pypath Documentation

Release 0.7.98

Dénes Türei

Nov 15, 2018

CONTENTS:

pypath is a Python package built around igraph to work with molecular network representations e.g. protein, miRNA and drug compound interaction networks.

note `pypath` supports both Python 2.7 and Python 3.6+. In the beginning, pypath has been developed only for Python 2.7. Then the code have been adjusted to Py3 however we can not guarantee no incompatibilities remained. If you find any method does not work please submit an issue on github. For few years I develop and test `pypath` in Python 3. Therefore this is the better supported Python variant.

contributions turei.denes@gmail.com

documentation http://pypath.omnipathdb.org/

issues https://github.com/saezlab/pypath/issues

CHAPTER

ONE

INSTALLATION

1.1 Linux

In almost any up-to-date Linux distribution the dependencies of **pypath** are built-in, or provided by the distributors. You only need to install a couple of things in your package manager (cairo, py(2)cairo, igraph, python(2)-igraph, graphviz, pygraphviz), and after install **pypath** by *pip* (see below). If any module still missing, you can install them the usual way by *pip* or your package manager.

1.1.1 igraph C library, cairo and pycairo

python(2)-igraph is a Python interface to use the igraph C library. The C library must be installed. The same goes for *cairo*, *py(2)cairo* and *graphviz*.

1.1.2 Directly from git

```
pip install git+https://github.com/saezlab/pypath.git
```

1.1.3 With pip

Download the package from /dist, and install with pip:

```
pip install pypath-x.y.z.tar.gz
```

1.1.4 Build source distribution

Clone the git repo, and run setup.py:

```
python setup.py sdist
```

1.2 Mac OS X

On OS X installation is not straightforward primarily because cairo needs to be compiled from source. We provide 2 scripts here: the **mac-install-brew.sh** installs everything with HomeBrew, and **mac-install-conda.sh** installs from Anaconda distribution. With these scripts installation of igraph, cairo and graphviz goes smoothly most of the time, and options are available for omitting the 2 latter. To know more see the description in the script header. There is a

third script **mac-install-source.sh** which compiles everything from source and presumes only Python 2.7 and Xcode installed. We do not recommend this as it is time consuming and troubleshooting requires expertise.

1.2.1 Troubleshooting

- `no module named ...` when you try to load a module in Python. Did

theinstallation of the module run without error? Try to run again the specific part from the mac install shell script to see if any error comes up. Is the path where the module has been installed in your `$PYTHONPATH`? Try `echo $PYTHONPATH` to see the current paths. Add your local install directories if those are not there, e.g. `export PYTHONPATH="/Users/me/local/python2.7/site-packages:$PYTHONPATH"`. If it works afterwards, don't forget to append these export path statements to your `~/.bash_profile`, so these will be set every time you launch a new shell.

- `pkgconfig` not found. Check if the `$PKG_CONFIG_PATH` variable is

set correctly, and pointing on a directory where pkgconfig really can be found.

- Error while trying to install py(2)cairo by pip. py(2)cairo could not be

installed by pip, but only by waf. Please set the `$PKG_CONFIG_PATH` before. See **mac-install-source.sh** on how to install with waf.

- Error at pygraphviz build: `graphviz/cgraph.h file not found`. This is

because the directory of graphviz detected wrong by pkgconfig. See **mac-install-source.sh** how to set include dirs and library dirs by `--global-option` parameters.

- Can not install bioservices, because installation of jurko-suds fails. Ok,

this fails because pip is not able to install the recent version of setuptools, because a very old version present in the system path. The development version of jurko-suds does not require setuptools, so you can install it directly from git as it is done in **mac-install-source.sh**.

- In **Anaconda**, *pypath* can be imported, but the modules and classes are

missing. Apparently Anaconda has some built-in stuff called *pypath*. This has nothing to do with this module. Please be aware that Anaconda installs a completely separated Python distribution, and does not detect modules in the main Python installation. You need to install all modules within Anaconda's directory. **mac-install-conda.sh** does exactly this. If you still experience issues, please contact us.

1.3 Microsoft Windows

Not many people have used *pypath* on Microsoft computers so far. Please share your experiences and contact us if you encounter any issue. We appreciate your feedback, and it would be nice to have better support for other computer systems.

1.3.1 With Anaconda

The same workflow like you see in `mac-install-conda.sh` should work for Anaconda on Windows. The only problem you certainly will encounter is that not all the channels have packages for all platforms. If certain channel provides no package for Windows, or for your Python version, you just need to find an other one. For this, do a search:

```
anaconda search -t conda <package name>
```

For example, if you search for *pycairo*, you will find out that *vgauther* provides it for osx-64, but only for Python 3.4, while *richlewis* provides also for Python 3.5. And for win-64 platform, there is the channel of *KristanAmstrong*. Go along all the commands in `mac-install-conda.sh`, and modify the channel if necessary, until all packages install successfully.

1.3.2 With other Python distributions

Here the basic principles are the same as everywhere: first try to install all external dependencies, after *pip* install should work. On Windows certain packages can not be installed by compiled from source by *pip*, instead the easiest to install them precompiled. These are in our case *fisher, lxml, numpy (mkl version), pycairo, igraph, pygraphviz, scipy and statsmodels*. The precompiled packages are available here: http://www.lfd.uci.edu/~gohlke/pythonlibs/. We tested the setup with Python 3.4.3 and Python 2.7.11. The former should just work fine, while with the latter we have issues to be resolved.

1.3.3 Known issues

- *"No module fabric available."* – or *pysftp* missing: this is not

important, only certain data download methods rely on these modules, but likely you won't call those at all. * Progress indicator floods terminal: sorry about that, will be fixed soon. * Encoding related exceptions in Python2: these might occur at some points in the module, please send the traceback if you encounter one, and we will fix as soon as possible.

Special thanks to Jorge Ferreira for testing pypath on Windows!

CHAPTER

TWO

REFERENCE

class pypath.main.**PyPath**(*ncbi_tax_id=9606*, *default_name_type={'drug': 'chembl', 'lncrna': 'lncrna-genesymbol', 'mirna': 'mirbase', 'protein': 'uniprot'}*, *copy=None*, *mysql=(None, 'mapping')*, *chembl_mysql=(None, 'chembl')*, *name='unnamed'*, *outdir='results'*, *loglevel='INFO'*, *loops=False*)

Main network object.

Parameters

- **ncbi_tax_id** (*int*) – Optional, `9606` (Homo sapiens) by default. NCBI Taxonomic identifier of the organism from which the data will be downloaded.
- **default_name_type** (*dict*) – Optional, `{'protein': 'uniprot', 'mirna': 'mirbase', 'drug': 'chembl', 'lncrna': 'lncrna-genesymbol'}` by default. Contains the default identifier types to which the downloaded data will be converted. If others are used, user may need to provide the format definitions for the conversion tables.
- **copy** (`pypath.main.PyPath`) – Optional, `None` by default. Other *pypath.main.PyPath* instance from which the data will be copied.
- **mysql** (*tuple*) – Optional, `(None, 'mapping')` by default. Contains the MySQL parameters used by the `pypath.mapping` module to load the ID conversion tables.
- **chembl_mysql** (*tuple*) – Optional, `(None, 'chembl')` by default. Contains the MySQL parameters used by the `pypath.mapping` module to load the ChEMBL ID conversion tables.
- **name** (*str*) – Optional, `'unnamed'` by default. Session or project name (custom).
- **outdir** (*str*) – Optional, `'results'` by default. Output directory where to store all output files.
- **loglevel** (*str*) – Optional, `'INFO'` by default. Sets the level of the logger. Possible levels are: `'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'` or `'CRITICAL'`.
- **loops** (*bool*) – Optional, `False` by default. Determines if self-loop edges are allowed in the graph.

Variables

- **adjlist** (*list*) – List of [set] containing the adjacency of each node. See *PyPath.update_adjlist()* method for more information.
- **chembl** (*pypath.chembl.Chembl*) – Contains the ChEMBL data. See `pypath.chembl` module documentation for more information.
- **chembl_mysql** (*tuple*) – Contains the MySQL parameters used by the `pypath.mapping` module to load the ChEMBL ID conversion tables.

- **data** (*dict*) – Stores the loaded interaction and attribute table. See *PyPath.read_data_file()* method for more information.
- **db_dict** (*dict*) – Dictionary of dictionaries. Outer-level keys are `'nodes'` and `'edges'`, corresponding values are [dict] whose keys are the database sources with values of type [set] containing the edge/node indexes for which that database provided some information.
- **dgraph** (*igraph.Graph*) – Directed network graph object.
- **disclaimer** (*str*) – Disclaimer text.
- **dlabDct** (*dict*) – Maps the directed graph node labels [str] (keys) to their indices [int] (values).
- **dnodDct** (*dict*) – Maps the directed graph node names [str] (keys) to their indices [int] (values).
- **dnodInd** (*set*) – Stores the directed graph node names [str].
- **dnodLab** (*dict*) – Maps the directed graph node indices [int] (keys) to their labels [str] (values).
- **dnodNam** (*dict*) – Maps the directed graph node indices [int] (keys) to their names [str] (values).
- **edgeAttrs** (*dict*) – Stores the edge attribute names [str] as keys and their corresponding types (e.g.: `set`, `list`, `str`, ...) as values.
- **exp** (*pandas.DataFrame*) – Stores the expression data for the nodes (if loaded).
- **exp_prod** (*pandas.DataFrame*) – Stores the edge expression data (as the product of the normalized expression between the pair of nodes by default). For more details see *PyPath.edges_expression()*.
- **exp_samples** (*set*) – Contains a list of tissues as downloaded by ProteomicsDB. See *PyPath.get_proteomicsdb()* for more information.
- **failed_edges** (*list*) – List of lists containing information about the failed edges. Each failed edge sublist contains (in this order): [tuple] with the node IDs, [str] names of nodes A and B, [int] IDs of nodes A and B and [int] IDs of the edges in both directions.
- **go** (*dict*) – Contains the organism(s) NCBI taxonomy ID as key [int] and `pypath.go.GOAnnotation` object as value, which contains the GO annotations for the nodes in the graph. See `pypath.go.GOAnnotation` for more information.
- **graph** (*igraph.Graph*) – Undirected network graph object.
- **gsea** (*pypath.gsea.GSEA*) – Contains the loaded gene-sets from MSigDB. See `pypath.gsea.GSEA` for more information.
- **has_cats** (*set*) – Contains the categories (e.g.: resources) [str] loaded in the current network.
- **htp** (*dict*) – Contains information about high-throughput data of the network for different thresholds [int] (keys). Values are [dict] containing the number of references (`'rnum'`) [int], number of edges (`'enum'`) [int], number of sources (`'snum'`) [int] and list of PMIDs of the most common references above the given threshold (`'htrefs'`) [set].
- **labDct** (*dict*) – Maps the undirected graph node labels [str] (keys) to their indices [int] (values).

