pypath Documentation

Release 0.7.97

Dénes Türei

Oct 22, 2018

CONTENTS:

- genindex
- modindex
- search

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*)

- **Attributes:**
  - *acsn_effects* []:
  - *add_genesets* []:
  - *add_grouped_eattr* []:
  - *add_grouped_set_eattr* []:
  - *add_list_eattr* []:
  - *add_set_eattr* []:
  - *add_update_edge* []:
  - *add_update_vertex* []:
  - *affected_by* []:
  - *affects* []:
  - *all_between* []:
  - *all_neighbours* []:
  - *apply_list* []:
  - *apply_negative* []:
  - *attach_network* []:
  - *basic_stats* []:
  - *basic_stats_intergroup* []:
  - *cancer_drivers_list* []:
  - *cancer_gene_census_list* []:
  - *clean_graph* []:
  - *collapse_by_name* []:
  - *combine_attr* []:
  - *communities* []:
  - *complex_comembership_network* []:
  - *complexes* []:
  - *complexes_in_network* []:
  - *compounds_from_chembl* []:
  - *consistency* []:
  - *copy* []:

- *copy_edges* []:
- *count_sol* []:
- *coverage* []:
- *curation_effort* []:
- *curation_stats* []:
- *curation_tab* []:
- *curators_work* []:
- *databases_similarity* []:
- *default_name_type* []:
- *degree_dist* []:
- *degree_dists* []:
- *delete_by_source* []:
- *delete_by_taxon* []:
- *delete_unknown* []:
- *delete_unmapped* []:
- *dgenesymbol* []:
- *dgenesymbols* []:
- *dgs* []:
- *dgss* []:
- *disease_genes_list* []:
- *dneighbors* []:
- *dp* []:
- *dproteins* []:
- *dps* []:
- *druggability_list* []:
- *duniprot* []:
- *duniprots* []:
- *dup* []:
- *dups* []:
- *dv* []:
- *dvs* []:
- *edge_exists* []:
- *edge_loc* []:
- *edge_names* []:
- *edges_3d* []:
- *edges_expression* []:

- *edges_in_comlexes* []:
- *edges_ptms* []:
- *edgeseq_inverse* []:
- *export_dot* []:
- *export_edgelist* []:
- *export_graphml* []:
- *export_ptms_tab* []:
- *export_sif* []:
- *export_struct_tab* []:
- *export_tab* []:
- *filters* []:
- *find_all_paths* []:
- *find_all_paths2* []:
- *find_complex* []:
- *first_neighbours* []:
- *fisher_enrichment* []:
- *geneset_enrichment* []:
- *genesymbol* []:
- *genesymbol_labels* []:
- *genesymbols* []:
- *get_attrs* []:
- *get_directed* []:
- *get_dirs_signs* []:
- *get_edge* []:
- *get_edges* []:
- *get_function* []:
- *get_giant* []:
- *get_max* []:
- *get_network* []:
- *get_node* []:
- *get_node_d* []:
- *get_node_pair* []:
- *get_nodes* []:
- *get_nodes_d* []:
- *get_pathways* []:
- *get_proteomicsdb* []:

- *get_sub* []:
- *get_taxon* []:
- *go_annotate* []:
- *go_dict* []:
- *go_enrichment* []:
- *gs* []:
- *gs_affected_by* []:
- *gs_affects* []:
- *gs_edge* []:
- *gs_in_directed* []:
- *gs_in_undirected* []:
- *gs_inhibited_by* []:
- *gs_inhibits* []:
- *gs_neighborhood* []:
- *gs_neighbors* []:
- *gs_stimulated_by* []:
- *gs_stimulates* []:
- *gss* []:
- *guide2pharma* []:
- *having_attr* []:
- *having_eattr* []:
- *having_ptm* []:
- *having_vattr* []:
- *homology_translation* []:
- *htp_stats* []:
- *in_complex* []:
- *in_directed* []:
- *in_undirected* []:
- *info* []:
- *init_complex_attr* []:
- *init_edge_attr* []:
- *init_gsea* []:
- *init_network* []:
- *init_vertex_attr* []:
- *intergroup_shortest_paths* []:
- *intogen_cancer_drivers_list* []:

