pypath Documentation

Release 0.10.6

Dénes Türei

Feb 27, 2020

CONTENTS

Important: New module structure and new network API

Around the end of December we added a new network API to `pypath` which is not based on `igraph` any more and provides a modular and versatile access interface to the network data (since version `0.9`). In January we reorganized the submodules in `pypath` in order to create a clear structure (since version `0.10`). These are important milestones towards version `1.0` and we hope they will make `pypath` more convenient to use for everyone. By 18 February we merged these changes to the master branch however the *pypath guide* is still to be updated. Apologies for this inconvenience and please don't hesitate to ask questions by opening an issue on github. The old `igraph` based network class is still available in the `pypath.legacy` module.

Py2/3 Although we still keep the compatibility with Python 2, we don't test `pypath` in this environment and very few people uses it already. We highly recommend to use `pypath` in Python 3.6+.

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

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

contact omnipathdb@gmail.com

developers `pypath` is developed in the Saez Lab (http://saezlab.org) by Olga Ivanova, Nicolàs Palacio and Dénes Türei; the R package and the Cytoscape app are developed and maintained by Francesco Ceccarelli, Attila Gábor, Alberto Valdeolivas and Nicolàs Palacio.

pypath is a Python module for processing molecular biology data resources, combining them into databases and providing a versatile interface in Python as well as exporting the data for access through other platforms such as the R (the OmnipathR R/Bioconductor package), web service (at http://omnipathdb.org), Cytoscape (the OmniPath Cytoscape app) and BEL (Biological Expression Language).

pypath provides access to more than 100 resources! It builds 5 major combined databases and within these we can distinguish different datasets. The 5 major databases are interactions (molecular interaction network or pathways), enzyme-substrate relationships, protein complexes, molecular annotations (functional roles, localizations, and more) and inter-cellular communication roles.

pypath consists of a number of submodules and each of them again contains a number of submodules. Overall **pypath** consists of around 100 modules. The most important higher level submodules:

- *pypath.core:* contains the database classes e.g. network, complex, annotations, etc
- *pypath.inputs:* contains the resource specific methods which directly downlad and preprocess data from the original sources
- *pypath.omnipath:* higher level applications, e.g. a database manager, a web server
- *pypath.utils:* stand alone useful utilities, e.g. identifier translator, Gene Ontology processor, BioPax processor, etc

CHAPTER

ONE

WEBSERVICE

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

The webservice implements a very simple REST style API, you can make requests by the HTTP protocol (browser, wget, curl or whatever). After defining the query type and optionally a set of molecular entities (proteins) you can add further GET parameters encoded in the URL.

1.1 Query types

The webservice currently recognizes 7 types of queries: `interactions`, `enz_sub`, `annotations`, `complexes`, `intercell`, `queries` and `info`. The query types `resources`, `network` and `about` have not been implemented yet in the new webservice.

1.2 Interaction datasets

The instance of the `pypath` webserver running at the domain http://omnipathdb.org/, serves not only the OmniPath data but also other datasets. Each of them has a short name what you can use in the queries (e.g. `&datasets=omnipath,pathwayextra`).

- `omnipath`: the OmniPath data as defined in the paper, an arbitrary optimum between coverage and quality
- `pathwayextra`: activity flow interactions without literature reference
- `kinaseextra`: enzyme-substrate interactions without literature reference
- `ligrecextra`: ligand-receptor interactions without literature reference
- `tfregulons`: transcription factor (TF)-target interactions from DoRothEA
- `mirnatarget`: miRNA-mRNA and TF-miRNA interactions

TF-target interactions from TF Regulons, a large collection additional enzyme-substrate interactions, and literature curated miRNA-mRNA interacions combined from 4 databases.

1.3 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 do not have 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.

1.4 Examples

A request without any parameter provides the main webpage:

http://omnipathdb.org

The `info` returns a HTML page with comprehensive information about the resources. The list here should be and will be updated as currently OmniPath includes much more databases:

http://omnipathdb.org/info

1.4.1 Molecular interaction network

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 include any other dataset you have 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.

1.4.2 Enzyme-substrate interactions

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

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

Is there any ubiquitination reaction?

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

1.4.3 Molecular complexes

The `complexes` query provides a comprehensive database of more than 22,000 protein complexes. For example, to query all complexes from CORUM and PDB containing MTOR (P42345):

http://omnipathdb.org/complexes?proteins=P42345&databases=CORUM,PDB

1.4.4 Annotations

The `annotations` query provides a large variety of data about proteins, complexes and in the future other kinds of molecules. For example an annotation can tell if a protein is a kinase, or if it is expressed in the hearth muscle. These data come from dozens of databases and each kind of annotation record contains different fields. Because of this here we have a `record_id` field which is unique within the records of each database. Each row contains one key value pair and you need to use the `record_id` to connect the related key-value pairs. You can easily do this with `tidyr` and `dplyr` in R or `pandas` in Python. An example to query the pathway annotations from SignaLink:

http://omnipathdb.org/annotations?databases=SignaLink3

Or the tissue expression of BMP7 from Human Protein Atlas:

http://omnipathdb.org/annotations?databases=HPA_tissue&proteins=BMP7

1.4.5 Roles in inter-cellular communication

Another query type is `intercell` providing information on the roles in inter-cellular signaling. E.g. if a protein is a ligand, a receptor, an extracellular matrix (ECM) component, etc. This query type is very similar to `annotations` but here the data does not come from original sources but combined from several databases by us. However we refer also to the original databases whenever the `class_type` is `sub` (subclass). E.g. the main class `ligand` is a combination of `Ramilowski 2015`, `CellPhoneDB`, `HPMR` and many other databases, hence besides the