- **lists** (*dict*) – Contains specific lists of nodes (values) for different categories [str] (keys). These can to be loaded from a file or a resource. Some methods include `PyPath.receptor_list()` (`'rec'`), *PyPath.druggability_list()* (`'dgb'`), *PyPath.kinases_list()* (`'kin'`), *PyPath.tfs_list()* (`'tf'`), *PyPath.disease_genes_list()* (`'dis'`), *PyPath.signaling_proteins_list()* (`'sig'`), *PyPath.proteome_list()* (`'proteome'`) and *PyPath.cancer_drivers_list()* (`'cdv'`).
- **loglevel** (*str*) – The level of the logger.
- **loops** (*bool*) – Whether if self-loop edges are allowed in the graph.
- **mapper** (*pypath.mapping.Mapper*) – `pypath.mapper.Mapper` object for ID conversion and other ID-related operations across resources.
- **mutation_samples** (*list*) – DEPRECATED
- **mysql_conf** (*tuple*) – Contains the MySQL parameters used by the `pypath.mapping` module to load the ID conversion tables.
- **name** (*str*) – Session or project name (custom).
- **ncbi_tax_id** (*int*) – NCBI Taxonomic identifier of the organism from which the data will be downloaded.
- **negatives** (*dict*) – Contains a list of negative interactions according to a given source (e.g.: Negatome database). See *PyPath.apply_negative()* for more information.
- **nodDct** (*dict*) – Maps the undirected graph node names [str] (keys) to their indices [int] (values).
- **nodInd** (*set*) – Stores the undirected graph node names [str].
- **nodLab** (*dict*) – Maps the undirected graph node indices [int] (keys) to their labels [str] (values).
- **nodNam** (*dict*) – Maps the directed graph node indices [int] (keys) to their names [str] (values).
- **outdir** (*str*) – Output directory where to store all output files.
- **ownlog** (*pypath.logn.logw*) – Logger class instance, see `pypath.logn.logw` for more information.
- **palette** (*list*) – Contains a list of hexadecimal [str] of colors. Used for plotting purposes.
- **pathway_types** (*list*) – Contains the names of all the loaded pathway resources [str].
- **pathways** (*dict*) – Contains the list of pathways (values) for each resource (keys) loaded in the network.
- **plots** (*dict*) – DEPRECATED (?)
- **proteomicsdb** (*pypath.proteomicsdb.ProteomicsDB*) – Contains a `pypath.proteomicsdb.ProteomicsDB` instance, see the class documentation for more information.
- **raw_data** (*list*) – Contains a list of loaded edges [dict] from a data file. See *PyPath.read_data_file()* for more information.
- **reflists** (*dict*) – Contains the reference list(s) loaded. Keys are [tuple] containing the node name type [str] (e.g.: `'uniprot'`), type [str] (e.g.: `'protein'`) and tax-

onomic ID [int] (e.g.: `'9606'`). Values are the corresponding `pypath.reflists.ReferenceList` instance (see class documentation for more information).

- **seq** (*dict*) – (?)
- **session** (*str*) – Session ID, a five random alphanumeric characters.
- **session_name** (*str*) – Session name and ID (e.g. `'unnamed-abc12'`).
- **sourceNetEdges** (*igraph.Graph*) – (?)
- **sourceNetNodes** (*igraph.Graph*) – (?)
- **sources** (*list*) – List contianing the names of the loaded resources [str].
- **u_pfam** (*dict*) – Dictionary of dictionaries, contains the mapping of UniProt IDs to their respective protein families and other information.
- **uniprot_mapped** (*list*) – DEPRECATED (?)
- **unmapped** (*list*) – Contains the names of unmapped items [str]. See *PyPath.map_item()* for more information.
- **vertexAttrs** (*dict*) – Stores the node attribute names [str] as keys and their corresponding types (e.g.: `set`, `list`, `str`, ...) as values.

acsn_effects (*graph=None*)

add_genesets (*genesets*)

add_list_eattr (*edge*, *attr*, *value*)

add_update_edge (*nameA*, *nameB*, *source*, *isDir*, *refs*, *stim*, *inh*, *taxA*, *taxB*, *typ*, *extraAttrs={}*, *add=False*)

add_update_vertex (*defAttrs*, *originalName*, *originalNameType*, *extraAttrs={}*, *add=False*)

Updates the attributes of one node in the network. Optionally it creates a new node and sets the attributes, but it is not efficient as igraph needs to reindex vertices after this operation, so better to create new nodes and edges in batch.

affects (*identifier*)

all_between (*nameA*, *nameB*)

Returns all edges between two given vertex names. Similar to straight_between(), but checks both directions, and returns list of edge ids in [undirected, straight, reversed] format, for both nameA -> nameB and nameB -> nameA edges.

all_neighbours (*indices=False*)

apply_list (*name*, *node_or_edge='node'*)

Creates vertex or edge attribute based on a list.

apply_negative (*settings*)

attach_network (*edgeList=False*, *regulator=False*)

Adds edges to the network from edgeList obtained from file or other input method.

basic_stats (*latex=False, caption='', latex_hdr=True, fontsize=8, font='HelveticaNeueLTStd-LtCn', fname=None, header_format='%s', row_order=None, by_category=True, use_cats=['p', 'm', 'i', 'r'], urls=True, annots=False*)

Returns basic numbers about the network resources, e.g. edge and node counts.

latex Return table in a LaTeX document. This can be compiled by PDFLaTeX: latex stats.tex

basic_stats_intergroup (*groupA*, *groupB*, *header=None*)

cancer_drivers_list(*intogen_file=None*)
Loads the list of cancer drivers. Contains information from COSMIC (needs user log in credentials) and IntOGen (if provided) and adds the attribute to the undirected network nodes.

Parameters **intogen_file** (*str*) – Optional, `None` by default. Path to the data file. Can also be [function] that provides the data. In general, anything accepted by `pypath.input_formats.ReadSettings.inFile`.

cancer_gene_census_list()
Loads the list of cancer driver proteins from the COSMIC Cancer Gene Census.

clean_graph()
Removes multiple edges, unknown molecules and those from wrong taxon. Multiple edges will be combined by *combine_attr()* method. Loops will be deleted unless the *loops* attribute set to *True*.

collapse_by_name(*graph=None*)
Collapses nodes with the same name by copying and merging all edges and attributes. Operates directly on the provided network object.

Parameters **graph** (*igraph.Graph*) – Optional, `None` by default. The network for which the nodes are to be collapsed. If none is provided, takes `pypath.main.PyPath.graph` (undirected network) by default.

combine_attr(*lst*, *num_method=<built-in function max>*)
Combines multiple attributes into one. This method attempts to find out which is the best way to combine attributes.

- If there is only one value or one of them is None, then returns the one available.
- Lists: concatenates unique values of lists.
- Numbers: returns the greater by default or calls *num_method* if given.
- Sets: returns the union.
- Dictionaries: calls `pypath.common.merge_dicts()`.
- Direction: calls their special *pypath.main.Direction.merge()* method.

Works on more than 2 attributes recursively.

Parameters

- **lst** (*list*) – List of one or two attribute values.
- **num_method** (*function*) – Optional, `max` by default. Method to merge numeric attributes.

communities(*method*, ***kwargs*)

complex_comembership_network(*graph=None*, *resources=None*)

complexes(*methods=['3dcomplexes', 'havugimana', 'corum', 'complexportal', 'compleat']*)

complexes_in_network(*csource='corum'*, *graph=None*)

compounds_from_chembl(*chembl_mysql=None, nodes=None, crit=None, andor='or', assay_types=['B', 'F'], relationship_types=['D', 'H'], multi_query=False, **kwargs*)

consistency()

copy(*other*)
Copies another *pypath.main.PyPath* instance into the current one.

Parameters **other** (`pypath.main.PyPath`) – The instance to be copied from.

copy_edges(*sources*, *target*, *move=False*, *graph=None*)
Copies edges of one node to another, keeping attributes and directions.

Parameters

- **sources** (*list*) – Vertex IDs to copy from.
- **target** (*int*) – Vertex ID to copy for.
- **move** (*bool*) – Whether perform copy or move, i.e. remove or keep the source edges.

count_sol()
Counts nodes with zero degree.

coverage(*lst*)
Computes the coverage (range [0, 1]) of a list of nodes against the current (undirected) network.

Parameters **lst** (*set*) – Can also be [list] (will be converted to [set]) or [str]. In the latter case it will retrieve the list with that name (if such list exists in `pypath.main.PyPath.lists`).

curation_effort(*sum_by_source=False*)
Returns the total number of reference-interactions pairs.

@**sum_by_source** [bool] If True, counts the refrence-interaction pairs by sources, and returns the sum of these values.

curation_stats(*by_category=True*)

curation_tab(*fname='curation_stats.tex', by_category=True, use_cats=['p', 'm', 'i', 'r'], header_size='normalsize', **kwargs*)

curators_work()
Computes and prints an estimation of how many years of curation took to achieve the amount of information on the network.

databases_similarity(*index='simpson'*)

degree_dist(*prefix*, *g*, *group=None*)

degree_dists()

delete_by_source(*source*, *vertexAttrsToDel=None*, *edgeAttrsToDel=None*)

delete_by_taxon(*tax*)
Removes the proteins of all organisms which are not listed.

Parameters **tax** (*list*) – List of NCBI Taxonomy IDs of the organisms. E.g. [7227, 9606]

delete_unknown(*tax*, *typ='protein'*, *defaultNameType=None*)
Removes those proteins which are not in the list of all default IDs of the organisms. By default, it means to remove all protein nodes not having a human SwissProt ID.

@**tax** [list] List of NCBI Taxonomy IDs of the organisms of interest. E.g. [7227, 9606]

@**typ** [str] Molecule type. E.g. 'protein' or 'mirna'

@**defaultNameType** [str] The default name type of the given molecular species. For proteins it's 'uniprot' by default.

delete_unmapped()

dgenesymbol(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default directed network, otherwise `None`.

@**genesymbol** [str] GeneSymbol.

dgenesymbols(*genesymbols*)

dgs(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default directed network, otherwise `None`.

@genesymbol [str] GeneSymbol.

dgss(*genesymbols*)

disease_genes_list(*dataset='curated'*)
Loads the list of all disease related genes from DisGeNet. This resource is human only.

dneighbors(*identifier*, *mode='ALL'*)

dp(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

dproteins(*identifiers*)

dps(*identifiers*)

druggability_list()
Loads the list of druggable proteins from DgiDB. This resource is human only.

duniprot(*uniprot*)
Same as `PyPath.uniprot(), just for directed graph. Returns ``igraph.Vertex()` object if the UniProt can be found in the default directed network, otherwise `None`.