- *jaccard_edges* []:
- *jaccard_meta* []:
- *kegg_directions* []:
- *kegg_pathways* []:
- *kinase_stats* []:
- *kinases_list* []:
- *label_by_go* []:
- *laudanna_directions* []:
- *laudanna_effects* []:
- *licence* []:
- *list_resources* []:
- *load_3dcomplexes* []:
- *load_3did_ddi* []:
- *load_3did_ddi2* []:
- *load_3did_dmi* []:
- *load_3did_interfaces* []:
- *load_all_pathways* []:
- *load_compleat* []:
- *load_complexportal* []:
- *load_comppi* []:
- *load_corum* []:
- *load_dbptm* []:
- *load_ddi* []:
- *load_ddis* []:
- *load_depod_dmi* []:
- *load_disgenet* []:
- *load_dmi* []:
- *load_dmis* []:
- *load_domino_dmi* []:
- *load_elm* []:
- *load_expression* []:
- *load_go* []:
- *load_havugimana* []:
- *load_hpa* []:
- *load_hprd_ptms* []:
- *load_ielm* []:

- *load_interfaces* []:
- *load_li2012_ptms* []:
- *load_ligand_receptor_network* []:
- *load_list* []:
- *load_lmpid* []:
- *load_mappings* []:
- *load_mimp_dmi* []:
- *load_mutations* []:
- *load_negatives* []:
- *load_old_omnipath* []:
- *load_omnipath* []:
- *load_pathways* []:
- *load_pdb* []:
- *load_pepcyber* []:
- *load_pfam* []:
- *load_pfam2* []:
- *load_pfam3* []:
- *load_phospho_dmi* []:
- *load_phosphoelm* []:
- *load_pisa* []:
- *load_pnetworks_dmi* []:
- *load_psite_phos* []:
- *load_psite_reg* []:
- *load_ptms* []:
- *load_ptms2* []:
- *load_reflist* []:
- *load_reflists* []:
- *load_resource* []:
- *load_resources* []:
- *load_signor_ptms* []:
- *load_tfregulons* []:
- *lookup_cache* []:
- *loop_edges* []:
- *map_edge* []:
- *map_item* []:
- *map_list* []:

- *mean_reference_per_interaction* []:
- *merge_lists* []:
- *merge_nodes* []:
- *mimp_directions* []:
- *mutated_edges* []:
- *names2vids* []:
- *negative_report* []:
- *neighborhood* []:
- *neighbors* []:
- *neighbourhood_network* []:
- *network_filter* []:
- *network_stats* []:
- *new_edges* []:
- *new_nodes* []:
- *node_exists* []:
- *numof_directed_edges* []:
- *numof_reference_interaction_pairs* []:
- *numof_references* []:
- *numof_undirected_edges* []:
- *orthology_translation* []:
- *p* []:
- *pathway_attributes* []:
- *pathway_members* []:
- *pathway_names* []:
- *pathway_similarity* []:
- *pathways_table* []:
- *pfam_regions* []:
- *phosphonetworks_directions* []:
- *phosphopoint_directions* []:
- *phosphorylation_directions* []:
- *phosphorylation_signs* []:
- *phosphosite_directions* []:
- *prdb_tissue_expr* []:
- *process_direction* []:
- *process_directions* []:
- *process_dmi* []:

- *process_sign* []:
- *protein* []:
- *protein_edge* []:
- *proteins* []:
- *proteome_list* []:
- *ps* []:
- *random_walk_with_return* []:
- *random_walk_with_return2* []:
- *read_data_file* []:
- *read_from_cache* []:
- *read_list_file* []:
- *receptors_list* []:
- *reference_edge_ratio* []:
- *reference_hist* []:
- *reload* []:
- *remove_htp* []:
- *remove_undirected* []:
- *run_batch* []:
- *save_network* []:
- *save_session* []:
- *search_attr_and* []:
- *search_attr_or* []:
- *second_neighbours* []:
- *select_by_go* []:
- *separate* []:
- *separate_by_category* []:
- *sequences* []:
- *set_boolean_vattr* []:
- *set_categories* []:
- *set_chembl_mysql* []:
- *set_disease_genes* []:
- *set_druggability* []:
- *set_drugtargets* []:
- *set_kinases* []:
- *set_receptors* []:
- *set_signaling_proteins* []:

