *weight=None*)
Gets all shortest paths from all nodes to all other nodes in the given list and returns the set of all nodes contained in those paths using `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
- **cutoff** (*int*) – Depth to stop the search. Only paths of length <= cutoff are returned.

- **weight** (*str*) – Edge data key corresponding to the edge weight. If none, uses unweighted search.

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

Return type set

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*)

Searches for nodes containing a given string

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.**search_node_cnames**(*graph*, *query*)

Searches for nodes whose canonical names contain 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 canonical 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]

Integration

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

class pybel_tools.integration.**NodeAnnotator**(*namespace*)
The base class for node annotators.

Parameters **or list[str] namespace** (*str*) – The name of the namespace or namespaces that this node annotator services

annotate(*graph*)
Annotates all nodes in this annotator's namespace

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

get_description(*name*)
Gets the description for the given name in this annotator's namespace.

get_label(*name*)
Gets the label for the given name in. If not overridden, uses each node's name as its label.

populate_by_graph(*graph*)
Optional hook for populating the annotator based on the nodes in a graph. Override this if your node annotator downloads data ahead of time, such as grouping requests to an external API.

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

class pybel_tools.integration.**HGNCAnnotator**(*preload=True*)
Annotates the labels and descriptions of Genes with HGNC identifiers using a mapping provided by HGNC and then the Entrez Gene Service.

Parameters **preload** (*bool*) – Should the data be pre-downloaded?

get_unpopulated_entrez(*entrez_ids*)
Gets the Entrez Gene Identifiers from this list that aren't already cached

load_hgnc_entrez_map()
Preloads the HGNC-Entrez map

map_entrez_ids(*entrez_ids*)
Maps a list of Entrez Gene Identifiers to HGNC Gene Symbols

map_hgnc(*hgnc_symbols*)
Maps a list of HGNC Gene Symbols to Entrez Gene Identifiers

populate(*entrez_ids*, *group_size=200*, *sleep_time=1*)
Download the descriptions from Entrez Gene Service for a given list of Entrez Gene Identifiers

Parameters

- **entrez_ids** (*iter*) – An iterable of Entrez Gene Identifiers
- **group_size** (*int*) – The number of entrez gene id's to send per query
- **sleep_time** (*int*) – The number of seconds to sleep between queries

populate_by_graph(*graph*)
Downloads the gene information only for genes in the given graph

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

populate_constrained(*hgnc_symbols*, *group_size=200*, *sleep_time=1*)
Downloads the gene information only for genes in the list of HGNC Gene Symbols

populate_unconstrained(*group_size=200*, *sleep_time=1*)
Downloads all descriptions for all Entrez Gene Identifiers

pybel_tools.integration.**overlay_data**(*graph*, *data*, *label*, *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** (*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*, *label*, *function*, *namespace*, *overwrite=False*, *impute=None*)
Overlays 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
- **data** (*dict*) – A dictionary of {name: data}

- **label** (*str*) – The annotation label to put in the node dictionary
- **function** (*str*) – The function of the keys in the data dictionary
- **namespace** (*str*) – The namespace of the keys in the data dictionary
- **overwrite** (*bool*) – Should old annotations be overwritten?
- **impute** – The value to use for missing data

Mutation

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

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

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **dict_of_sets_of_nodes** (*dict[tuple,set[tuple]]*) – A dictionary of {node: set of nodes}

pybel_tools.mutation.**build_central_dogma_collapse_dict**(*graph*)
Builds a dictionary to direct the collapsing on the central dogma

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

Returns A dictionary of {node: set of nodes}

Return type dict[tuple,set[tuple]]

pybel_tools.mutation.**build_central_dogma_collapse_gene_dict**(*graph*)
Builds a dictionary to direct the collapsing on the central dogma

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

Returns A dictionary of {node: set of PyBEL node tuples}

Return type dict[tuple,set[tuple]]

pybel_tools.mutation.**collapse_by_central_dogma**(*graph*)
Collapses all nodes from the central dogma (GENE, RNA, PROTEIN) to PROTEIN, or most downstream possible entity, in place. This function wraps *collapse_nodes()* and *build_central_dogma_collapse_dict()*.

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

Equivalent to:

```
>>> collapse_nodes(graph, build_central_dogma_collapse_dict(graph))
```

pybel_tools.mutation.**collapse_by_central_dogma_to_genes**(*graph*)
Collapses all nodes from the central dogma (pybel.constants.GENE, pybel.constants.RNA, pybel.constants.MIRNA, and pybel.constants.PROTEIN) to pybel.constants.GENE, in-place. This function wraps *collapse_nodes()* and *build_central_dogma_collapse_gene_dict()*.

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

Equivalent to:

```
>>> infer_central_dogma(graph)
>>> collapse_nodes(graph, build_central_dogma_collapse_gene_dict(graph))
```

pybel_tools.mutation.**collapse_by_central_dogma_to_genes_out_place**(*graph*)

Collapses all nodes from the central dogma (pybel.constants.GENE, pybel.constants.RNA, pybel.constants.MIRNA, and pybel.constants.PROTEIN) to pybel.constants.GENE, in-place. This function wraps *collapse_nodes()* and *build_central_dogma_collapse_gene_dict()*.

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

Return type pybel.BELGraph

Equivalent to:

```
>>> infer_central_dogma(graph)
>>> collapse_nodes(graph, build_central_dogma_collapse_gene_dict(graph))
```

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_by_central_dogma_to_genes()*

pybel_tools.mutation.**collapse_all_variants**(*graph*)

Collapses all hasVariant edges to the parent node, in place

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

pybel_tools.mutation.**collapse_all_variants_out_place**(*graph*)

Collapses all hasVariant edges to the parent node, not in place

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

Return type pybel.BELGraph

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.**opening_on_central_dogma**(*graph*)

Infers central dogmatic relations with *infer_central_dogma()* then successively prunes gene leaves then RNA leaves with *prune_central_dogma()* to connect disparate elements in a knowledge assembly

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

Equivalent to:

```
>>> infer_central_dogma(graph)
>>> prune_central_dogma(graph)
```

pybel_tools.mutation.**collapse_consistent_edges**(*graph*)

Collapses consistent edges together

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

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

pybel_tools.mutation.**remove_filtered_edges**(*graph*, *edge_filters*)
Removes edges passing the given edge filters

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **edge_filters** (*types.FunctionType or iter[types.FunctionType]*) – An edge filter or list of edge filters (graph, node, node, key, data)-> bool