@uniprot [str] UniProt ID.

duniprots(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default directed graph.

dup(*uniprot*)
Same as `PyPath.uniprot(), just for directed graph. Returns ``igraph.Vertex()` object if the UniProt can be found in the default directed network, otherwise `None`.

@uniprot [str] UniProt ID.

dups(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default directed graph.

dv(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

dvs(*identifiers*)

edge_exists(*nameA*, *nameB*)
Returns a tuple of vertice indices if edge doesn't exists, otherwise edge id. Not sensitive to direction.

edge_loc(*graph=None*, *topn=2*)

edge_names(*e*)

edges_3d(*methods=['dataio.get_instruct', 'dataio.get_i3d']*)

edges_expression(*func=<function <lambda>>*)

Executes function *func* for each pairs of connected proteins in the network, for every expression dataset. By default, *func* simply gives the product the (normalized) expression values.

func [callable] Function to handle 2 vectors (pandas.Series() objects), should return one vector of the same length.

edges_in_comlexes(*csources=['corum'], graph=None*)

Creates edge attributes `complexes` and `in_complex`. These are both dicts where the keys are complex resources. The values in `complexes` are the list of complex names both the source and the target vertices belong to. The values `in_complex` are boolean values whether there is at least one complex in the given resources both the source and the target vertex of the edge belong to.

@csources [list] List of complex resources. Should be already loaded.

@graph [igraph.Graph()] The graph object to do the calculations on.

edges_ptms()

edgeseq_inverse(*edges*)

export_dot(*nodes=None*, *edges=None*, *directed=True*, *labels='genesymbol'*, *edges_filter=<function <lambda>>*, *nodes_filter=<function <lambda>>*, *edge_sources=None*, *dir_sources=None*, *graph=None*, *return_object=False*, *save_dot=None*, *save_graphics=None*, *prog='neato'*, *format=None*, *hide=False*, *font=None*, *auto_edges=False*, *hide_nodes=[]*, *defaults={}*, ***kwargs*)

Builds a pygraphviz.AGraph() object with filtering the edges and vertices along arbitrary criteria. Returns the Agraph object if requesred, or exports the dot file, or saves the graphics.

@nodes : list List of vertex ids to be included. @edges : list List of edge ids to be included. @directed : bool Create a directed or undirected graph. @labels : str Name type to be used as id/label in the dot format. @edges_filter : function Function to filter edges, accepting igraph.Edge as argument. @nodes_filter : function Function to filter vertices, accepting igraph.Vertex as argument. @edge_sources : list Sources to be included. @dir_sources : list Direction and effect sources to be included. @graph : igraph.Graph The graph object to export. @return_object : bool Whether to return the pygraphviz.AGraph object. @save_dot : str Filename to export the dot file to. @save_graphics : str Filename to export the graphics, the extension defines the format. @prog : str The graphviz layout algorithm to use. @format : str The graphics format passed to pygraphviz.AGrapg().draw(). @hide : bool Hide filtered edges instead of omit them. @hide nodes : list Nodes to hide. List of vertex ids. @auto_edges : str Automatic, built-in style for edges. 'DIRECTIONS' or 'RESOURCE_CATEGORIES' are supported. @font : str Font to use for labels. For using more than one fonts refer to graphviz attributes with constant values or define callbacks or mapping dictionaries. @defaults : dict Default values for graphviz attributes, labeled with the entity, e.g. *{'edge_penwidth': 0.2}*. @**kwargs : constant, callable or dict Graphviz attributes, labeled by the target entity. E.g. *edge_penwidth*, 'vertex_shape' or *graph_label*. If the value is constant, this value will be used. If the value is dict, and has *_name* as key, for every instance of the given entity, the value of the attribute defined by *_name* will be looked up in the dict, and the corresponding value will be given to the graphviz attribute. If the key *_name* is missing from the dict, igraph vertex and edge indices will be looked up among the keys. If the value is callable, it will be called with the current instance of the entity and the returned value will be used for the graphviz attribute. E.g. *edge_arrowhead(edge)* or *vertex_fillcolor(vertex)* Example:

import pypath from pypath import data_formats net = pypath.PyPath() net.init_network(pfile = 'cache/default.pickle') #net.init_network({'arn': data_formats.omnipath['arn']}) tgf = [v.index for v in net.graph.vs if 'TGF' in v['slk_pathways']] dot = net.export_dot(nodes = tgf, save_graphics = 'tgf_slk.pdf', prog = 'dot',

main_title = 'TGF-beta pathway', return_object = True, label_font = 'HelveticaNeueLT-Std Med Cn', edge_sources = ['SignaLink3'], dir_sources = ['SignaLink3'], hide = True)

export_edgelist(*fname, graph=None, names=['name'], edge_attributes=[], sep='\t'*)
Write edge list to text file with attributes

@param fname: the name of the file or a stream to read from. @param graph: the igraph object containing the network @param names: list with the vertex attribute names to be printed

for source and target vertices

@param edge_attributes: list with the edge attribute names to be printed

@param sep: string used to separate columns

export_graphml(*outfile=None*, *graph=None*, *name='main'*)

export_ptms_tab(*outfile=None*)

export_sif(*outfile=None*)

export_struct_tab(*outfile=None*)

export_tab(*outfile=None*, *extra_node_attrs={}*, *extra_edge_attrs={}*, *unique_pairs=True*, ***kwargs*)
Exports the network in a tabular format.

By default UniProt IDs, Gene Symbols, source databases, literature references, directionality and sign information and interaction type are included.

Parameters

- **outfile** (*str*) – Name of the output file. If *None* a file name "netrowk-<session id>.tab" is used.
- **extra_node_attrs** (*dict*) – Additional node attributes to be included in the exported table. Keys are column ames used in the header while values are names of vertex attributes. In the header *_A* and *_B* suffixes will be appended to the column names so the values can be assigned to A and B side interaction partners.
- **extra_edge_attrs** (*dict*) – Additional edge attributes to be included in the exported table. Keys are column ames used in the header while values are names of edge attributes.

filters(*line*, *positiveFilters=[]*, *negativeFilters=[]*)

find_all_paths(*start*, *end*, *mode='OUT'*, *maxlen=2*, *graph=None*, *silent=False*)
Finds all paths up to length *maxlen* between groups of vertices. This function is needed only becaues igraph's get_all_shortest_paths() finds only the shortest, not any path up to a defined length.

@start [int or list] Indices of the starting node(s) of the paths.

@end [int or list] Indices of the target node(s) of the paths.

@mode ['IN', 'OUT', 'ALL'] Passed to igraph.Graph.neighbors()

@maxlen [int] Maximum length of paths in steps, i.e. if maxlen = 3, then the longest path may consist of 3 edges and 4 nodes.

@graph [igraph.Graph object] The graph you want to find paths in. self.graph by default.

find_all_paths2(*graph*, *start*, *end*, *mode='OUT'*, *maxlen=2*, *psize=100*)

find_complex(*search*)
Finds complexes by their non standard names. E.g. to find DNA polymerases you can use the search term *DNA pol* which will be tested against complex names in CORUM.

first_neighbours(*node*, *indices=False*)

fisher_enrichment(*lst*, *attr*, *ref='proteome'*)
Computes an enrichment analysis using Fisher's exact test. The contingency table is built as follows: First row contains the number of nodes in the *ref* list (such list is considered to be loaded in `pypath.main.PyPath.lists`) and the number of nodes in the current (undirected) network. Second row contains the number of nodes in *lst* list (also considered to be already loaded) and the number of nodes in the network with a non-empty attribute *attr*. Uses **:py:fun:'scipy.stats.fisher_exact'**, see the documentation of the corresponding package for more information.

Parameters

- **lst** (*str*) – Name of the list in `pypath.main.PyPath.lists` whose number of elements will be the first element in the second row of the contingency table.
- **attr** (*str*) – The node attribute name for which the number of nodes in the network with such attribute will be the second element of the second row of the contingency table.
- **ref** (*str*) – Optional, `'proteome'` by default. The name of the list in `pypath.main.PyPath.lists` whose number of elements will be the first element of the first row of the contingency table.

Returns

- (*float*) – Prior odds ratio.
- (*float*) – P-value or probability of obtaining a distribution as extreme as the observed, assuming that the null hypothesis is true.

geneset_enrichment(*proteins*, *all_proteins=None*, *geneset_ids=None*, *alpha=0.05*, *correction_method='hommel'*)

genesymbol(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default undirected network, otherwise `None`.

@genesymbol [str] GeneSymbol.

genesymbol_labels(*graph=None*, *remap_all=False*)
Creats vertex attribute `label` and fills up with Gene Symbols of all proteins where the Gene Symbol can be looked up based on the default name of the protein vertex. If the attribute `label` had been already initialized, updates this attribute or recreates if `remap_all` is `True`.

genesymbols(*genesymbols*)

get_attrs(*line*, *spec*, *lnum*)

get_directed(*graph=False*, *conv_edges=False*, *mutual=False*, *ret=False*)
Converts `graph` undirected `igraph.Graph` object to a directed one. By default it converts the graph in `PyPath.graph` and places the directed instance in `PyPath.dgraph`.

@graph [igraph.Graph] Undirected graph object.

@conv_edges [bool] Whether to convert undirected edges (those without explicit direction information) to an arbitrary direction edge or a pair of opposite edges. Otherwise those will be deleted. Default is `False`.

@mutual [bool] If `conv_edges` is `True`, whether to convert the undirected edges to a single, arbitrary directed edge, or a pair of opposite directed edges. Default is `False`.

@ret [bool] Return the directed graph instance, or return `None`. Default is `False` (returns `None`).

get_dirs_signs()

get_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

Parameters

- **source** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **target** (*int, str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **directed** (*bool*) – To be passed to igraph.Graph.get_eid()

get_edges(*sources*, *targets*, *directed=True*)
Returns a generator with all edges between source and target vertices.

Parameters

- **sources** (*iterable*) – Source vertex IDs, names, labels, or any iterable yielding `igraph.Vertex` objects.
- **targets** (*iterable*) – Target vertec IDs, names, labels, or any iterable yielding `igraph.Vertex` objects.
- **directed** (*bool*) – Passed to `igraph.get_eid()`.

get_function(*fun*)

get_giant(*replace=False*, *graph=None*)
Returns the giant component of the *graph*, or replaces the `igraph.Graph` instance with only the giant component if specified.

Parameters

- **replace** (*bool*) – Optional, `False` by default. Specifies whether to replace the `igraph.Graph` instance. This can be either the undirected network of the current *pypath.main.PyPath* instance (default) or the one passed under the keyword argument *graph*.
- **graph** (*igraph.Graph*) – Optional, `None` by default. The graph object from which the giant component is to be computed. If none is specified, takes the undirected network of the current *pypath.main.PyPath* instance.