- *set_tfs* []:
- *set_transcription_factors* []:
- *shortest_path_dist* []:
- *signaling_proteins_list* []:
- *signor_pathways* []:
- *similarity_groups* []:
- *small_plot* []:
- *sorensen_pathways* []:
- *source_diagram* []:
- *source_network* []:
- *source_similarity* []:
- *source_stats* []:
- *sources_hist* []:
- *sources_overlap* []:
- *sources_venn_data* []:
- *straight_between* []:
- *string_effects* []:
- *sum_in_complex* []:
- *table_latex* []:
- *tfs_list* []:
- *third_source_directions* []:
- *tissue_network* []:
- *transcription_factors* []:
- *translate_refsdir* []:
- *uniprot* []:
- *uniprots* []:
- *uniq_node_list* []:
- *uniq_ptm* []:
- *uniq_ptms* []:
- *up* []:
- *up_affected_by* []:
- *up_affects* []:
- *up_edge* []:
- *up_in_directed* []:
- *up_in_undirected* []:
- *up_inhibited_by* []:

- *up_inhibits* []:
- *up_neighborhood* []:
- *up_neighbors* []:
- *up_stimulated_by* []:
- *up_stimulates* []:
- *update_adjlist* []:
- *update_attrs* []:
- *update_cats* []:
- *update_db_dict* []:
- *update_pathway_types* []:
- *update_pathways* []:
- *update_sources* []:
- *update_vertex_sources* []:
- *update_vindex* []:
- *update_vname* []:
- *ups* []:
- *v* []:
- *vertex_pathways* []:
- *vs* []:
- *vsgs* []:
- *vsup* []:
- *wang_effects* []:
- *write_table* []:

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.

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.

apply_list(*name*, *node_or_edge='node'*)
Creates vertex or edge attribute based on a list.

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

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 with copying and merging all edges and attributes.

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
- dicts: calls *common.merge_dicts()*
- Direction: calls their special *merge()* method

Works on more than 2 attributes recursively.

Parameters

- **lst** (*list*) – List of one or two attribute values.
- **num_method** (*callable*) – Method to merge numeric attributes.

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.

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.

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.

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

@genesymbol [str] GeneSymbol.

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

@genesymbol [str] GeneSymbol.

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

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).

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).

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

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.

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_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.

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_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.

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`.

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_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
- **target** (*int, str*) – Vertex index or UniProt ID or GeneSymbol
- **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 or labels.
- **targets** (*iterable*) – Target vertec IDs, names or labels.
- **directed** (*bool*) – Passed to *igraph.get_eid()*.

get_giant (*replace=False*, *graph=None*)
Returns the giant component of the graph, or replaces the igraph object with only the giant component.

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).

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).

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.

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

@genesymbol [str] 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()

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.

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

init_network(*lst={'arn': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517d88>, 'biogrid': <pypath.input_formats.ReadSettings instance at 0x7f6bc528cef0>, 'ca1': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517c68>, 'ccmap': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c248>, 'cellphonedb': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a7050>, 'dbptm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c680>, 'death': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f38>, 'depod': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c4d0>, 'dip': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c2d8>, 'domino': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c5f0>, 'elm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c5a8>, 'guide2pharma': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b90>, 'hpmr': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a2f80>, 'hprd': <pypath.input_formats.ReadSettings instance at 0x7f6bc529b098>, 'hprd_p': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c6c8>, 'innatedb': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c368>, 'intact': <pypath.input_formats.ReadSettings instance at 0x7f6bc528cea8>, 'lmpid': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c518>, 'macrophage': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517ef0>, 'matrixdb': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c440>, 'mppi': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c290>, 'nrf2': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517e60>, 'pdz': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f80>, 'phelm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c560>, 'psite': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c488>, 'ramilowski2015': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a70e0>, 'signalink3': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b48>, 'signor': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c1b8>, 'spike': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b00>, 'trip': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517a28>}*, *exclude=[]*, *cache_files={}*, *pfile=False*, *save=False*, *reread=False*, *redownload=False*, ***kwargs*)
This is a lazy way to start the module, load data and build the high confidence, literature curated part of the signaling network.

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

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

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.

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

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

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_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_ligand_receptor_network(*lig_rec_resources=True*, *inference_from_go=True*, *sources={'arn': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517d88>, 'ca1': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517c68>, 'cellphonedb': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a7050>, 'death': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f38>, 'guide2pharma': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b90>, 'hpmr': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a2f80>, 'macrophage': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517ef0>, 'nrf2': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517e60>, 'pdz': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f80>, 'ramilowski2015': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a70e0>, 'signalink3': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b48>, 'signor': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c1b8>, 'spike': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b00>, 'trip': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517a28>}*, *keep_undirected=False*, *keep_rec_rec=False*, *keep_lig_lig=False*)

Initializes a ligand-receptor network.