Returns

pybel_tools.mutation.**remove_leaves_by_type**(*graph*, *function=None*, *prune_threshold=1*)
Removes 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
- **function** (*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

pybel_tools.mutation.**prune_central_dogma**(*graph*)
Prunes genes, then RNA, in place

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

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_upstream_causal_subgraph**(*graph*, *nbunch*)
Induces a subgraph from all of the upstream causal entities of the nodes in the nbunch

Parameters

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

Returns A BEL Graph

Return type pybel.BELGraph

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.**expand_node_neighborhood**(*universe*, *graph*, *node*)

Expands around the neighborhoods of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuples from the query graph

pybel_tools.mutation.**expand_nodes_neighborhoods**(*universe*, *graph*, *nodes*)

Expands around the neighborhoods of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **nodes** (*list[tuple]*) – A node tuples from the query graph

pybel_tools.mutation.**expand_all_node_neighborhoods**(*universe*, *graph*, *filter_pathologies=False*)

Expands the neighborhoods of all nodes in the given graph based on the universe graph.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **filter_pathologies** (*bool*) – Should expansion take place around pathologies?

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

Adds the upstream causal relations to the given subgraph

Parameters

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

Adds all of the members of the complexes in the subgraph that are in the original graph with appropriate pybel.constants.HAS_COMPONENT relationships, in place.

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 in the subgraph that are in the original graph with appropriate pybel.constants.HAS_COMPONENT relationships, in place.

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 in the subgraph that are in the original graph with appropriate pybel.constants.HAS_REACTANT and pybel.constants.HAS_PRODUCT relationships, respectively, in place.

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*, *function=None*)

Adds 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] function** (*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*)

Enriches 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
>>> pbt.mutation.expand_internal(universe, graph, edge_filters=pbt.filters.edge_
↪is_causal)
```

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

Adds the downstream causal relations to the given subgraph

Parameters

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

pybel_tools.mutation.**expand_node_predecessors**(*universe*, *graph*, *node*)
Expands around the predecessors of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuple from the query graph

pybel_tools.mutation.**expand_node_successors**(*universe*, *graph*, *node*)
Expands around the successors of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuples from the query graph

pybel_tools.mutation.**get_downstream_causal_subgraph**(*graph*, *nbunch*)
Induces a subgraph from all of the downstream causal entities of the nodes in the nbunch

Parameters

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

Returns A BEL Graph

Return type pybel.BELGraph

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.**infer_central_dogma**(*graph*)
Adds all RNA-Protein translations then all Gene-RNA transcriptions by applying `infer_central_dogmatic_translations()` then `infer_central_dogmatic_transcriptions()`

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

pybel_tools.mutation.**infer_missing_two_way_edges**(*graph*)
If a two way edge exists, and the opposite direction doesn't exist, add it to the graph

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*)
Adds the same edge, but in the opposite direction if not already present

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **u** (*tuple*) – A BEL node
- **v** (*tuple*) – A BEL node
- **k** (*int*) – The edge key

pybel_tools.mutation.**infer_missing_inverse_edge**(*graph*, *relations*)
Adds inferred edges based on pre-defined axioms

Parameters

- **graph** (*pybel.BELGraph*) – A BEL network
- **relations** (*str or iter[str]*) – single or iterable of relation names to add their inverse inferred edges

pybel_tools.mutation.**enrich_internal_unqualified_edges**(*graph*, *subgraph*)
Adds 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.**add_canonical_names**(*graph*, *replace=False*)
Adds a canonical name to each node's data dictionary if they are missing, in place.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **replace** (*bool*) – Should the canonical names be recalculated?

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.**add_identifiers**(*graph*)
Adds stable node and edge identifiers to the graph, in-place using the PyBEL node and edge hashes as a hexadecimal str.

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

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.**ensure_node_from_universe**(*universe*, *graph*, *node*, *raise_for_missing=False*)
Makes sure that the subgraph has the given node

Parameters

- **universe** (*pybel.BELGraph*) – The universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph
- **node** (*tuple*) – A BEL node
- **raise_for_missing** (*bool*) – Should an error be thrown if the given node is not in the universe?

pybel_tools.mutation.**remove_isolated_nodes**(*graph*)
Removes isolated nodes from the network

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

pybel_tools.mutation.**update_node_helper**(*universe*, *graph*)
Updates the nodes' data dictionaries from the universe

Parameters

- **universe** (*pybel.BELGraph*) – The universe of all knowledge
- **graph** (*pybel.BELGraph*) – The target BEL graph

Visualization

pybel_tools.visualization.**to_html**(*graph*, *color_map=None*)
Creates an HTML visualization for the given JSON representation of a BEL graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **color_map** (*dict[str,str]*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`

Returns HTML string representing the graph

Return type str

pybel_tools.visualization.**to_html_file**(*graph*, *file*, *color_map=None*)
Writes the HTML visualization to a file or file-like

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **color_map** (*dict[str,str]*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`
- **file** (*file*) – A writable file or file-like

pybel_tools.visualization.**to_html_path**(*graph*, *path*, *color_map=None*)
Writes the HTML visualization to a file specified by the file path

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **color_map** (*dict[str,str]*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`
- **path** (*str*) – The file path

pybel_tools.visualization.**to_jupyter**(*graph*, *width=1000*, *height=650*, *color_map=None*, *replace_cnames=False*)
Displays the BEL graph inline in a Jupyter notebook.

To use successfully, make run as the last statement in a cell inside a Jupyter notebook.

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **width** (*int*) – The width of the visualization window to render

- **height** (*int*) – The height of the visualization window to render
- **color_map** (*dict*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`

Returns An IPython notebook Javascript object

Return type `IPython.display.Javascript`

`pybel_tools.visualization.`**`to_jupyter_str`**(*graph*, *width=1000*, *height=650*, *color_map=None*, *replace_cnames=False*)

Returns the string to be javascript-ified by the Jupyter notebook function `IPython.display.Javascript`

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **width** (*int*) – The width of the visualization window to render
- **height** (*int*) – The height of the visualization window to render
- **color_map** (*dict*) – A dictionary from PyBEL internal node functions to CSS color strings like #FFEE00. Defaults to `default_color_map`

Returns The javascript string to turn into magic

Return type str

Stability

`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

Orthology Resolution Workflow

This module has tools for downloading and structuring gene orthology data from HGNC, RGD, and MGI

pybel_tools.orthology.**download_orthologies_from_hgnc**(*path*)
Downloads the full dump to the given path

Parameters **path** – output path

pybel_tools.orthology.**integrate_orthologies_from_hgnc**(*graph*, *lines=None*)
Adds orthology statements to graph using HGNC symbols, MGI IDs, and RGD IDs.

For MGI symbols and RGD symbols, use *integrate_orthologies_from_rgd()*

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **lines** (*list[str]*) –

pybel_tools.orthology.**integrate_orthologies_from_rgd**(*graph*, *path=None*)
Adds orthology statements to graph using HGNC symbols, MGI symbols, and RGD symbols.

For MGI IDs and RGD IDs, use *integrate_orthologies_from_hgnc()*

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **path** – optional path to local RGD_ORTHOLOGS.txt. Defaults to downloading directly from RGD FTP server with pandas

pybel_tools.orthology.**ortholog_filter**(*graph*, *u*, *v*, *k*, *d*)
Filters to keep only edges representing orthologies

pybel_tools.orthology.**collapse_orthologies**(*graph*)
Collapses all orthology relations.

Assumes: orthologies are annotated for edge (u,v) where u is the higher priority node

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

> **Warning:** This won't work for two way orthology annotations, so it's best to use *integrate_orthologies_from_rgd()* first

Subgraph Expansion Workflow

pybel_tools.mutation.expansion.**get_upstream_causal_subgraph**(*graph*, *nbunch*)
Induces a subgraph from all of the upstream causal entities of the nodes in the nbunch

Parameters

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

Returns A BEL Graph

Return type pybel.BELGraph

pybel_tools.mutation.expansion.**get_downstream_causal_subgraph**(*graph*, *nbunch*)
Induces a subgraph from all of the downstream causal entities of the nodes in the nbunch

Parameters

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

Returns A BEL Graph

Return type pybel.BELGraph

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*)

Adds all of the members of the complexes in the subgraph that are in the original graph with appropriate `pybel.constants.HAS_COMPONENT` relationships, in place.

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 in the subgraph that are in the original graph with appropriate pybel.constants.HAS_COMPONENT relationships, in place.

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 in the subgraph that are in the original graph with appropriate pybel.constants.HAS_REACTANT and pybel.constants.HAS_PRODUCT relationships, respectively, in place.

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*, *function=None*)
Adds 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] function** (*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*)
Enriches 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
>>> pbt.mutation.expand_internal(universe, graph, edge_filters=pbt.filters.edge_
↪is_causal)
```