Returns (*igraph.Graph*) – If `replace=False`, returns a copy of the giant component graph.

get_max(*attrList*)

get_network(*crit*, *andor='or'*, *graph=None*)

get_node(*identifier*)
Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

get_node_d(*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

get_node_pair(*nameA*, *nameB*, *directed=False*)

get_nodes(*identifiers*)

get_nodes_d(*identifiers*)

get_pathways(*source*)

get_proteomicsdb(*user*, *passwd*, *tissues=None*, *pickle=None*)

get_sub(*crit*, *andor='or'*, *graph=None*)

get_taxon(*tax_dict*, *fields*)

go_annotate(*aspects=('C', 'F', 'P')*)
Annotates protein nodes with GO terms. In the `go` vertex attribute each node is annotated by a dict of sets where keys are one letter codes of GO aspects and values are sets of GO accessions.

go_dict(*organism=9606*)
Creates a `pypath.go.GOAnnotation` object for one organism in the dict under `go` attribute.

Parameters **organism** (*int*) – NCBI Taxonomy ID of the organism.

go_enrichment(*proteins=None*, *aspect='P'*, *alpha=0.05*, *correction_method='hommel'*, *all_proteins=None*)

gs(*genesymbol*)
Returns `igraph.Vertex()` object if the GeneSymbol can be found in the default undirected network, otherwise `None`.

@genesymbol [str] GeneSymbol.

gs_affected_by(*genesymbol*)

gs_affects(*genesymbol*)

gs_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

@source [str] GeneSymbol

@target [str] GeneSymbol

@directed [bool] To be passed to igraph.Graph.get_eid()

gs_in_directed(*genesymbol*)

gs_in_undirected(*genesymbol*)

gs_inhibited_by(*genesymbol*)

gs_inhibits(*genesymbol*)

gs_neighborhood(*genesymbols*, *order=1*, *mode='ALL'*)

gs_neighbors(*genesymbol*, *mode='ALL'*)

gs_stimulated_by(*genesymbol*)

gs_stimulates(*genesymbol*)

gss(*genesymbols*)

guide2pharma()

having_attr(*attr*, *graph=None*, *index=True*, *edges=True*)

having_eattr(*attr*, *graph=None*, *index=True*)

having_ptm(*index=True*, *graph=None*)

having_vattr(*attr*, *graph=None*, *index=True*)

homology_translation(*target*, *source=None*, *only_swissprot=True*, *graph=None*)
Translates the current object to another organism by orthology. Proteins without known ortholog will be deleted.

Parameters **target** (*int*) – NCBI Taxonomy ID of the target organism. E.g. 10090 for mouse.

htp_stats()

in_complex(*csources=['corum']*)

in_directed(*vertex*)

in_undirected(*vertex*)

info(*name*)

init_complex_attr(*graph*, *name*)

init_edge_attr(*attr*)
Fills edge attribute with its default values, creates lists if in *edgeAttrs* the attribute is registered as list.

init_gsea(*user*)

init_network(*lst=None*, *exclude=[]*, *cache_files={}*, *pfile=False*, *save=False*, *reread=False*, *redownload=False*, ***kwargs*)
Loads the network data.

This is a lazy way to start the module, load data and build the high confidence, literature curated part of the signaling network.

Parameters

- **lst** (*dict*) – Optional, `None` by default. Specifies the data input formats for the different resources (keys) [str]. Values are `pypath.input_formats.ReadSettings` instances containing the information. By default uses the set of resources of OmniPath.
- **exclude** (*list*) – Optional, `[]` by default. List of resources [str] to exclude from the network.
- **cache_files** (*dict*) – Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **pfile** (*str*) – Optional, `False` by default. If any, provides the file name or path to a previously saved network pickle file. If `True` is passed, takes the default path from *PyPath.save_network()* (`'cache/default_network.pickle'`).
- **save** (*bool*) – Optional, `False` by default. If set to `True`, saves the loaded network to its default location (`'cache/default_network.pickle'`).
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).
- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.
- ****kwargs** – Not used.

init_vertex_attr(*attr*)
Fills vertex attribute with its default values, creates lists if in *vertexAttrs* the attribute is registered as list.

intergroup_shortest_paths(*groupA*, *groupB*, *random=False*)

intogen_cancer_drivers_list(*intogen_file*)
Loads the list of cancer driver proteins from IntOGen data.

Parameters **intogen_file** (*str*) – Path to the data file. Can also be [function] that provides the data. In general, anything accepted by `pypath.input_formats.ReadSettings.inFile`.

jaccard_edges()

jaccard_meta(*jedges*, *critical*)

kegg_directions(*graph=None*)

kegg_pathways(*graph=None*)

kinase_stats()

kinases_list()
Loads the list of all known kinases in the proteome from kinase.com. This resource is human only.

label_by_go(*label*, *go_terms*, ***kwargs*)
Assigns a boolean vertex attribute to nodes which tells whether the node is annotated by all or any (see `method` parameter of `select_by_go`) the GO terms.

laudanna_directions(*graph=None*)

laudanna_effects(*graph=None*)

licence()

list_resources()

load_3dcomplexes(*graph=None*)

load_3did_ddi()

load_3did_ddi2(*ddi=True*, *interfaces=False*)

load_3did_dmi()

load_3did_interfaces()

load_all_pathways(*graph=None*)

load_compleat(*graph=None*)
Loads complexes from Compleat. Loads data into vertex attribute *graph.vs['complexes']['compleat']*. This resource is human only.

load_complexportal(*graph=None*)
Loads complexes from ComplexPortal. Loads data into vertex attribute *graph.vs['complexes']['complexportal']*. This resource is human only.

load_comppi(*graph=None*)

load_corum(*graph=None*)
Loads complexes from CORUM database. Loads data into vertex attribute *graph.vs['complexes']['corum']*. This resource is human only.

load_dbptm(*non_matching=False*, *trace=False*, ***kwargs*)

load_ddi(*ddi*)
ddi is either a list of intera.DomainDomain objects, or a function resulting this list

load_ddis(*methods=['dataio.get_3dc_ddi', 'dataio.get_domino_ddi', 'self.load_3did_ddi2']*)

load_depod_dmi()

load_disgenet(*dataset='curated'*, *score=0.0*, *umls=False*, *full_data=False*)
Assigns DisGeNet disease-gene associations to the proteins in the network. Disease annotations will be added to the *dis* vertex attribute.

Parameters

- **score** (*float*) – Confidence score from DisGeNet. Only associations above the score provided will be considered.
- **ulms** (*bool*) – By default we assign a list of disease names to each protein. To use Unified Medical Language System IDs instead set this to *True*.
- **full_data** (*bool*) – By default we load only disease names. Set this to *True* if you wish to load additional annotations like number of PubMed IDs, number of SNPs and original sources.

load_dmi(*dmi*)
dmi is either a list of intera.DomainMotif objects, or a function resulting this list

load_dmis(*methods=['self.pfam_regions', 'self.load_depod_dmi', 'self.load_dbptm', 'self.load_mimp_dmi', 'self.load_pnetworks_dmi', 'self.load_domino_dmi', 'self.load_pepcyber', 'self.load_psite_reg', 'self.load_psite_phos', 'self.load_ielm', 'self.load_phosphoelm', 'self.load_elm', 'self.load_3did_dmi']*)

load_domino_dmi(*organism=None*)

load_elm()

load_expression(*array=False*)
Expression data can be loaded into vertex attributes, or into a pandas DataFrame – the latter offers faster ways to process and use these huge matrices.

load_go(*aspects=('C', 'F', 'P')*)
Annotates protein nodes with GO terms. In the `go` vertex attribute each node is annotated by a dict of sets where keys are one letter codes of GO aspects and values are sets of GO accessions.

load_havugimana(*graph=None*)
Loads complexes from Havugimana 2012. Loads data into vertex attribute *graph.vs['complexes']['havugimana']*. This resource is human only.

load_hpa(*normal=True*, *pathology=True*, *cancer=True*, *summarize_pathology=True*, *tissues=None*, *quality=set(['Supported', 'Approved'])*, *levels={'High': 3, 'Low': 1, 'Medium': 2, 'Not detected': 0}*, *graph=None*, *na_value=0*)
Loads Human Protein Atlas data into vertex attributes.

load_hprd_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_ielm()

load_interfaces()

load_li2012_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_ligand_receptor_network(*lig_rec_resources=True*, *inference_from_go=True*, *sources=None*, *keep_undirected=False*, *keep_rec_rec=False*, *keep_lig_lig=False*)
Initializes a ligand-receptor network.

load_list(*lst*, *name*)
Loads a custom list to the object's node data lists. See `pypath.main.PyPath.lists` attribute for more information.

Parameters

- **lst** (*list*) – The list containing the node names [str] from the given category (*name*).
- **name** (*str*) – The category or identifier for the list of nodes provided.

load_lmpid(*method*)

load_mappings()

load_mimp_dmi(*non_matching=False*, *trace=False*, ***kwargs*)

load_mutations(*attributes=None*, *gdsc_datadir=None*, *mutation_file=None*)
Mutations are listed in vertex attributes. Mutation() objects offers methods to identify residues and look up in Ptm(), Motif() and Domain() objects, to check if those residues are modified, or are in some short motif or domain.

load_negatives()

load_old_omnipath(*kinase_substrate_extra=False*, *remove_htp=False*, *htp_threshold=1*, *keep_directed=False*, *min_refs_undirected=2*)
Loads the OmniPath network as it was before August 2016. Furthermore it gives some more options.

load_omnipath(*kinase_substrate_extra=False*, *remove_htp=True*, *htp_threshold=1*, *keep_directed=True*, *min_refs_undirected=2*, *old_omnipath_resources=False*)
Loads the OmniPath network.

load_pathways(*source*, *graph=None*)
Generic method to load pathway annotations from a resource. We don't recommend calling this method but either specific methods for a single source e.g. *kegg_pathways()* or *sirnor_pathways()* or call *load_all_pathways()* to load all resources.

Parameters

- **source** (*str*) – Name of the source, this need to match a method in the dict in *get_pathways()* method and the edge and vertex attributes with pathway annotations will be called "<source>_pathways".
- **graph** (*igraph.Graph*) – A graph, by default the default the *graph* attribute of the current instance.

load_pdb(*graph=None*)

load_pepcyber()

load_pfam(*graph=None*)

load_pfam2()

load_pfam3()

load_phospho_dmi(*source*, *trace=False*, *return_raw=False*, ***kwargs*)

load_phosphoelm(*trace=False*, ***kwargs*)

load_pisa(*graph=None*)

load_pnetworks_dmi(*trace=False*, ***kwargs*)

load_psite_phos(*trace=False*, ***kwargs*)

load_psite_reg()

load_ptms()

load_ptms2(*input_methods=None, map_by_homology_from=[9606], homology_only_swissprot=True, ptm_homology_strict=False, nonhuman_direct_lookup=True, inputargs={}*)

This is a new method which will replace *load_ptms*. It uses *pypath.ptm.PtmAggregator*, a newly introduced module for combining enzyme-substrate data from multiple resources using homology translation on users demand.

