pypath Documentation

Release 0.8.27

Dénes Türei

Jul 23, 2019

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.

documentation http://saezlab.github.io/pypath

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

MODULE REFERENCE

2.1 annot

2.2 bel

2.3 cellphonedb

2.4 complex

class pypath.complex.**AbstractComplexResource**(*name*, *ncbi_tax_id=9606*, *input_method=None*, *input_args=None*, *dump=None*, ***kwargs*)

A resource which provides information about molecular complexes.

class pypath.complex.**CellPhoneDB**(***kwargs*)

class pypath.complex.**Compleat**(*input_args=None*, ***kwargs*)

class pypath.complex.**ComplexAggregator**(*resources=None*, *pickle_file=None*)

reload()

Reloads the object from the module level.

class pypath.complex.**ComplexPortal**(*input_args=None*, ***kwargs*)

class pypath.complex.**Corum**(*input_args=None*, ***kwargs*)

class pypath.complex.**GuideToPharmacology**(*input_args=None*, ***kwargs*)

class pypath.complex.**Havugimana**(*input_args=None*, ***kwargs*)

class pypath.complex.**Hpmr**(*input_args=None*, ***kwargs*)

class pypath.complex.**Humap**(*input_args=None*, ***kwargs*)

class pypath.complex.**Pdb**(*input_args=None*, ***kwargs*)

class pypath.complex.**Signor**(*input_args=None*, ***kwargs*)

pypath.complex.**all_complexes**()

Returns a set of all complexes in the database which serves as a reference set for many methods, just like `uniprot_input.all_uniprots` represents the proteome.

pypath.complex.**get_db**(***kwargs*)

Retrieves the current database instance and initializes it if does not exist yet.

pypath.complex.**init_db**(***kwargs*)
Initializes or reloads the complex database. The database will be assigned to the db attribute of this module.

2.5 go

pypath.go.**annotate**(*graph*, *organism=9606*, *aspects=('C', 'F', 'P')*)
Adds Gene Ontology annotations to the nodes of a graph.

Parameters **graph** (*igraph.Graph*) – Any igraph.Graph object with uniprot IDs in its name vertex attribute.

pypath.go.**get_db**()
Retrieves the current database instance and initializes it if does not exist yet.

pypath.go.**init_db**()
Initializes or reloads the GO annotation database. The database will be assigned to the db attribute of this module.

pypath.go.**load_go**(*graph*, *organism=9606*, *aspects=('C', 'F', 'P')*)
Adds Gene Ontology annotations to the nodes of a graph.

Parameters **graph** (*igraph.Graph*) – Any igraph.Graph object with uniprot IDs in its name vertex attribute.

2.6 homology

2.7 intera

This module provides classes to represent and handle structural details of protein interactions i.e. residues, post-translational modifications, short motifs, domains, domain-motifs and domain-motif interactions, binding interfaces.

2.8 intercell

2.9 network

class pypath.network.**Interaction**(*id_a*, *id_b*, *type_a*, *type_b*, *directed*, *effect*, *type*, *sources*, *references*)

directed
Alias for field number 4

effect
Alias for field number 5

id_a
Alias for field number 0

id_b
Alias for field number 1

references
Alias for field number 8

sources
Alias for field number 7

type
Alias for field number 6

type_a
Alias for field number 2

type_b
Alias for field number 3

2.10 pdb

class pypath.pdb.**ResidueMapper**
This class stores and serves the PDB –> UniProt residue level mapping. Attempts to download the mapping, and stores it for further use. Converts PDB residue numbers to the corresponding UniProt ones.

clean()
Removes cached mappings, freeing up memory.

2.11 ptm

2.12 pyreact

class pypath.pyreact.**BioPaxReader**(*biopax*, *source*, *cleanup_period=800*, *file_from_archive=None*, *silent=False*)
This class parses a BioPAX file and exposes its content easily accessible for further processing. First it opens the file, if necessary it extracts from the archive. Then an *lxml.etree.iterparse* object is created, so the iteration is efficient and memory requirements are minimal. The iterparse object is iterated then, and for each tag included in the *BioPaxReader.methods* dict, the appropriate method is called. These me- thods extract information from the BioPAX entity, and store it in arbit- rary data structures: strings, lists or dicts. These are stored in dicts where keys are the original IDs of the tags, prefixed with the unique ID of the parser object. This is necessary to give a way to merge later the result of parsing more BioPAX files. For example, *id42* may identify EGFR in one file, but AKT1 in the other. Then, the parser of the first file has a unique ID of a 5 letter random string, the second parser a different one, and the molecules with the same ID can be distinguished at merging, e.g. EGFR will be *ffjh2@id42* and AKT1 will be *tr9gy@id42*. The methods and the resulted dicts are named after the BioPAX elements, sometimes abbreviated. For example, *BioPaxReader.protein()* processes the <*bp:Protein*> elements, and stores the results in *BioPaxReader.proteins*.

In its current state, this class does not parse every information and all BioPax entities. For example, nucleic acid related entities and interactions are omitted. But these easily can be added with minor mo- difications.

biopax_size()
Gets the uncompressed size of the BioPax XML. This is needed in order to have a progress bar. This method should not be called directly, `BioPaxReader.process()` calls it.

cleanup_hook()
Removes the used elements to free up memory. This method should not be called directly, `BioPaxReader.iterate()` calls it.

close_biopax()
Deletes the iterator and closes the file object. This method should not be called directly, `BioPaxReader.process()` calls it.

extract()
Extracts the BioPax file from compressed archive. Creates a temporary file. This is needed to trace the progress of processing, which is useful in case of large files. This method should not be called directly, `BioPaxReader.process()` calls it.

init_etree()
Creates the `lxml.etree.iterparse` object. This method should not be called directly, `BioPaxReader.process()` calls it.

iterate()
Iterates the BioPax XML and calls the appropriate methods for each element. This method should not be called directly, `BioPaxReader.process()` calls it.

open_biopax()
Opens the BioPax file. This method should not be called directly, `BioPaxReader.process()` calls it.

process(*silent=False*)
This method executes the total workflow of BioPax processing.

Parameters **silent** (*bool*) – whether to print status messages and progress bars.

set_progress()
Initializes a progress bar. This method should not be called directly, `BioPaxReader.process()` calls it.

2.13 seq

pypath.seq.**get_isoforms**(*organism=9606*)
Loads UniProt sequences for all isoforms.

pypath.seq.**read_fasta**(*fasta*)
Parses a fasta file. Returns dict with headers as keys and sequences as values.

pypath.seq.**swissprot_seq**(*organism=9606*, *isoforms=False*)
Loads all sequences for an organism, optionally for all isoforms, by default only first isoform.

2.14 unichem

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

T