pybel_tools.mutation.expansion.**expand_node_predecessors**(*universe*, *graph*, *node*)

Expands around the predecessors of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuple from the query graph

pybel_tools.mutation.expansion.**expand_node_successors**(*universe*, *graph*, *node*)

Expands around the successors of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuples from the query graph

pybel_tools.mutation.expansion.**expand_node_neighborhood**(*universe*, *graph*, *node*)

Expands around the neighborhoods of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **node** (*tuple*) – A node tuples from the query graph

pybel_tools.mutation.expansion.**expand_nodes_neighborhoods**(*universe*, *graph*, *nodes*)

Expands around the neighborhoods of the given node in the result graph by looking at the universe graph, in place.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **nodes** (*list[tuple]*) – A node tuples from the query graph

pybel_tools.mutation.expansion.**expand_all_node_neighborhoods**(*universe*, *graph*, *filter_pathologies=False*)

Expands the neighborhoods of all nodes in the given graph based on the universe graph.

Parameters

- **universe** (*pybel.BELGraph*) – The graph containing the stuff to add
- **graph** (*pybel.BELGraph*) – The graph to add stuff to
- **filter_pathologies** (*bool*) – Should expansion take place around pathologies?

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

Adds the upstream causal relations to the given subgraph

Parameters

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

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

Adds the downstream causal relations to the given subgraph

Parameters

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

Candidate Mechanism Generation

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.

pybel_tools.generation.**remove_unweighted_leaves**(*graph*, *key*)

Parameters

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

pybel_tools.generation.**is_unweighted_source**(*graph*, *node*, *key*)

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*)

Gets unannotated nodes on the periphery of the subgraph

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*)

Prunes unannotated nodes on the periphery of the subgraph

Parameters

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

pybel_tools.generation.**prune_mechanism_by_data**(*graph*, *key*)

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** (*str*) – The key in the node data dictionary representing the experimental data. If none, does not prune unannotated nodes after generation

Equivalent to:

```
>>> remove_unweighted_leaves(graph, key)
>>> remove_unweighted_sources(graph, key)
```

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** (*str*) – The key in the node data dictionary representing the experimental data. If none, does not prune unannotated nodes after generation

Returns A subgraph grown around the target BEL node

Return type pybel.BELGraph

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

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **key** (*str*) – The key in the node data dictionary representing the experimental data. If none, does not prune unannotated nodes after generation

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

Return type dict[tuple, pybel.BELGraph]

Heat Diffusion Workflow

An variant of the Network Perturbation Amplitude algorithm

pybel_tools.analysis.cmpa.**RESULT_LABELS = ['avg', 'stddev', 'normality', 'median', 'neighbors', 'subgraph_size']**
The columns in the score tuples

class pybel_tools.analysis.cmpa.**Runner**(*graph*, *target_node*, *key*, *tag=None*, *default_score=None*)
The NpaRunner class houses the data related to a single run of the CMPA analysis

Initializes the CMPA runner class

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **target_node** (*tuple*) – The BEL node that is the focus of this analysis
- **key** (*str*) – The key for the nodes' data dictionaries that points to their original experimental measurements
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA 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 CMPA 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 CMPA 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 CMPA 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()
Removes a random in-edge from the node with the lowest in/out degree ratio

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

score_leaves()
Calculates the CMPA score for all leaves

Returns The set of leaf nodes that were scored

Return type set

run()
Calculates CMPA 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()
Calculates CMPA 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

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()
Returns the final score for the target node

Returns The final score for the target node

Return type float

calculate_score(*node*)
Calculates 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 subgraph induced by all unscored nodes

Returns The remaining unscored BEL graph

Return type pybel.BELGraph

pybel_tools.analysis.cmpa.**multirun**(*graph*, *node*, *key*, *tag=None*, *default_score=None*, *runs=None*)
Runs CMPA multiple times and yields the NpaRunner object after each run has been completed

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **node** (*tuple*) – The BEL node that is the focus of this analysis
- **key** (*str*) – The key for the nodes' data dictionaries that points to their original experimental measurements
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

Returns An iterable over the runners after each iteration

Return type iter[NpaRunner]

pybel_tools.analysis.cmpa.**workflow**(*graph*, *node*, *key*, *tag=None*, *default_score=None*, *runs=None*)
Generates candidate mechanism and runs CMPA.