Parameters

- **input_methods** (*list*) – Resources to collect enzyme-substrate interactions from. E.g. *['Signor', 'phosphoELM']*. By default it contains Signor, PhosphoSitePlus, HPRD, phosphoELM, dbPTM, PhosphoNetworks, Li2012 and MIMP.
- **map_by_homology_from** (*list*) – List of NCBI Taxonomy IDs of source taxons used for homology translation of enzyme-substrate interactions. If you have a human network and you add here *[10090, 10116]* then mouse and rat interactions from the source databases will be translated to human.
- **homology_only_swissprot** (*bool*) – *True* by default which means only SwissProt IDs are accepted at homology translateion, Trembl IDs will be dropped.
- **ptm_homology_strict** (*bool*) – For homology translation use PhosphoSite's PTM homology table. This guarantees that only truely homologous sites will be included. Otherwise we only check if at the same numeric offset in the homologous sequence the appropriate residue can be find.
- **nonhuman_direct_lookup** (*bool*) – Fetch also directly nonhuman data from the resources whereever it's available. PhosphoSite contains mouse enzyme-substrate interactions and it is possible to extract these directly beside translating the human ones to mouse.
- **inputargs** (*dict*) – Additional arguments passed to *PtmProcessor*. A *dict* can be supplied for each resource, e.g. *{'Signor': {...}, 'PhosphoSite': {...}, ...}*. Those not used by *PtmProcessor* are forwarded to the *pypath.dataio* methods.

load_reflist(*reflist*)

load_reflists(*reflst=None*)

load_resource(*settings*, *clean=True*, *cache_files={}*, *reread=False*, *redownload=False*)

load_resources(*lst=None*, *exclude=[]*, *cache_files={}*, *reread=False*, *redownload=False*)

Loads multiple resources, and cleans up after. Looks up ID types, and loads all ID conversion tables from UniProt if necessary. This is much faster than loading the ID conversion and the resources one by one.

load_signor_ptms(*non_matching=False*, *trace=False*, ***kwargs*)

load_tfregulons(*levels=set(['A', 'B'])*, *only_curated=False*)

Adds TF-target interactions from TF regulons to the network.

Parameters

- **levels** (*set*) – Confidence levels to be used.
- **only_curated** (*bool*) – Retrieve only literature curated interactions.

TF regulons is a comprehensive resource of TF-target interactions combining multiple lines of evidences: literature curated databases, ChIP-Seq data, PWM based prediction using HOCOMOCO and JASPAR matrices and prediction from GTEx expression data by ARACNe.

For details see https://github.com/saezlab/DoRothEA.

```
>>> import pypath
>>> pa = pypath.PyPath()
>>> pa.load_tfregulons(levels = {'A'})
```

lookup_cache(*name*, *cache_files*, *int_cache*, *edges_cache*)
Checks up the cache folder for the files of a given resource. First checks if *name* is on the *cache_files* dictionary. If so, loads either the interactions or edges otherwise. If not, checks *edges_cache* or *int_cache* otherwise.

Parameters

- **name** (*str*) – Name of the resource (lower-case).
- **cache_files** (*dict*) – Contains the resource name(s) [str] (keys) and the corresponding cached file name [str] (values).
- **int_cache** (*str*) – Path to the interactions cache file of the resource.
- **edges_cache** (*str*) – Path to the edges cache file of the resource.

Returns

- (*file*) – The loaded pickle file from the cache if the file is contains the interactions. ‘‘None‘self.graph‘ otherwise.
- (*list*) – List of mapped edges if the file contains the information from the edges. `[]` otherwise.

loop_edges(*index=True*, *graph=None*)

map_edge(*edge*)
Translates the name in *edge* representing an edge. Default name types are defined in `pypath.main.PyPath.default_name_type` If the mapping is unsuccessful, the item will be added to `pypath.main.PyPath.unmapped` list.

Parameters **edge** (*dict*) – Item whose name is to be mapped to a default name type.

Returns (*list*) – Contains the edge(s) [dict] with default mapped names.

map_item(*item*)
Translates the name in *item* representing a molecule. Default name types are defined in `pypath.main.PyPath.default_name_type` If the mapping is unsuccessful, the item will be added to `pypath.main.PyPath.unmapped` list.

Parameters **item** (*dict*) – Item whose name is to be mapped to a default name type.

Returns (*list*) – The default mapped name(s) [str] of *item*.

map_list(*lst*, *singleList=False*)
Maps the names from a list of edges or items (molecules).

Parameters

- **lst** (*list*) – List of items or edge dictionaries whose names have to be mapped.
- **singleList** (*bool*) – Optional, `False` by default. Determines whether the provided elements are items or edges. This is, either calls *pypath.main.PyPath.map_edge()* or *pypath.main.PyPath.map_item()* to map the item names.

Returns (*list*) – Copy of *lst* with their elements’ names mapped.

mean_reference_per_interaction()
Computes the mean number of references per interaction of the network.

Returns (*float*) – Mean number of interactions per edge.

merge_lists(*nameA*, *nameB*, *name=None*, *and_or='and'*, *delete=False*, *func='max'*)
Merges two lists in *lists*.

merge_nodes(*nodes*, *primary=None*, *graph=None*)
Merges all attributes and all edges of selected nodes and assigns them to the primary node (by default the one with lowest ID).

Parameters

- **nodes** (*list*) – List of edge IDs.
- **primary** (*int*) – ID of the primary edge; if None the lowest ID selected.

mimp_directions(*graph=None*)

mutated_edges(*sample*)
Compares the mutated residues and the modified residues in PTMs. Interactions are marked as mutated if the target residue in the underlying PTM is mutated.

names2vids(*names*)

negative_report(*lst=True*, *outFile=None*)

neighborhood(*identifiers*, *order=1*, *mode='ALL'*)

neighbors(*identifier*, *mode='ALL'*)

neighbourhood_network(*center*, *second=False*)

network_filter(*p=2.0*)
This function aims to cut the number of edges in the network, without loosing nodes, to make the network less connected, less hairball-like, more usable for analysis.

network_stats(*outfile=None*)
Calculates basic statistics for the whole network and each of sources. Writes the results in a tab file.

new_edges(*edges*)

new_nodes(*nodes*)

node_exists(*name*)

numof_directed_edges()

numof_reference_interaction_pairs()
Returns the total of unique references per interaction.

Returns (*int*) – Total number of unique references per interaction.

numof_references()
Counts the number of reference on the network.

Counts the total number of unique references in the edges of the network.

Returns (*int*) – Number of unique references in the network.

numof_undirected_edges()

orthology_translation(*target*, *source=None*, *only_swissprot=True*, *graph=None*)
Translates the current object to another organism by orthology. Proteins without known ortholog will be deleted.

Parameters **target** (*int*) – NCBI Taxonomy ID of the target organism. E.g. 10090 for mouse.

p(*identifier*)
Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

pathway_attributes(*graph=None*)

pathway_members(*pathway*, *source*)
Returns an iterator with the members of a single pathway. Apart from the pathway name you need to supply its source database too.

pathway_names(*source*, *graph=None*)
Returns the names of all pathways having at least one member in the current graph.

pathway_similarity(*outfile=None*)

pathways_table(*filename='genes_pathways.list', pw_sources=['signalink', 'signor', 'netpath', 'kegg'], graph=None*)

pfam_regions()

phosphonetworks_directions(*graph=None*)

phosphopoint_directions(*graph=None*)

phosphorylation_directions()

phosphorylation_signs()

phosphosite_directions(*graph=None*)

prdb_tissue_expr(*tissue*, *prdb=None*, *graph=None*, *occurrence=1*, *group_function=<function <lambda>>*, *na_value=0.0*)

process_direction(*line*, *dirCol*, *dirVal*, *dirSep*)
Processes the direction information of an interaction according to a data file from a source.

Parameters

- **line** (*list*) – The stripped and separated line from the resource data file containing the information of an interaction.
- **dirCol** (*int*) – The column/position number where the information about the direction is to be found (on *line*).
- **dirVal** (*list*) – Contains the terms [str] for which that interaction is to be considered directed.
- **dirSep** (*str*) – Separator for the field in *line* containing the direction information (if any).

Returns (*bool*) – Determines whether the given interaction is directed or not.

process_directions(*dirs*, *name*, *directed=None*, *stimulation=None*, *inhibition=None*, *graph=None*, *id_type=None*, *dirs_only=False*)

process_dmi(*source*, ***kwargs*)
This is an universal function for loading domain-motif objects like load_phospho_dmi() for PTMs. TODO this will replace load_elm, load_ielm, etc

process_sign(*signData*, *signDef*)
Processes the sign of an interaction, used when processing an input file.

Parameters

- **signData** (*str*) – Data regarding the sign to be processed.
- **signDef** (*tuple*) – Contains information about how to process *signData*. This is defined in `pypath.data_formats`. First element determines the position on the direction information of each line on the data file [int], second element is either [str] or [list] and defines the terms for which an interaction is defined as stimulation, third element is similar but for the inhibition and third (optional) element determines the separator for *signData* if contains more than one element.

Returns

- (*bool*) – Determines whether the processed interaction is considered stimulation or not.
- (*bool*) – Determines whether the processed interaction is considered inhibition or not.

protein (*identifier*)
Same as `PyPath.get_node`, just for the directed graph. Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default directed graph, or a UniProt ID or GeneSymbol which can be found in the default directed network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

protein_edge (*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

Parameters

- **source** (*int,str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **target** (*int,str*) – Vertex index or UniProt ID or GeneSymbol or `igraph.Vertex` object.
- **directed** (*bool*) – To be passed to igraph.Graph.get_eid()

proteins (*identifiers*)

proteome_list (*swissprot=True*)
Loads the whole proteome as a list.