`ligand` category you will find sub-categories like `ligand_ramilowski`, `ligand_cellphonedb` and so on. An example how to get all intercell annotations for 4 selected proteins:

http://omnipathdb.org/intercell?proteins=EGFR,ULK1,ATG4A,BMP8B

Or all the main classes for one protein:

http://omnipathdb.org/intercell?levels=main&proteins=P00533

Or a list of all ECM proteins:

http://omnipathdb.org/intercell?categories=ecm

1.4.6 Exploring possible parameters

Sometimes the names and values of the query parameters are not intuitive, even though in many cases the server accepts multiple alternatives. To see the possible parameters with all possible values you can use the `queries` query type. The server checks the parameter names and values exactly against these rules and if any of them don't match you will get an error message instead of reply. To see the parameters for the `interactions` query:

http://omnipathdb.org/queries/interactions

CHAPTER

TWO

CAN I USE OMNIPATH IN R?

You can download the data from the webservice and load into R. Thanks to our colleague Attila Gabor we have a dedicated package for this:

https://github.com/saezlab/OmnipathR

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 can simply install **pypath** by **pip** (see below). If any non mandatory dependency is still missing, you can install them the usual way by *pip* or your package manager.

3.2 igraph C library, cairo and pycairo

For the legacy network class or the `igraph` conversion from the current network class *python-igraph* must be installed. *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

Recently the installation on Mac should not be more complicated than on Linux: you can simply install by **pip** (see above).

When `igraph` was a mandatory dependency and it didn't provide wheels the OS X installation was not straightforward primarily because cairo needs to be compiled from source. If you want igraph and cairo we provide two 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 to omit the last two. 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 the installation 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. 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.
- For Mac OS X (v >= 10.11 El Capitan) import of pypath fails with error: "libcurl link-time ssl backend (openssl) is different from compile-time ssl backend (none/other)". To fix it, you may need to reinstall pycurl library using special flags. More information and steps can be found here.

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 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 0.5.32

- *chembl*, *unichem*, *mysql* and *mysql_connect* modules made Python3 compatible

4.7 0.6.31

- Orthology translation of network
- Homologene UniProt dict to translate between different organisms UniProt-to-UniProt
- Orthology translation of PTMs
- Better processing of PhosphoSite regulatory sites

4.8 0.7.0

- TF-target, miRNA-mRNA and TF-miRNA interactions from many databases

4.9 0.7.74

- New web server based on *pandas* data frames
- New module *export* for generating data frames of interactions or enzyme-substrate interactions
- New module *websrvtab* for exporting data frames for the web server
- TF-target interactions from DoRothEA

4.10 0.7.93

- New *dataio* methods for Gene Ontology

4.11 0.7.110

- Many new docstrings

4.12 0.8

- New module *complex*: a comprehensive database of complexes
- New module *annot*: database of protein annotations (function, location)
- New module *intercell*: special methods for data integration focusing on intercellular communication
- New module *bel*: BEL integration
- Module *go* and all the connected *dataio* methods have been rewritten offering a workaround for data access despite GO's terrible web services and providing much more versatile query methods

- Removed MySQL support (e.g. loading mapping tables from MySQL)
- Modules *mapping*, *reflists*, *complex*, *ptm*, *annot*, *go* became services: these modules build databases and provide query methods, sometimes they even automatically delete data to free memory
- New interaction category in *data_formats*: *ligand_receptor*
- Improved logging and control over verbosity
- Better control over parameters by the *settings* module
- Many methods in *dataio* have been improved or fixed, docs and code style largely improved
- Started to add tests especially for methods in *dataio*

4.13 0.9

- The network database is not dependent any more on *python-igraph* hence it has been removed from the mandatory dependencies
- New API for the network, interactions, evidences, molecular entities

4.14 0.10.0

- New module structure: modules grouped into *core*, *inputs*, *internals*, *legacy*, *omnipath*, *resources*, *share* and *utils* submodules.

4.15 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)

CHAPTER

FIVE

FEATURES

Warning: The sections below are outdated, will be updated soon

In the beginning the primary aim of **pypath** was to build networks from multiple sources using an igraph object as the fundament of the integrated data structure. From version 0.7 and 0.8 this design principle started to change. Today **pypath** builds a number of different databases each having **pandas.DataFrame** as a final format. Each of these integrates a specific kind of data from various databases (e.g. protein complexes, interactions, enzyme-PTM relationships, etc). **pypath** has many submodules with standalone functionality which can be used in other modules and scripts. For example the ID conversion module **pypath.mapping**.

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 `utils.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 input 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 three 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

The module `pypath.curl` provides a very flexible **download manager** built on top of `pycurl`. The classes `pypath.curl.Curl()` and `pypath.curl.FileOpener` accept numerous arguments 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 `~/.pypath/cache` by default. If you experience issues using `pypath` these are most often related to failed downloads which often result nonsense cache contents. To debug such issues you can see the cache file names and cache usage in the log, and you can use the context managers in `pypath.curl` to show, delete or bypass the cache for some particular method calls (`pypath.curl.cache_print_on()`, `pypath.curl.cache_delete_on()` and `pypath.curl.cache_off()`. You can always set up an alternative cache directory for the entire session using the `pypath.settings` module.

The `pypath.session` and `pypath.log` modules take care of setting up session level parameters and logging. Each session has a random 5 character identifier e.g. `y5jzx`. The default log file in this case is `pypath_log/pypath-y5jzx.log`. The log messages are flushed every 2 seconds by default. You can always change these things using the `settings` module. In this module you can get and set the values of various parameters using the `pypath.settings.setup()` and the `pypath.settings.get()` methods.

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.