Parameters

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

- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

Returns A list of runners

Return type list[*Runner*]

pybel_tools.analysis.cmpa.**workflow_average**(*graph*, *node*, *key*, *tag=None*, *default_score=None*, *runs=None*)

Gets the average CMPA score over multiple runs.

This function is very simple, and can be copied to do more interesting statistics over the `NpaRunner` 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** (*str*) – The key for the nodes' data dictionaries that points to their original experimental measurements
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

Returns The average score for the target node

Return type float

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

Runs CMPA 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.CMPA on each candidate mechanism for multiple runs

4.Return all runner results

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*str*) – The key in the node data dictionary representing the experimental data
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

Returns A dictionary of {node: list of runners}

Return type dict

pybel_tools.analysis.cmpa.**workflow_all_average**(*graph*, *key*, *tag=None*, *default_score=None*, *runs=None*)

Runs CMPA 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.CMPA on each candidate mechanism for multiple runs

4.Report average CMPA scores for each candidate mechanism

Parameters

- **graph** (*pybel.BELGraph*) – A BEL graph
- **key** (*str*) – The key in the node data dictionary representing the experimental data
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

Returns A dictionary of {node: upstream causal subgraph}

Return type dict

pybel_tools.analysis.cmpa.**calculate_average_scores_on_subgraphs**(*candidate_mechanisms*, *key*, *tag=None*, *default_score=None*, *runs=None*)

Calculates the scores over precomputed candidate mechanisms

Parameters

- **candidate_mechanisms** (*dict[tuple, pybel.BELGraph]*) – A dictionary of {tuple node: pybel.BELGraph candidate mechanism}
- **key** (*str*) – The key in the node data dictionary representing the experimental data
- **tag** (*str*) – The key for the nodes' data dictionaries where the CMPA scores will be put. Defaults to 'score'
- **default_score** (*float*) – The initial CMPA score for all nodes. This number can go up or down.
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

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

Return type dict[tuple, tuple]

Example Usage:

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

pybel_tools.analysis.cmpa.**calculate_average_score_by_annotation**(*graph*, *key*, *annotation*, *runs=None*)

For each subgraph induced over the edges matching the annotation, calculate the average CMPA score for all of the contained biological processes

Assumes you haven't done anything yet

1. Generates biological process upstream candidate mechanistic subgraphs with `generate_bioprocess_mechanisms()`
2. Calculates scores for each subgraph with *calculate_average_scores_on_subgraphs()*
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
- **key** (*str*) – The key in the node data dictionary representing the experimental data
- **annotation** (*str*) – A BEL annotation
- **runs** (*int*) – The number of times to run the CMPA algorithm. Defaults to 1000.

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.cmpa import calculate_average_score_by_annotation
>>> graph = pybel.from_path(...)
>>> key = ...
>>>
>>> scores = calculate_average_score_by_annotation(graph, key, 'Subgraph')
```

Pipeline Builder

This module assists in running complex workflows on BEL graphs.

Example Pipeline #1

This example shows a pipeline that acquires a subgraph and finds possible additions to the subgraph.

```
>>> network = ...
>>> example = Pipeline()
>>> example.append('get_subgraph_by_annotation_value', 'Subgraph', 'Blood vessel␣
↪dilation subgraph')
>>> example.append('enrich_unqualified')
>>> example.append('infer_central_dogma')
>>> example.append('expand_periphery')
>>> result = example.run(network)
```

Example Pipeline #2

This example shows how additional data can be integrated into a graph.

```
>>> from pybel.constants import PROTEIN
>>> network = ...
>>> example = Pipeline()
>>> example.append('infer_central_dogma')
>>> example.append('enrich_unqualified')
>>> example.append('infer_central_dogma')
>>> example.append('expand_periphery')
>>> example.append('expand_nodes_neighborhoods', [(PROTEIN, 'HGNC', 'AKT1'), (PROTEIN,
↪ 'HGNC', 'AKT2')])
>>> result = example.run(network)
```

Example Pipeline #3

This example shows how the results from multiple pipelines can be combine.