Parameters **swissprot** (*bool*) – Optional, `True` by default. Determines whether to use also the information from SwissProt.

ps (*identifiers*)

random_walk_with_return (*q*, *graph=None*, *c=0.5*, *niter=1000*)
Random walk with return (RWR) starting from one or more query nodes. Returns affinity (probability) vector of all nodes in the graph.

param int,list q Vertex IDs of query nodes.

param igraph.Graph graph An *igraph.Graph* object.

param float c Probability of restart.

param int niter Number of iterations.

```
>>> import igraph
>>> import pypath
>>> pa = pypath.PyPath()
>>> pa.init_network({
        'signor': pypath.data_formats.pathway['signor']
    })
```

(continues on next page)

(continued from previous page)

```
>>> q = [
        pa.gs('EGFR').index,
        pa.gs('ATG4B').index
    ]
>>> rwr = pa.random_walk_with_return(q = q)
>>> palette = igraph.RainbowPalette(n = 100)
>>> colors  = [palette.get(int(round(i))) for i in rwr / max(rwr) * 99]
>>> igraph.plot(pa.graph, vertex_color = colors)
```

random_walk_with_return2(*q*, *c=0.5*, *niter=1000*)
Literally does random walks. Only for testing of the other method, to be deleted later.

read_data_file(*settings*, *keep_raw=False*, *cache_files={}*, *reread=False*, *redownload=False*)
Reads interaction data file containing node and edge attributes that can be read from simple text based files and adds it to the networkdata. This function works not only with files, but with lists as well. Any other function can be written to download and preprocess data, and then give it to this function to finally attach to the network.

Parameters

- **settings** (*pypath.input_formats.ReadSettings*) – `pypath.input_formats.ReadSettings` instance containing the detailed definition of the input format of the file. Instead of the file name (on the `pypath.input_formats.ReadSettings.inFile` attribute) you can give a custom function name, which will be executed, and the returned data will be used instead.
- **keep_raw** (*bool*) – Optional, `False` by default. Whether to keep the raw data read by this function, in order for debugging purposes, or further use.
- **cache_files** (*dict*) – Optional, `{}` by default. Contains the resource name(s) [str] (keys) and the corresponding cached file name [str]. If provided (and file exists) bypasses the download of the data for that resource and uses the cache file instead.
- **reread** (*bool*) – Optional, `False` by default. Specifies whether to reread the data files from the cache or omit them (similar to *redownload*).
- **redownload** (*bool*) – Optional, `False` by default. Specifies whether to re-download the data and ignore the cache.

read_from_cache(*cache_file*)
Reads a pickle file from the cache and returns it. It is assumed that the subfolder `cache/` is on the supplied path.

Parameters **cache_file** (*str*) – Path to the cache file that is to be loaded.

Returns (*file*) – The loaded pickle file from the cache. Type will depend on the file itself (e.g.: if the pickle was saved from a dictionary, the type will be [dict]).

read_list_file(*settings*, ***kwargs*)
Reads a list from a file and adds it to `pypath.main.PyPath.lists`.

Parameters

- **settings** (*pypath.input_formats.ReadList*) – `python.data_formats.ReadList` instance specifying the settings of the file to be read. See the class documentation for more details.
- ****kwargs** – Extra arguments passed to the file reading function. Such function name is outlined in the `python.data_formats.ReadList.inFile` attribute and defined in `pypath.dataio`.

receptors_list()
Loads the Human Plasma Membrane Receptome as a list. This resource is human only.

reference_edge_ratio()
Computes the average number of references per edge (as in the undirected graph).

Returns (*float*) – Average number of references per edge.

reference_hist(*filename=None*)

reload()

remove_htp(*threshold=50*, *keep_directed=False*)

remove_undirected(*min_refs=None*)

run_batch(*methods*, *toCall=None*)

save_network(*pfile=None*)
Saves the network object.

Stores the instance into a pickle (binary) file which can be reloaded in the future.

Parameters **pfile** (*str*) – Optional, `None` by default. The path/file name where to store the pcikle file. If not specified, saves the network to its default location (`'cache/default_network.pickle'`).

save_session()
Save current state into pickle dump.

search_attr_and(*obj*, *lst*)

search_attr_or(*obj*, *lst*)

second_neighbours(*node*, *indices=False*, *with_first=False*)

select_by_go(*go_terms*, *go_desc=None*, *aspects=('C', 'F', 'P')*, *method='ANY'*)
Selects the nodes annotated by certain GO terms.

Returns set of vertex IDs.

Parameters **method** (*str*) – If *ANY* nodes annotated with any of the terms returned. If *ALL* nodes annotated with all the terms returned.

separate()
Separates networks from different sources. Returns dict of igraph objects.

separate_by_category()
Separate networks based on categories. Returns dict of igraph objects.

sequences(*isoforms=True*, *update=False*)

set_boolean_vattr(*attr*, *vids*, *negate=False*)

set_categories()

set_chembl_mysql(*title*, *config_file=None*)
Sets the ChEMBL MySQL configuration according to the *title* section in *config_file* ini file configuration.

Parameters

- **title** (*str*) – Section title of the ini file.
- **config_file** (*str*) – Optional, `None` by default. Specifies the configuration file name if none is passed, `mysql_config/defaults.mysql` will be used.

set_disease_genes(*dataset='curated'*)
Creates a vertex attribute named *dis* with boolean values *True* if the protein encoded by a disease related gene according to DisGeNet.

Parameters **dataset** (*str*) – Which dataset to use from DisGeNet. Default is *curated*.

set_druggability()
Creates a vertex attribute *dgb* with value *True* if the protein is druggable, otherwise *False*.

set_drugtargets(*pchembl=5.0*)
Creates a vertex attribute *dtg* with value *True* if the protein has at least one compound binding with affinity higher than *pchembl*, otherwise *False*.

Parameters **pchembl** (*float*) – Pchembl threshold.

set_kinases()
Creates a vertex attribute *kin* with value *True* if the protein is a kinase, otherwise *False*.

set_receptors()
Creates a vertex attribute *rec* with value *True* if the protein is a receptor, otherwise *False*.

set_signaling_proteins()
Creates a vertex attribute *kin* with value *True* if the protein is a kinase, otherwise *False*.

set_tfs(*classes=['a', 'b', 'other']*)

set_transcription_factors(*classes=['a', 'b', 'other']*)
Creates a vertex attribute *tf* with value *True* if the protein is a transcription factor, otherwise *False*.

Parameters **classes** (*list*) – Classes to use from TF Census. Default is *['a', 'b', 'other']*.

shortest_path_dist(*graph=None*, *subset=None*, *outfile=None*, ***kwargs*)
subset is a tuple of two lists if you wish to look for paths between elements of two groups, or a list if you wish to look for shortest paths within this group

signaling_proteins_list()
Compiles a list of signaling proteins (as opposed to other proteins like metabolic enzymes, matrix proteins, etc), by looking up a few simple keywords in short description of GO terms.

signor_pathways(*graph=None*)

similarity_groups(*groups*, *index='simpson'*)

small_plot(*graph*, ***kwargs*)
This method is deprecated, do not use it.

sorensen_pathways(*pwlist=None*)

source_diagram(*outf=None*, ***kwargs*)

source_network(*font='HelveticaNeueLTStd'*)
For EMBL branding, use Helvetica Neue Linotype Standard light

source_similarity(*outfile=None*)

source_stats()

sources_hist()

sources_overlap(*diagonal=False*)

sources_venn_data(*fname=None*, *return_data=False*)

straight_between(*nameA*, *nameB*)
This does actually the same as get_edge(), but by names instead of vertex ids.

string_effects(*graph=None*)

sum_in_complex(*csources=['corum'], graph=None*)
Returns the total number of edges in the network falling between two members of the same complex. Returns as a dict by complex resources. Calls :py:func:pypath.pypath.Pypath.edges_in_comlexes() to do the calculations.

@csources [list] List of complex resources. Should be already loaded.

@graph [igraph.Graph()] The graph object to do the calculations on.

table_latex(*fname*, *header*, *data*, *sum_row=True*, *row_order=None*, *latex_hdr=True*, *caption=''*, *font='HelveticaNeueLTStd-LtCn'*, *fontsize=8*, *sum_label='Total'*, *sum_cols=None*, *header_format='%s'*, *by_category=True*)

tfs_list()
Loads the list of all known transcription factors from TF census (Vaquerizas 2009). This resource is human only.

third_source_directions(*graph=None*, *use_string_effects=False*, *use_laudanna_data=False*)
This method calls a series of methods to get additional direction & effect information from sources having no literature curated references, but giving sufficient evidence about the directionality for interactions already supported by literature evidences from other sources.

tissue_network(*tissue*, *graph=None*)
Returns a network which includes the proteins expressed in certain tissue according to ProteomicsDB.

Parameters

- **tissue** (*str*) – Tissue name as used in ProteomicsDB.
- **graph** (*igraph.Graph*) – A graph object, by default the *graph* attribute of the current instance.

transcription_factors()

translate_refsdir(*rd*, *ids*)

uniprot(*uniprot*)
Returns `igraph.Vertex()` object if the UniProt can be found in the default undirected network, otherwise `None`.

@uniprot [str] UniProt ID.

uniprots(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default undirected graph.

uniq_node_list(*lst*)
Returns a given list of nodes containing only the unique elements.

Parameters **lst** (*list*) – List of nodes.

Returns (*list*) – Copy of *lst* containing only unique nodes.

uniq_ptm(*ptms*)

uniq_ptms()

up(*uniprot*)
Returns `igraph.Vertex()` object if the UniProt can be found in the default undirected network, otherwise `None`.

@uniprot [str] UniProt ID.

up_affected_by(*uniprot*)

up_affects(*uniprot*)

up_edge(*source*, *target*, *directed=True*)
Returns `igraph.Edge` object if an edge exist between the 2 proteins, otherwise `None`.

@source [str] UniProt ID

@target [str] UniProt ID

@directed [bool] To be passed to igraph.Graph.get_eid()

up_in_directed(*uniprot*)

up_in_undirected(*uniprot*)

up_inhibited_by(*uniprot*)

up_inhibits(*uniprot*)

up_neighborhood(*uniprot*, *order=1*, *mode='ALL'*)

up_neighbors(*uniprot*, *mode='ALL'*)

up_stimulated_by(*uniprot*)

up_stimulates(*uniprot*)

update_adjlist(*graph=None*, *mode='ALL'*)
Creates an adjacency list in a list of sets format.

update_attrs()

update_cats()
Makes sure that the *has_cats* attribute is an up to date set of all categories in the current network.

update_db_dict()

update_pathway_types()

update_pathways()

update_sources()
Makes sure that the *sources* attribute is an up to date list of all sources in the current network.

update_vertex_sources()

update_vindex()
This is deprecated.

update_vname()
Fast lookup of node names and indexes, these are hold in a [list] and a [dict] as well. However, every time new nodes are added, these should be updated. This function is automatically called after all operations affecting node indices.

ups(*uniprots*)
Returns list of `igraph.Vertex()` object for a list of UniProt IDs omitting those could not be found in the default undirected graph.

v(*identifier*)
Returns `igraph.Vertex()` object if the identifier is a valid vertex index in the default undirected graph, or a UniProt ID or GeneSymbol which can be found in the default undirected network, otherwise `None`.