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_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_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_resources(*lst={'arn': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517d88>, 'biogrid': <pypath.input_formats.ReadSettings instance at 0x7f6bc528cef0>, 'ca1': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517c68>, 'ccmap': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c248>, 'cellphonedb': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a7050>, 'dbptm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c680>, 'death': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f38>, 'depod': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c4d0>, 'dip': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c2d8>, 'domino': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c5f0>, 'elm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c5a8>, 'guide2pharma': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b90>, 'hpmr': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a2f80>, 'hprd': <pypath.input_formats.ReadSettings instance at 0x7f6bc529b098>, 'hprd_p': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c6c8>, 'innatedb': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c368>, 'intact': <pypath.input_formats.ReadSettings instance at 0x7f6bc528cea8>, 'lmpid': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c518>, 'macrophage': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517ef0>, 'matrixdb': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c440>, 'mppi': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c290>, 'nrf2': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517e60>, 'pdz': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517f80>, 'phelm': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c560>, 'psite': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c488>, 'ramilowski2015': <pypath.input_formats.ReadSettings instance at 0x7f6bc52a70e0>, 'signalink3': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b48>, 'signor': <pypath.input_formats.ReadSettings instance at 0x7f6bc528c1b8>, 'spike': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517b00>, 'trip': <pypath.input_formats.ReadSettings instance at 0x7f6bc5517a28>}*, *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_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'})
```

map_edge(*edge*)

Translates molecule names in dict representing an edge.

map_item(*item*)

Translates the name in item representing a molecule.

map_list (*lst*, *singleList=False*)
Only a wrapper for map_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.

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.

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.

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).

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.

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

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).

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

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

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

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']
    })
>>> 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*)
Interaction data with node and edge attributes can be read from simple text based files. This function works not only with files, but with lists as well. Any other function can be written to download a preprocess data, and then give it to this function to finally attach to the network.

@settings [ReadSettings instance] The detailed definition of the input format. Instead of the file name you can give a function name, which will be executed, and the returned data will be used.

@keep_raw [boolean] To keep the raw data read by this function, in order for debugging purposes, or further use.

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

save_session ()
Save current state into pickle dump.

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.

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

title (str): section title in ini file config_file (str, NoneType): config file name;

if None, the *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_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), by looking up a few simple keywords in short description of GO terms.

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

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

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

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.

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.

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.

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

@uniprot [str] UniProt ID.

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()

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

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

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

update_vindex()
This is deprecated.

update_vname()
For 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).

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

class pypath.main.**Direction**(*nameA*, *nameB*)
Object storing directionality information of an edge.

- **Arguments:**
  - *nameA* [str]: Name of the source node.
  - *nameB* [str]: Name of the target node.
- **Attributes:**
  - *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 the original order e.g. `(source, target)`.
  - *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 reverse order e.g. `(target, source)`.

consensus_edges()
Returns list of edges based on majority consensus of directions and signs.

get_dir(*direction*, *sources=False*)
Returns boolean or list of sources

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

majority_dir()
Returns directionality based on majority consensus. Returns *None* if the number of sources supporting the two opposite directions are the same. Returns *'undirected'* if there is no directionality information. Returns *tuple* of IDs if one direction is supported by more sources.

majority_sign()
Returns signs based on majority consensus. Keys in the returned *dict* are directions. Values are *None* if the direction lacks effect sign. Otherwise *tuples* with their first element *True* if the number of sources supporting stimulation in the given direction is greater or equal compared to those supporting inhibition. The second value is the same for inhibition.

set_dir(*direction*, *source*)
Adds directionality information with the corresponding data source named.

src(*undirected=False*)
Returns the IDs of effector molecules in this directed interaction. If the interaction is bidirectional, the list will contain 2 IDs. If the interaction is undirec- ted, an empty list will be returned.

tgt(*undirected=False*)
Returns the IDs of the target molecules in the inter- action. Same behaviour as *Direction.src()*.

unset_dir(*direction*, *source=None*)
Removes directionality information, or single source.

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

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

CHAPTER

ONE

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.

1.1 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.

1.2 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

TWO

CAN I USE OMNIPATH IN R?

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

https://github.com/saezlab/pypath/tree/master/r_import

CHAPTER

THREE

INSTALLATION

3.1 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.

3.2 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*.

3.3 Directly from git

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

3.4 With pip

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

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

3.5 Build source distribution

Clone the git repo, and run setup.py:

```
python setup.py sdist
```

3.6 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.

3.6.1 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.

3.7 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.

3.7.1 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.

3.7.2 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.

3.7.3 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

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 whith 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.

PYTHON MODULE INDEX

p

INDEX

M

N

O

P

R

S

T

U

V