```
>>> network = ...
>>> pipeline_a = Pipeline()
>>> pipeline_a.append('get_subgraph_by_annotation_value', 'Subgraph', 'Blood vessel␣
↪dilation subgraph')
>>> pipeline_b = Pipeline()
>>> pipeline_b.append('get_subgraph_by_annotation_value', 'Subgraph', 'Tau protein␣
↪subgraph')
>>> pipeline_c = Pipeline.union(pipeline_a, pipeline_b)
>>> result = pipeline_c.run(network)
```

pybel_tools.pipeline.**in_place_mutator**(*f*)
A function decorator to inform the Pipeline how to handle a function

pybel_tools.pipeline.**uni_in_place_mutator**(*f*)
A function decorator to inform the Pipeline how to handle a function

pybel_tools.pipeline.**uni_mutator**(*f*)
A function decorator to inform the Pipeline how to handle a function

pybel_tools.pipeline.**mutator**(*f*)
A function decorator to inform the Pipeline how to handle a function

pybel_tools.pipeline.**splitter**(*f*)
A function decorator that signifies a function that takes in a graph and returns a dictionary of keys to graphs

Parameters **f** (*types.FunctionType*) – A function

Return type types.FunctionType

class pybel_tools.pipeline.**Pipeline**(*protocol=None*, *universe=None*)
Builds and runs analytical pipelines on BEL graphs

Parameters

- **protocol** (*list[dict]*) – The list of dictionaries describing how to mutate/filter a network
- **universe** (*pybel.BELGraph*) – The entire set of known knowledge to draw from

has_function(*name*)
Checks if a function is a valid pipeline function

Parameters **name** (*str*) – The name of the function

Return type bool

get_function(*name*)
Wraps a function with the universe and in-place

Parameters **name** (*str*) – The name of the function

Return type types.FunctionType

append(*name*, **args*, ***kwargs*)
Adds a function and arguments to the pipeline

Parameters

- **name** (*str*) – The name of the function
- **args** – The positional arguments to call in the function
- **kwargs** – The keyword arguments to call in the function

Returns This pipeline for fluid query building

Return type *Pipeline*

extend(*pipeline*)
Adds another pipeline to the end of the current pipeline

Parameters **pipeline** (Pipeline) – Another pipeline

Returns This pipeline for fluid query building

Return type *Pipeline*

run(*graph*, *universe=None*, *in_place=True*)
Runs the contained protocol on a seed graph

Parameters

- **graph** (*pybel.BELGraph*) – The seed BEL graph
- **universe** (*pybel.BELGraph*) – Allows just-in-time setting of the universe in case it wasn't set before. Defaults to the given network.
- **in_place** (*bool*) – Should the graph be copied before applying the algorithm?

Returns The new graph is returned if not applied in-place

wrap_universe(*f*)
Takes a function that needs a universe graph as the first argument and returns a wrapped one

static **wrap_in_place**(*f*)
Takes a function that doesn't return the graph and returns the graph

to_json()
Gives this pipeline as json

Return type list[dict]

to_jsons()
Gives this pipeline as a JSON string

Return type str

dump_json(*file*)
Dumps this protocol to a file in JSON

static from_json(*d*)
Loads a pipeline from a JSON object

Parameters **d** (*list[dict]*) –

Returns The pipeline represented by the JSON

Return type *Pipeline*

static from_jsons(*s*)
Loads a pipeline from the JSON in a string

Parameters **s** (*str*) – A string containing the JSON representing a pipeline

Return type *Pipeline*

static from_json_file(*file*)
Loads a protocol from JSON contained in file using *Pipeline.from_json()*.

Returns The pipeline represented by the JSON in the file

Return type *Pipeline*

static union(**pipelines*)
Takes the union of multiple pipelines

Returns The union of the results from multiple pipelines

Return type *Pipeline*

static intersection(**pipelines*)
Takes the intersection of the results from multiple pipelines

Returns The intersection of results from multiple pipelines

Return type *Pipeline*

Database Service

class pybel_tools.api.**DatabaseService**(*manager*, *autocache=False*)
The dictionary service contains functions that implement the PyBEL API with a in-memory backend using dictionaries.

Parameters **manager** (*pybel.manager.Manager*) – A cache manager

networks = None
dictionary of {int id: BELGraph graph}

hash_to_node_cache = None
dictionary of {node hash: tuple node}

bel_id = None
dictionary of {str BEL: node hash}

id_bel = None
dictionary of {node hash: BEL}

universe = None
The complete graph of all knowledge stored in the cache

node_degrees = None
A dictionary from {int id: {tuple node: int degree}}

overlap_cache = None
A dictionary from {int network_id: {int network_id: float tanimoto node overlap}}

clear()
Clear the cache and start over

relabel_nodes_to_identifiers(*graph*, *copy=True*)
Relabels all nodes by their identifiers, in place. This function is a thin wrapper around `networkx.relabel.relabel_nodes()`

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **copy** (*bool*) – Copy the graph first?

Return type pybel.BELGraph

cache_networks(*force_reload=False*, *eager=False*, *maintain_universe=True*, *infer_origin=False*)
This function needs to get all networks from the graph cache manager and make a dictionary

Parameters

- **force_reload** (*bool*) – Should all graphs be reloaded even if they have already been cached?
- **eager** (*bool*) – Should difficult to preload features be calculated?
- **maintain_universe** (*bool*) –
- **infer_origin** (*bool*) – Should the central dogma be inferred for proteins, RNA, and miRNA?

get_graph_by_id(*network_id=None*)
Gets a network by its ID or super network if identifier is not specified

Parameters **network_id** (*int*) – The internal ID of the network to get

Returns A BEL Graph

Return type pybel.BELGraph

get_graphs_by_ids(*network_ids*)
Gets a list of networks given the ids

Parameters **network_ids** (*list[int]*) – A list of network identifiers

Return type list[pybel.BELGraph]

get_graph_by_ids(*network_ids*)
Gets a networks by a list of database identifiers

Parameters **network_ids** (*list[int]*) – A list of network identifiers

Return type pybel.BELGraph

get_node_tuple_by_hash(*node_hash*)
Returns the node tuple based on the node id

Parameters **node_hash** – The node's identifier

Returns A PyBEL node tuple

Return type tuple

get_nodes_by_hashes(*node_hashes*)
Gets a list of node tuples from a list of ids

Parameters **node_hashes** (*list*) – A list of node identifiers

Return type list[tuple]

get_nodes_containing_keyword(*keyword*)
Gets a list with all cnames that contain a certain keyword adding to the duplicates their function

Parameters **keyword** (*str*) – Search for nodes whose cnames have this as a substring

Return type list[dict]

get_pubmed_containing_keyword(*keyword*)
Gets a list of PubMed identifiers that contain a certain keyword

Parameters **keyword** (*str*) – Search for PubMed identifiers who have this as a substring

Return type list[str]

get_authors_containing_keyword(*keyword*, *use_cache=True*)
Gets a list with authors that contain a certain keyword

Parameters

- **keyword** (*str*) – Search for authors whose names have this as a substring
- **use_cache** (*bool*) – If true, look up by cache, else look up in database

Return type list[str]

get_cname_by_node_hash(*node_hash*)
Gets the canonical name of a node

Parameters **node_hash** – A BEL node identifier

Return type str

get_cname(*node*)
Gets the canonical name of a node

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

Return type str

get_top_degree(*network_id*, *count=20*)
Gets the nodes with the highest degrees

Parameters

- **network_id** (*int*) – The network database identifier
- **count** (*int*) – The number of top degree nodes to get

Return type dict[str,int]

get_top_pathologies
Gets the top most frequent pathologies mentioned in a graph

Parameters

- **network_id** (*int*) – The network database identifier
- **count** (*int*) – The number of most frequently mentioned pathologies to get

Return type dict[str,int]

get_node_overlap(*network_id*)
Calculates overlaps to all other networks in the database

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

Returns A dictionary from {int network_id: float similarity} for this network to all other networks

Return type collections.Counter[int,float]

get_annotations_containing_keyword(*keyword*)
Gets annotation/value pairs for values for whom the search string is a substring

Parameters **keyword** (*str*) – Search for annotations whose values have this as a substring