@identifier [int, str] Vertex index (int) or GeneSymbol (str) or UniProt ID (str) or `igraph.Vertex` object.

vertex_pathways()
Some resources assignes interactions some others proteins to pathways. This function copies pathway annotations from edge attributes to vertex attributes.

vs(*identifiers*)

vsgs()
Returns a generator sequence of the node names as GeneSymbols [str] (from the undirected graph).

Returns (*generator*) – Sequence containing the node names as GeneSymbols [str].

vsup()
Returns a generator sequence of the node names as UniProt IDs [str] (from the undirected graph).

Returns (*generator*) – Sequence containing the node names as UniProt IDs [str].

wang_effects(*graph=None*)

write_table(*tbl*, *outfile*, *sep='\t'*, *cut=None*, *colnames=True*, *rownames=True*)

class pypath.main.**Direction**(*nameA*, *nameB*)
Object storing directionality information of an edge. Also includes information about the reverse direction, mode of regulation and sources of that information.

Parameters

- **nameA** (*str*) – Name of the source node.
- **nameB** (*str*) – Name of the target node.

Variables

- **dirs** (*dict*) – Dictionary containing the presence of directionality of the given edge. Keys are *straight*, *reverse* and `'undirected'` and their values denote the presence/absence [bool].
- **negative** (*dict*) – Dictionary contianing the presence/absence [bool] of negative interactions for both `straight` and `reverse` directions.
- **negative_sources** (*dict*) – Contains the resource names [str] supporting a negative interaction on `straight` and `reverse` directions.
- **nodes** (*list*) – Contains the node names [str] sorted alphabetically (*nameA*, *nameB*).
- **positive** (*dict*) – Dictionary contianing the presence/absence [bool] of positive interactions for both `straight` and `reverse` directions.
- **positive_sources** (*dict*) – Contains the resource names [str] supporting a positive interaction on `straight` and `reverse` directions.
- **reverse** (*tuple*) – Contains the node names [str] in reverse order e.g. (*nameB*, *nameA*).
- **sources** (*dict*) – Contains the resource names [str] of a given edge for each directionality (`straight`, `reverse` and `'undirected'`). Values are sets containing the names of those resources supporting such directionality.
- **straight** (*tuple*) – Contains the node names [str] in the original order e.g. (*nameA*, *nameB*).

check_nodes(*nodes*)
Checks if *nodes* is contained in the edge.

Parameters **nodes** (*list*) – Or [tuple], contains the names of the nodes to be checked.

Returns (*bool*) – `True` if all elements in *nodes* are contained in the object `nodes` list.

check_param(*di*)
Checks if *di* is `'undirected'` or contains the nodes of the current edge. Used internally to check that *di* is a valid key for the object attributes declared on dictionaries.

Parameters **di** (*tuple*) – Or [str], key to be tested for validity.

Returns

(*bool*) – `True` if *di* is `'undirected'` or a tuple of node names contained in the edge, `False` otherwise.

consensus_edges()
Infers the consensus edge(s) according to the number of supporting sources. This includes direction and sign.

Returns (*list*) – Contains the consensus edge(s) along with the consensus sign. If there is no major directionality, both are returned. The structure is as follows: `['<source>', '<target>', '<(un)directed>', '<sign>']`

get_dir(*direction*, *sources=False*)
Returns the state (or *sources* if specified) of the given *direction*.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`). Pair of nodes from which direction information is to be retrieved.
- **sources** (*bool*) – Optional, `'False'` by default. Specifies if the `sources` information of the given direction is to be retrieved instead.

Returns (*bool* or *set*) – (if `sources=True`). Presence/absence of the requested direction (or the list of sources if specified). Returns `None` if *direction* is not valid.

get_dirs(*src*, *tgt*, *sources=False*)
Returns all directions with boolean values or list of sources.

Parameters

- **src** (*str*) – Source node.
- **tgt** (*str*) – Target node.
- **sources** (*bool*) – Optional, `False` by default. Specifies whether to return the `sources` attribute instead of `dirs`.

Returns Contains the `dirs` (or `sources` if specified) of the given edge.

get_sign(*direction*, *sign=None*, *sources=False*)
Retrieves the sign information of the edge in the given diretion. If specified in *sign*, only that sign's information will be retrieved. If specified in *sources*, the sources of that information will be retrieved instead.

Parameters

- **direction** (*tuple*) – Contains the pair of nodes specifying the directionality of the edge from which th information is to be retrieved.
- **sign** (*str*) – Optional, `None` by default. Denotes whether to retrieve the `'positive'` or `'negative'` specific information.
- **sources** (*bool*) – Optional, `False` by default. Specifies whether to return the sources instead of sign.

Returns (*list*) – If `sign=None` containing [bool] values denoting the presence of positive and negative sign on that direction, if `sources=True` the [set] of sources for each of them will be returned instead. If *sign* is specified, returns [bool] or [set] (if `sources=True`) of that specific direction and sign.

has_sign(*direction=None*)
Checks whether the edge (or for a specific *direction*) has any signed information (about positive/negative interactions).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, only the information of that direction is checked for sign.

Returns

(*bool*) – `True` if there exist any information on the sign of the interaction, `False` otherwise.

is_directed()
Checks if edge has any directionality information.

Returns (*bool*) – Returns `True```if any of the :py:attr:`dirs` attribute values is ``True` (except `'undirected'`), `False` otherwise.

is_inhibition(*direction=None*)
Checks if any (or for a specific *direction*) interaction is inhibition (negative interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `negative` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is inhibitory (negative).

is_stimulation(*direction=None*)
Checks if any (or for a specific *direction*) interaction is activation (positive interaction).

Parameters **direction** (*tuple*) – Optional, `None` by default. If specified, checks the `positive` attribute of that specific directionality. If not specified, checks both.

Returns (*bool*) – `True` if any interaction (or the specified *direction*) is activatory (positive).

majority_dir()
Infers which is the major directionality of the edge by number of supporting sources.

Returns (*tuple*) – Contains the pair of nodes denoting the consensus directionality. If the number of sources on both directions is equal, `None` is returned. If there is no directionality information, `'undirected'`` will be returned.

majority_sign()
Infers which is the major sign (activation/inhibition) of the edge by number of supporting sources on both directions.

Returns (*dict*) – Keys are the node tuples on both directions (`straight`/`reverse`) and values can be either `None` if that direction has no sign information or a list of two [bool] elements corresponding to majority of positive and majority of negative support. In case both elements of the list are `True`, this means the number of supporting sources for both signs in that direction is equal.

merge(*other*)
Merges current edge with another (if and only if they are the same class and contain the same nodes). Updates the attributes `dirs`, `sources`, `positive`, `negative`, `positive_sources` and `negative_sources`.

Parameters **other** (`pypath.main.Direction`) – The new edge object to be merged with the current one.

negative_reverse()
Checks if the reverse directionality is a negative interaction.

Returns (*bool*) – True if there is supporting information on the reverse direction of the edge as inhibition. False otherwise.

negative_sources_reverse()
Retrieves the list of sources for the reverse direction and negative sign.

Returns (*set*) – Contains the names of the sources supporting the reverse directionality of the edge with a negative sign.

negative_sources_straight()
Retrieves the list of sources for the straight direction and negative sign.

Returns (*set*) – Contains the names of the sources supporting the straight directionality of the edge with a negative sign.

negative_straight()
Checks if the straight directionality is a negative interaction.

Returns (*bool*) – True if there is supporting information on the straight direction of the edge as inhibition. False otherwise.

positive_reverse()
Checks if the reverse directionality is a positive interaction.

Returns (*bool*) – True if there is supporting information on the reverse direction of the edge as activation. False otherwise.

positive_sources_reverse()
Retrieves the list of sources for the reverse direction and positive sign.

Returns (*set*) – Contains the names of the sources supporting the reverse directionality of the edge with a positive sign.

positive_sources_straight()
Retrieves the list of sources for the straight direction and positive sign.

Returns (*set*) – Contains the names of the sources supporting the straight directionality of the edge with a positive sign.

positive_straight()
Checks if the straight directionality is a positive interaction.

Returns (*bool*) – True if there is supporting information on the straight direction of the edge as activation. False otherwise.

reload()
Reloads the object from the module level.

set_dir(*direction*, *source*)
Adds directionality information with the corresponding data source named. Modifies self attributes dirs and sources.

Parameters

- **direction** (*tuple*) – Or [str], the directionality key for which the value on dirs has to be set True.
- **source** (*set*) – Contains the name(s) of the source(s) from which such information was obtained.

set_sign(*direction*, *sign*, *source*)
Sets sign and source information on a given direction of the edge. Modifies the attributes `positive` and `positive_sources` or `negative` and `negative_sources` depending on the sign. Direction is also updated accordingly, which also modifies the attributes `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Pair of edge nodes specifying the direction from which the information is to be set/updated.
- **sign** (*str*) – Specifies the type of interaction. If `'positive'`, is considered activation, otherwise, is assumed to be negative (inhibition).
- **source** (*set*) – Contains the name(s) of the source(s) from which the information was obtained.

sources_reverse()
Retrieves the list of sources for the `reverse` direction.

Returns (*set*) – Contains the names of the sources supporting the `reverse` directionality of the edge.

sources_straight()
Retrieves the list of sources for the `straight` direction.

Returns (*set*) – Contains the names of the sources supporting the `straight` directionality of the edge.

sources_undirected()
Retrieves the list of sources without directed information.

Returns (*set*) – Contains the names of the sources supporting the edge presence but without specific directionality information.

src(*undirected=False*)
Returns the name(s) of the source node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the source node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

src_by_source(*source*)
Returns the name(s) of the source node(s) for each existing direction on the interaction for a specific *source*.

Parameters **source** (*str*) – Name of the source according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the source node(s) according to the specified *source*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

tgt(*undirected=False*)
Returns the name(s) of the target node(s) for each existing direction on the interaction.

Parameters **undirected** (*bool*) – Optional, `False` by default.

Returns (*list*) – Contains the name(s) for the target node(s). This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the interaction is undirected, an empty list will be returned.

tgt_by_source(*source*)
Returns the name(s) of the target node(s) for each existing direction on the interaction for a specific *source*.

Parameters **source** (*str*) – Name of the source according to which the information is to be retrieved.

Returns (*list*) – Contains the name(s) for the target node(s) according to the specified *source*. This means if the interaction is bidirectional, the list will contain both identifiers on the edge. If the specified *source* is not found or invalid, an empty list will be returned.

translate(*ids*)
Translates the node names/identifiers according to the dictionary *ids*.

Parameters **ids** (*dict*) – Dictionary containing (at least) the current names of the nodes as keys and their translation as values.

Returns (*pypath.main.Direction*) – The copy of current edge object with translated node names.

unset_dir(*direction*, *source=None*)
Removes directionality and/or source information of the specified *direction*. Modifies attribute `dirs` and `sources`.