Return type list[dict[str,str]]

forget_network(*network_id*)
Removes all cached data from the given network id

IO Utilities

Utilities for loading and exporting BEL graphs

pybel_tools.ioutils.**load_paths**(*paths*, *connection=None*)
Loads a group of BEL graphs.

Internally, this function uses a shared pybel.parser.MetadataParser to cache the definitions more efficiently.

Parameters

- **paths** (*iter[str]*) – An iterable over paths to BEL scripts
- **connection** (*None or str or pybel.manager.Manager*) – A custom database connection string

Returns A BEL graph comprised of the union of all BEL graphs produced by each BEL script

Return type pybel.BELGraph

pybel_tools.ioutils.**load_directory**(*directory*, *connection=None*)
Compiles all BEL scripts in the given directory and returns as a merged BEL graph using *load_paths()*

Parameters

- **directory** (*str*) – A path to a directory
- **connection** (*None or str or pybel.manager.Manager*) – A custom database connection string

Returns A BEL graph comprised of the union of all BEL graphs produced by each BEL script

Return type pybel.BELGraph

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*, *connection=None*, *upload=False*, *pickle=False*, *infer_central_dogma=True*, *enrich_citations=False*, *enrich_genes=False*, *enrich_go=False*, *send=False*, *version_in_path=False*, ***kwargs*)

Recursively parses and either uploads/pickles graphs in a given set of files

Parameters

- **paths** (*iter[str]*) – The paths to convert
- **connection** (*None or str or pybel.manager.Manager*) – The connection
- **upload** (*bool*) – Should the networks be uploaded to the cache?
- **pickle** (*bool*) – Should the networks be saved as pickles?
- **infer_central_dogma** (*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?
- **version_in_path** (*bool*) – Add the current pybel version to the pathname
- **kwargs** – Parameters to pass to `pybel.from_path()`

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

Recursively parses and either uploads/pickles graphs in a given directory and sub-directories

Parameters

- **directory** (*str*) – The directory to look through
- **connection** (*None or str or pybel.manager.Manage.*) – The connection
- **upload** (*bool*) – Should the networks be uploaded to the cache?
- **pickle** (*bool*) – Should the networks be saved as pickles?
- **infer_central_dogma** (*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*, *connection=None*, *exclude_directory_pattern=None*)

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

Parameters

- **directory** (*str*) – the directory to traverse
- **connection** (*None or str or pybel.manage.Manager*) – A connection string or manager
- **exclude_directory_pattern** (*str*) – Any directory names to exclude

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

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
- **directory** (*str*) – A directory to output the pickles
- **annotation** (*str*) – An annotation to split by. Suggestion: `Subgraph`

`pybel_tools.ioutils.`**to_pybel_web**(*network*, *service=None*)

Sends a graph to the receiver service and returns the `requests` response object

Parameters

- **network** (*pybel.BELGraph*) – A BEL network
- **service** (*str*) – The location of the PyBEL web server. Defaults to `DEFAULT_SERVICE_URL`

Returns The response object from `requests`

Return type requests.Response

Document Utilities

Creating Definition Documents

Utilities for serializing to BEL namespace and BEL annotation files

`pybel_tools.definition_utils.`**make_namespace_header**(*name*, *keyword*, *domain*, *query_url=None*, *description=None*, *species=None*, *version=None*, *created=None*)

Makes the `[Namespace]` section of a BELNS file

Parameters

- **name** (*str*) – The namespace name
- **keyword** (*str*) – Preferred BEL Keyword, maximum length of 8

- **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`
- **query_url** (*str*) – HTTP URL to query for details on namespace values (must be valid URL)
- **description** (*str*) – Namespace description
- **species** (*str*) – Comma-separated list of species taxonomy id's
- **version** (*str*) – Namespace version. Defaults to current date in `YYYYMMDD` format.
- **created** (*str*) – Namespace public timestamp, ISO 8601 datetime

Returns An iterator over the lines of the `[Namespace]` section of a BELNS file

Return type iter[str]

pybel_tools.definition_utils.**make_author_header**(*name=None*, *contact=None*, *copyright_str=None*)

Makes the `[Author]` section of a BELNS file

Parameters

- **name** (*str*) – Namespace's authors
- **contact** (*str*) – Namespace author's contact info/email address
- **copyright_str** (*str*) – Namespace's copyright/license information. Defaults to `Other/Proprietary`

Returns An iterable over the lines of the `[Author]` section of a BELNS file

Return type iter[str]

pybel_tools.definition_utils.**make_citation_header**(*name*, *description=None*, *url=None*, *version=None*, *date=None*)

Makes the `[Citation]` section of a BEL config file.

Parameters

- **name** (*str*) – Citation name
- **description** (*str*) – Citation description
- **url** (*str*) – URL to more citation information
- **version** (*str*) – Citation version
- **date** (*str*) – Citation publish timestamp, ISO 8601 Date

Returns An iterable over the lines of the `[Citation]` section of a BEL config file

Return type iter[str]

pybel_tools.definition_utils.**make_properties_header**(*case_sensitive=True*, *delimiter='|'*, *cacheable=True*)

Makes the `[Processing]` section of a BEL config file.

Parameters

- **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?

Returns An iterable over the lines of the `[Processing]` section of a BEL config file

Return type iter[str]

pybel_tools.definition_utils.**write_namespace**(*namespace_name*, *namespace_keyword*, *namespace_domain*, *author_name*, *citation_name*, *values*, *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*, *file=None*, *value_prefix=''*, *sort_key=None*)

Writes a BEL namespace (BELNS) to a file

Parameters

- **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
- **values** (*iter[str]*) – An iterable of values (strings)
- **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** (*str*) – The encoding for the elements in this namespace. See `pybel.constants.belns_encodings`
- **file** (*file*) – A writable file or file-like
- **value_prefix** (*str*) – a prefix for each name
- **sort_key** – A function to sort the values with `sorted()`. Give `False` to not sort

pybel_tools.definition_utils.**make_annotation_header**(*keyword*, *description=None*, *usage=None*, *version=None*, *created=None*)

Makes the `[AnnotationDefinition]` section of a BELANNO file

Parameters

- **keyword** (*str*) – Preferred BEL Keyword, maximum length of 8
- **description** (*str*) – A description of this annotation
- **usage** (*str*) – How to use this annotation
- **version** (*str*) – Namespace version. Defaults to date in `YYYYMMDD` format.
- **created** (*str*) – Namespace public timestamp, ISO 8601 datetime

Returns A iterator over the lines for the `[AnnotationDefinition]` section

Return type iter[str]

pybel_tools.definition_utils.**write_annotation**(*keyword*, *values*, *citation_name*, *description=None*, *usage=None*, *version=None*, *created=None*, *author_name=None*, *author_copyright=None*, *author_contact=None*, *case_sensitive=True*, *delimiter='|'*, *cacheable=True*, *file=None*, *value_prefix=''*)

Writes a BEL annotation (BELANNO) to a file

Parameters

- **keyword** (*str*) – The annotation keyword
- **str] values** (*dict[str,*) – A dictionary of {name: label}
- **citation_name** (*str*) – The citation name
- **description** (*str*) – A description of this annotation
- **usage** (*str*) – How to use this annotation
- **version** (*str*) – The version of this annotation. Defaults to date in `YYYYMMDD` format.
- **created** (*str*) – The annotation's public timestamp, ISO 8601 datetime
- **author_name** (*str*) – The author's name
- **author_copyright** (*str*) – The copyright information for this annotation. Defaults to `Other/Proprietary`
- **author_contact** (*str*) – The contact information for the author of this annotation.
- **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?
- **file** (*file*) – A writable file or file-like
- **value_prefix** (*str*) – An optional prefix for all values

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.

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

```
>>> 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.**check_cacheable**(*config*)

Checks the config returned by `pybel.utils.get_bel_resource()` to determine if the resource should be cached.

If cannot be determined, returns `False`

Parameters **config** (*dict*) – A configuration dictionary representing a BEL resource

Returns Should this resource be cached

Return type bool

Creating Knowledge Documents

Utilities to merge multiple BEL documents on the same topic

pybel_tools.document_utils.**merge**(*output_path*, *input_paths*, *merged_name=None*, *merged_contact=None*, *merged_description=None*, *merged_author=None*)

Merges multiple BEL documents and maintains author information in comments

Steps:

1. Load all documents
2. Identify document metadata information and ns/annot defs
3. Postpend all statement groups with "- {author email}" and add comments with document information

Parameters

- **output_path** (*str*) – Path to file to write merged BEL document
- **input_paths** (*iter[str]*) – List of paths to input BEL document files
- **merged_name** (*str*) – name for combined document
- **merged_contact** (*str*) – contact information for combine document
- **merged_description** (*str*) – description of combine document

pybel_tools.document_utils.**make_document_metadata**(*name*, *contact*, *description*, *authors*, *version=None*, *copyright=None*, *licenses=None*)

Builds a list of lines for the document metadata section of a BEL document

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 date in YYYYMMDD format.
- **copyright** (*str*) – Copyright information about this document
- **licenses** (*str*) – The license applied to this document

Returns An iterator over the lines for the document metadata section

Return type iter[str]

pybel_tools.document_utils.**make_document_namespaces**(*namespace_dict=None*, *namespace_patterns=None*)

Builds a list of lines for the namespace definitions

Parameters

- **namespace_dict** (*dict[str,str]*) – dictionary of {str name: str URL} of namespaces
- **namespace_patterns** (*dict[str,str]*) – A dictionary of {str name: str regex}

Returns An iterator over the lines for the namespace definitions

Return type iter[str]

pybel_tools.document_utils.**make_document_annotations**(*annotation_dict=None*, *annotation_patterns=None*)

Builds a list of lines for the annotation definitions

Parameters

- **annotation_dict** (*dict[str,str]*) – A dictionary of {str name: str URL} of annotations
- **annotation_patterns** (*dict[str,str]*) – A dictionary of {str name: str regex}

Returns An iterator over the lines for the annotation definitions

Return type iter[str]

pybel_tools.document_utils.**make_pubmed_abstract_group**(*pmids*)

Builds a skeleton for the citations' statements

Parameters **pmids** (*iter[str] or iter[int]*) – A list of PubMed identifiers

Returns An iterator over the lines of the citation section

Return type iter[str]

pybel_tools.document_utils.**get_entrez_gene_data**(*entrez_ids*)

Gets gene info from Entrez

pybel_tools.document_utils.**make_pubmed_gene_group**(*entrez_ids*)
Builds a skeleton for gene summaries

Parameters **entrez_ids** (*list[str]*) – A list of entrez id's to query the pubmed service

Returns An iterator over statement lines for NCBI entrez gene summaries

Return type iter[str]

pybel_tools.document_utils.**write_boilerplate**(*document_name*, *contact*, *description*, *authors*, *version=None*, *copyright=None*, *licenses=None*, *namespace_dict=None*, *namespace_patterns=None*, *annotations_dict=None*, *annotations_patterns=None*, *pmids=None*, *entrez_ids=None*, *file=None*)
Writes a boilerplate BEL document, with standard document metadata, definitions. Optionally, if a list of PubMed identifiers are given, the citations and abstracts will be written for each.

Parameters

- **document_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
- **str] namespace_dict** (*dict[str,*) – an optional dictionary of {str name: str URL} of namespaces
- **str] namespace_patterns** (*dict[str,*) – An optional dictionary of {str name: str regex} namespaces
- **str] annotations_dict** (*dict[str,*) – An optional dictionary of {str name: str URL} of annotations
- **str] annotations_patterns** (*dict[str,*) – An optional dictionary of {str name: str regex} 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

Resource Utilities

This file contains the curated resource dictionary

pybel_tools.resources.**get_arty_namespace_url**(*namespace*, *version*)
Gets a BEL namespace file from artifactory given the name and version

pybel_tools.resources.**get_arty_annotation_url**(*module_name*, *version*)
Gets a BEL annotation file from artifactory given the name and version

pybel_tools.resources.**get_arty_knowledge_url**(*module_name*, *version*)
Gets a BEL knowledge file from artifactory given the name and version

pybel_tools.resources.**get_today_arty_namespace**(*module_name*)
Gets the right name for the next version of the namespace

pybel_tools.resources.**get_today_arty_annotation**(*module_name*)
Gets the right name for the next version of the annotation

pybel_tools.resources.**get_arty_auth**()
Gets the arty authentication tuple from the environment variables ARTY_USERNAME and ARTY_PASSWORD, respectively.

Return type tuple[str]

pybel_tools.resources.**get_namespace_history**(*module_name*)
Gets the Artifactory path for a namespace module

Parameters **module_name** (*str*) – The name of the namespace module

Return type artifactory.ArtifactoryPath

pybel_tools.resources.**get_annotation_history**(*module_name*)
Gets the Artifactory path for an annotation module

Parameters **module_name** (*str*) – The name of the annotation module

Return type artifactory.ArtifactoryPath

pybel_tools.resources.**get_knowledge_history**(*module_name*)
Gets the Artifactory path for a knowledge module

Parameters **module_name** (*str*) – The name of the knowledge module

Return type artifactory.ArtifactoryPath

pybel_tools.resources.**get_latest_arty_namespace**(*module_name*)
Gets the latest path for this BEL namespace module. For historical reasons, some of these are not the same as the keyword. For example, the module name for HGNC is hgnc-human-genes due to the Selventa nomenclature. See https://arty.scai.fraunhofer.de/artifactory/bel/namespace/ for the entire manifest of available namespaces.