Parameters

- **direction** (*tuple*) – Or [str] (if `'undirected'`) the pair of nodes specifying the directionality from which the information is to be removed.
- **source** (*set*) – Optional, `None` by default. If specified, determines which specific source(s) is(are) to be removed from `sources` attribute in the specified *direction*.

unset_sign(*direction*, *sign*, *source=None*)
Removes sign and/or source information of the specified *direction* and *sign*. Modifies attribute `positive` and `positive_sources` or `negative` and `negative_sources` (or `positive_attributes`/`negative_sources` only if `source=True`).

Parameters

- **direction** (*tuple*) – The pair of nodes specifying the directionality from which the information is to be removed.
- **sign** (*str*) – Sign from which the information is to be removed. Must be either `'positive'` or `'negative'`.
- **source** (*set*) – Optional, `None` by default. If specified, determines which source(s) is(are) to be removed from the sources in the specified *direction* and *sign*.

which_dirs()
Returns the pair(s) of nodes for which there is information about their directionality.

Returns (*list*) – List of tuples containing the nodes for which their attribute `dirs` is `True`.

class pypath.main.**AttrHelper**(*value*, *name=None*, *defaults={}*)
Attribute helper class.

- **Initialization arguments:**
  - *value* [dict/str]?:
  - *name* [str]?: Optional, `None` by default.
  - *defaults* [dict]:
- **Attributes:**
  - *value* [dict]?:

  - *name* [str]?:
  - *defaults* [dict]:
  - *id_type* [type]:

- **Call arguments:**
  - *instance* []:
  - *thisDir* [tuple?]: Optional, `None` by default.
  - *thisSign* []: Optional, `None` by default.
  - *thisDirSources* []: Optional, `None` by default.
  - *thisSources* []: Optional, `None` by default.

- **Returns:**
  -

CHAPTER

THREE

WEBSERVICE

New webservice from 14 June 2018: the queries slightly changed, have been largely extended. See the examples below.

One instance of the pypath webservice runs at the domain http://omnipathdb.org/, serving not only the OmniPath data but other datasets: TF-target interactions from TF Regulons, a large collection additional enzyme-substrate interactions, and literature curated miRNA-mRNA interacions combined from 4 databases. The webservice implements a very simple REST style API, you can make requests by HTTP protocol (browser, wget, curl or whatever).

The webservice currently recognizes 3 types of queries: `interactions`, `ptms` and `info`. The query types `resources`, `network` and `about` have not been implemented yet in the new webservice.

3.1 Mouse and rat

Except the miRNA interactions all interactions are available for human, mouse and rat. The rodent data has been translated from human using the NCBI Homologene database. Many human proteins have no known homolog in rodents hence rodent datasets are smaller than their human counterparts. Note, if you work with mouse omics data you might do better to translate your dataset to human (for example using the `pypath.homology` module) and use human interaction data.

3.2 Examples

A request without any parameter, gives some basic numbers about the actual loaded dataset:

> http://omnipathdb.org

The `info` returns a HTML page with comprehensive information about the resources:

> http://omnipathdb.org/info

The `interactions` query accepts some parameters and returns interactions in tabular format. This example returns all interactions of EGFR (P00533), with sources and references listed.

> http://omnipathdb.org/interactions/?partners=P00533&fields=sources,references

By default only the OmniPath dataset used, to query the TF Regulons or add the extra enzyme-substrate interactions you need to set additional parameters. For example to query the transcriptional regulators of EGFR:

> http://omnipathdb.org/interactions/?targets=EGFR&types=TF

The TF Regulons database assigns confidence levels to the interactions. You might want to select only the highest confidence, *A* category:

> http://omnipathdb.org/interactions/?targets=EGFR&types=TF&tfregulons_levels=A

Show the transcriptional targets of Smad2 homology translated to rat including the confidence levels from TF Regulons:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=type,ncbi_tax_id,tfregulons_level&organisms=10116&sources=Smad2&types=TF

Query interactions from PhosphoNetworks which is part of the *kinaseextra* dataset:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources&databases=PhosphoNetworks&datasets=kinaseextra

Get the interactions from Signor, SPIKE and SignaLink3:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&databases=Signor,SPIKE,SignaLink3

All interactions of MAP1LC3B:

> http://omnipathdb.org/interactions/?genesymbols=1&partners=MAP1LC3B

By default `partners` queries the interaction where either the source or the arget is among the partners. If you set the `source_target` parameter to `AND` both the source and the target must be in the queried set:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&sources=ATG3,ATG7,ATG4B,SQSTM1&targets=MAP1LC3B,MAP1LC3A,MAP1LC3C,Q9H0R8,GABARAP,GABARAPL2&source_target=AND

As you see above you can use UniProt IDs and Gene Symbols in the queries and also mix them. Get the miRNA regulating NOTCH1:

> http://omnipathdb.org/interactions/?genesymbols=1&fields=sources,references&datasets=mirnatarget&targets=NOTCH1

Note: with the exception of mandatory fields and genesymbols, the columns appear exactly in the order you provided in your query.

Another query type available is `ptms` which provides enzyme-substrate interactions. It is very similar to the `interactions`:

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references,isoforms&enzymes=FYN

Is there any ubiquitination reaction?

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=ubiquitination

And acetylation in mouse?

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&types=acetylation&organisms=10090

Rat interactions, both directly from rat and homology translated from human, from the PhosphoSite database:

> http://omnipathdb.org/ptms?genesymbols=1&fields=sources,references&organisms=10116&databases=PhosphoSite,PhosphoSite_noref

CHAPTER

FOUR

RELEASE HISTORY

Main improvements in the past releases:

4.1 0.1.0

- First release of pypath, for initial testing.

4.2 0.2.0

- Lots of small improvements in almost every module
- Networks can be read from local files, remote files, lists or provided by any function
- Almost all redistributed data have been removed, every source downloaded from the original provider.

4.3 0.3.0

- First version with partial Python 3 support.

4.4 0.4.0

- **pyreact** module with **BioPaxReader** and **PyReact** classes added
- Process description databases, BioPax and PathwayCommons SIF conversion rules are supported
- Format definitions for 6 process description databases included.

4.5 0.5.0

- Many classes have been added to the **plot** module
- All figures and tables in the manuscript can be generated automatically
- This is supported by a new module, **analysis**, which implements a generic

workflow in its **Workflow** class.

4.6 0.7.74

- **homology** module: finds the homologs of proteins using the NCBI

Homologene database and the homologs of PTM sites using UniProt sequences and PhosphoSitePlus homology table * **ptm** module: fully integrated way of processing enzyme-substrate interactions from many databases and their translation by homology to other species * **export** module: creates `pandas.DataFrame` or exports the network into tabular file * New webservice * TF Regulons database included and provides much more comprehensive transcriptional regulation resources, including literature curated, in silico predicted, ChIP-Seq and expression pattern based approaches * Many network resources added, including miRNA-mRNA and TF-miRNA interactions

4.7 Upcoming

- New, more flexible network reader class
- Full support for multi-species molecular interaction networks

(e.g. pathogene-host) * Better support for not protein only molecular interaction networks (metabolites, drug compounds, RNA) * ChEMBL webservice interface, interface for PubChem and eventually forDrugBank * Silent mode: a way to suppress messages and progress bars

CHAPTER

FIVE

FEATURES

The primary aim of **pypath** is to build up networks from multiple sources on one igraph object. **pypath** handles ambiguous ID conversion, reads custom edge and node attributes from text files and **MySQL**.

Submodules perform various features, e.g. graph visualization, working with rug compound data, searching drug targets and compounds in **ChEMBL**.

5.1 ID conversion

The ID conversion module `mapping` can be used independently. It has the feature to translate secondary UniProt IDs to primaries, and Trembl IDs to SwissProt, using primary Gene Symbols to find the connections. This module automatically loads and stores the necessary conversion tables. Many tables are predefined, such as all the IDs in **UniProt mapping service,** while users are able to load any table from **file** or **MySQL,** using the classes provided in the module `input_formats`.

5.2 Pathways

pypath includes data and predefined format descriptions for more than 25 high quality, literature curated databases. The inut formats are defined in the `data_formats` module. For some resources data downloaded on the fly, where it is not possible, data is redistributed with the module. Descriptions and comprehensive information about the resources is available in the `descriptions` module.

5.3 Structural features

One of the modules called `intera` provides many classes for representing structures and mechanisms behind protein interactions. These are `Residue` (optionally mutated), `Motif`, `Ptm`, `Domain`, `DomainMotif`, `DomainDomain` and `Interface`. All these classes have `__eq__()` methods to test equality between instances, and also `__contains__()` methods to look up easily if a residue is within a short motif or protein domain, or is the target residue of a PTM.

5.4 Sequences

The module `seq` contains a simple class for quick lookup any residue or segment in **UniProt** protein sequences while being aware of isoforms.

5.5 Tissue expression

For 3 protein expression databases there are functions and modules for downloading and combining the expression data with the network. These are the Human Protein Atlas, the ProteomicsDB and GIANT. The `giant` and `proteomicsdb` modules can be used also as stand alone Python clients for these resources.

5.6 Functional annotations

GSEA and **Gene Ontology** are two approaches for annotating genes and gene products, and enrichment analysis technics aims to use these annotations to highlight the biological functions a given set of genes is related to. Here the `enrich` module gives abstract classes to calculate enrichment statistics, while the `go` and the `gsea` modules give access to GO and GSEA data, and make it easy to count enrichment statistics for sets of genes.

5.7 Drug compounds

UniChem submodule provides an interface to effectively query the UniChem service, use connectivity search with custom settings, and translate SMILEs to ChEMBL IDs with ChEMBL web service.

ChEMBL submodule queries directly your own ChEMBL MySQL instance, has the features to search targets and compounds from custom assay types and relationship types, to get activity values, binding domains, and action types. You need to download the ChEMBL MySQL dump, and load into your own server.

5.8 Technical

MySQL submodule helps to manage MySQL connections and track queries. It is able to run queries parallely to optimize CPU and memory usage on the server, handling queues, and serve the result by server side or client side storage. The `chembl` and potentially the `mapping` modules rely on this `mysql` module.

The most important function in module `dataio` is a very flexible **download manager** built around `curl`. The function `dataio.curl()` accepts numerous arguments, tries to deal in a smart way with local **cache,** authentication, redirects, uncompression, character encodings, FTP and HTTP transactions, and many other stuff. Cache can grow to several GBs, and takes place in `./cache` by default. Please be aware of this, and use for example symlinks in case of using multiple working directories.

A simple **webservice** comes with this module: the `server` module based on `twisted.web.server` opens a custom port and serves plain text tables over HTTP with REST style querying.

CHAPTER

SIX

OMNIPATH IN R

You can download the data from the webservice and load into R. Look here for an example.

PYTHON MODULE INDEX

p

INDEX

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W