Parameters **module_name** (*str*) – The BEL namespace module name

Returns The URL of the latest version of this namespace

Return type str

pybel_tools.resources.**get_latest_arty_annotation**(*module_name*)
Gets the latest path for this BEL annotation module

Parameters **module_name** (*str*) – The BEL annotation module name

Returns The URL of the latest version of this annotation

Return type str

pybel_tools.resources.**get_latest_arty_knowledge**(*module_name*)
Gets the latest path for this BEL annotation module

Parameters **module_name** (*str*) – The BEL knowledge module name

Returns The URL of the latest version of this knowledge document

Return type str

pybel_tools.resources.**deploy_namespace**(*filename*, *module_name*, *hash_check=True*, *auth=None*)
Deploys a file to the Artifactory BEL namespace cache

Parameters

- **filename** (*str*) – The physical path
- **module_name** (*str*) – The name of the module to deploy to
- **hash_check** (*bool*) – Ensure the hash is unique before deploying
- **auth** (*tuple[str]*) – A pair of (str username, str password) to give to the auth keyword of the constructor of `artifactory.ArtifactoryPath`. Defaults to the result of *get_arty_auth()*.

Returns The resource path, if it was deployed successfully, else none.

Return type str

pybel_tools.resources.**deploy_knowledge**(*filename*, *module_name*, *auth=None*)
Deploys a file to the Artifactory BEL knowledge cache

Parameters

- **filename** (*str*) – The physical file path
- **module_name** (*str*) – The name of the module to deploy to
- **auth** (*tuple[str]*) – A pair of (str username, str password) to give to the auth keyword of the constructor of `artifactory.ArtifactoryPath`. Defaults to the result of *get_arty_auth()*.

Returns The resource path, if it was deployed successfully, else none.

Return type str

pybel_tools.resources.**deploy_annotation**(*filename*, *module_name*, *hash_check=True*, *auth=None*)
Deploys a file to the Artifactory BEL annotation cache

Parameters

- **filename** (*str*) – The physical file path
- **module_name** (*str*) – The name of the module to deploy to
- **hash_check** (*bool*) – Ensure the hash is unique before deploying

- **auth** (*tuple[str]*) – A pair of (str username, str password) to give to the auth keyword of the constructor of `artifactory.ArtifactoryPath`. Defaults to the result of *get_arty_auth()*.

Returns The resource path, if it was deployed successfully, else none.

Return type str

`pybel_tools.resources.`**deploy_directory**(*directory*, *auth=None*)
Uploads all stuff from a directory to artifactory

Parameters

- **directory** (*str*) – the path to a directory
- **auth** (*tuple[str]*) – A pair of (str username, str password) to give to the auth keyword of the constructor of `artifactory.ArtifactoryPath`. Defaults to the result of *get_arty_auth()*.

Utilities

This module contains functions useful throughout PyBEL Tools

`pybel_tools.utils.`**multidict_list**(*it*)
Collects a list of pairs with a group by into a list

Parameters **it** (*iter[tuple[X,Y]]*) – An iterable of pairs where the first element is hashable

Return type dict[X,list[Y]]

`pybel_tools.utils.`**multidict_set**(*it*)
Collects a list of pairs with a group by into a set

Parameters **it** (*iter[tuple[X,Y]]*) – An iterable of pairs where the first and second elements are hashable

Return type dict[X,set[Y]]

`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*)
Iterates over the edge data dictionaries

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

Returns An iterator over the edge dictionaries in the graph

Return type iter

`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.`**get_value_sets**(*dict_of_iterables*)
Takes a dictionary of lists/iterables/counters and gets the sets of the values

Parameters **dict_of_iterables** (*dict or collections.defaultdict*) – A dictionary of lists

Returns A dictionary of {key: set of 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.**check_has_annotation**(*data*, *key*)
Checks that ANNOTATION is included in the data dictionary and that the key is also present

Parameters

- **data** (*dict*) – The data dictionary from a BELGraph's edge
- **key** (*str*) – An annotation key

Returns If the annotation key is present in the current data dictionary

Return type bool

For example, it might be useful to print all edges that are annotated with 'Subgraph':

```
>>> from pybel import BELGraph
>>> graph = BELGraph()
>>> ...
>>> for u, v, data in graph.edges_iter(data=True):
>>>     if not check_has_annotation(data, 'Subgraph')
>>>         continue
>>>     print(u, v, data)
```

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.**all_edges_iter**(*graph*, *u*, *v*)
Lists all edges between the given nodes

Parameters

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

Returns A list of (node, node, key)

Return type list[tuple]

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.**citation_to_tuple**(*citation*)
Converts a citation dictionary to a tuple. Can be useful for sorting and serialization purposes

Parameters **citation** (*dict*) – A citation dictionary

Returns A citation tuple

Return type tuple

pybel_tools.utils.**is_edge_consistent**(*graph*, *u*, *v*)
Checks if all edges between two nodes 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 all edges from the source to target node have the same relation

Return type bool

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.**safe_add_edges**(*graph*, *edges*)
Adds an iterable of edges to the graph

Parameters

- **graph** (*pybel.BELGraph*) – A BEL Graph
- **edges** (*iter[tuple*) – An iterable of 4-tuples of (source, target, key, data)

pybel_tools.utils.**load_differential_gene_expression**(*data_path*, *gene_symbol_column='Gene.symbol'*, *logfc_column='logFC'*)
Quick and dirty loader for differential gene expression data

Parameters

- **data_path** (*str*) –
- **gene_symbol_column** (*str*) –
- **logfc_colun** (*str*) –

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

Return type dict

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.**get_version**()
Gets the current PyBEL Tools version

Returns The current PyBEL Tools version

Return type str

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.**grouper**(*n*, *iterable*, *fillvalue=None*)
grouper(3, 'ABCDEFG', 'x') –> ABC DEF Gxx

pybel_tools.utils.**get_iso_8601_date**()
Gets the current ISO 8601 date as a string

Return type str

pybel_tools.utils.**hash_str_to_int**(*hash_str*, *length=16*)
Hashes a tuple to the given number of digits

Parameters

- **hash_str** (*str*) – Basically anything that can be pickled deterministically
- **length** (*int*) – The length of the hash to keep

Return type int

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

Lexer

Get the Code

Get the pygment source code from bitbucket repo with `hg clone http://bitbucket.org/birkenfeld/pygments-main pygments`

Register Your Lexer

To make Pygments aware of your new lexer, you have to perform the following steps:

1. First, change to the current directory containing the Pygments source code:

```
$ cd .../pygments
```

   Select a matching module under pygments/lexers, or create a new module for your lexer class. Copy the BELLexer.py in that module

2. The lexer can be made publicly known by rebuilding the lexer mapping:

```
$ make mapfiles
```

Test

To test the new lexer, store an example file with the proper extension in tests/examplefiles. For example, to test the BELLexer, add a tests/examplefiles/Example.bel containing a sample bel code.

Now you can use pygmentize to render your example to HTML:

```
$ ./pygmentize -O full -f html -o /tmp/example.html tests/examplefiles/Example.bel
```

To view the result, open ./example.html in your browser.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W