DyNetx Documentation

Release 0.3.0

Giulio Rossetti

Jun 06, 2023

Contents

coverage 93%

DyNetx is a Python software package that extends `networkx` with dynamic network models and algorithms.

Date	Python Versions	Main Author	GitHub	pypl
June 9, 2020	3.x	Giulio Rossetti	Source	Distribution

Contents:

CHAPTER 1

Overview

DyNetx is a Python language software package for describing, model, and study dynamic complex networks.

1.1 Who uses DyNetx?

The potential audience for DyNetx includes mathematicians, physicists, biologists, computer scientists, and social scientists.

1.2 Goals

DyNetx is built upon the NetworkX python library and is intended to provide:

- tools for the study dynamic social, biological, and infrastructure networks,
- a rapid development environment for collaborative, multidisciplinary, projects.

1.3 The Python DyNetx library

DyNetx is a powerful Python package that allows simple and flexible modelling of dynamic networks.

Most importantly, DyNetx, as well as the Python programming language, is free, well-supported, and a joy to use.

1.4 Free software

DyNetx is free software; you can redistribute it and/or modify it under the terms of the BSD License. We welcome contributions from the community.

CHAPTER 2

Download

2.1 Software

Source and binary releases: https://pypi.python.org/pypi/dynetx

Github (latest development): https://github.com/GiulioRossetti/dynetx

2.2 Documentation

CHAPTER 3

Installing

Before installing DyNetx, you need to have setuptools installed.

3.1 Note

In case of misaligned versions between pypl and GitHub, the documentation will refer to the GitHub version.

3.1.1 Quick install

Get DyNetx from the Python Package Index at pypl.

or install it with

```
pip install dynetx
```

and an attempt will be made to find and install an appropriate version that matches your operating system and Python version.

You can install the development version with

```
pip install git://github.com/GiulioRossetti/dynetx.git
```

3.1.2 Installing from source

You can install from source by downloading a source archive file (tar.gz or zip) or by checking out the source files from the GitHub source code repository.

DyNetx is a pure Python package; you don't need a compiler to build or install it.

Source archive file

Download the source (tar.gz or zip file) from pypl or get the latest development version from GitHub

Unpack and change directory to the source directory (it should have the files README.txt and setup.py).

Run python setup.py install to build and install

GitHub

Clone the DyNetx repostitory (see GitHub for options)

```
git clone https://github.com/GiulioRossetti/dynetx.git
```

Change directory to ndlib

Run python setup.py install to build and install

If you don't have permission to install software on your system, you can install into another directory using the –user, –prefix, or –home flags to setup.py.

For example

```
python setup.py install --prefix=/home/username/python
```

or

```
python setup.py install --home=~
```

or

```
python setup.py install --user
```

If you didn't install in the standard Python site-packages directory you will need to set your PYTHONPATH variable to the alternate location. See http://docs.python.org/2/install/index.html#search-path for further details.

3.1.3 Requirements

Python

To use DyNetx you need Python 2.7, 3.2 or later.

The easiest way to get Python and most optional packages is to install the Enthought Python distribution "Canopy" or using Anaconda.

There are several other distributions that contain the key packages you need for scientific computing.

Required packages

The following are packages required by `DyNetx`.

NetworkX

`DyNetx` extends the networkx python library adding dynamic network facilities.

Download: http://networkx.github.io/download.html

CHAPTER 4

DyNetx Tutorial

DyNetx is built upon networkx and is designed to configure, model and analyze dynamic networks.

In this tutorial we will introduce the `DynGraph` object that can be used to describe undirected, temporal graphs.

4.1 Creating a graph

Create an empty dynamic graph with no nodes and no edges.

```
import dynetx as dn
g = dn.DynGraph(edge_removal=True)
```

During the construction phase the `edge_removal` parameter allows to specify if the dynamic graph will allow edge removal or not.

4.1.1 Interactions

`G` can be grown by adding one interaction at a time. Every interaction is univocally defined by its endpoints, `u` and `v`, as well as its timestamp `t`.

```
g.add_interaction(u=1, v=2, t=0)
```

Moreover, also interaction duration can be specified at creation time, by setting kwarg `e` equal to the last timestamp at which the interaction is present:

```
g.add_interaction(u=1, v=2, t=0, e=3)
```

In the above example the interaction `(1, 2)` appear at time `0` and vanish at time `3`, thus being present in `[0, 2]`.

Interaction list can also be added: in such scenario all the interactions in the list will have a same timestamp (i.e. they will belong to a same network *snapshot*)

```
g.add_interactions_from([(1, 2), (2, 3), (3, 1)], t=2)
```

The same method can be used to add any ebunch of interaction. An *ebunch* is any iterable container of interaction-tuples.

```
g.add_interactions_from(H.edges(), t=2)
```

4.1.2 Nodes

Flattened node degree can be computed via the usual degree method exposed by networkx graph objects. In order to get the time dependent degree a parameter t, identifying the desired snapshot, must be specified.

Similarly, the neighbors method has been extended with a similar optional filtering parameter t.

4.2 Read graph from file

DyNetx allows to read/write networks from files in two formats:

- snapshot graph (extended edgelist)
- interaction graph (extended edgelist)

The former format describes the dynamic graph one edge per row as a 3-tuple

```
n1 n2 t1
```

where

- n1 and n2 are nodes
- t1 is the timestamp of interaction appearance

The latter format describes the dynamic graph one interaction per row as a 4-tuple

```
n1 n2 op t1
```

where

- n1 and n2 are nodes
- t1 is the timestamp of interaction appearance
- op identify either the insertion, +, or deletion, − of the edge

4.2.1 Snapshot Graph

In order to read a snapshot graph file

```
g = dn.read_snapshots(graph_filename, nodetype=int, timestamptype=int)
```

in order to save a graph in the same format

```
dn.write_snapshots(graph, graph_filename)
```

4.2.2 Interaction Graph

In order to read an interaction graph file

```
g = dn.read_interactions(graph_filename, nodetype=int, timestamptype=int)
```

in order to save a graph in the same format

```
dn.write_interactions(graph, graph_filename)
```

4.3 Snapshots and Interactions

The timestamps associated to graph edges can be retrieved through

```
g.temporal_snapshots_ids()
```

Similarly, the number of interactions in a given snapshot can be obtained via

```
g.number_of_interactions(t=snapshot_id)
```

if the parameter `t` is not specified a dictionary snapshot->edges number will be returned.

4.4 Slicing a Dynamic Network

Once loaded a graph it is possible to extract from it a time slice, i.e., a time-span graph

```
s = g.time_slice(t_from=2, t_to=3)
```

the resulting `DynGraph` stored in `s` will be composed by nodes and interactions existing within the time span `[2, 3]`.

4.5 Obtain the Interaction Stream

A dynamic network can be also described as stream of interactions, a chronologically ordered list of interactions

```
for e in g.stream_interactions():
        print e
```

the `stream_interactions` method returns a generator that streams the interactions in `g`, where `e` is a 4-tuple `(u, v, op, t)`

- `u, v` are nodes
- `op` is a edge creation or deletion event (respectively `+`, `-`)
- `t` is the interactions timestamp

CHAPTER 5

Reference

In this section are introduced the components that constitute `DyNetx`, namely

- The implemented dynamic graph models
- The implemented algorithms

In `DyNetx` are implemented the following **Dynamic Graph** models:

5.1 Graph types

DyNetx provides data structures and methods for storing graphs.

The choice of graph class depends on the structure of the graph you want to represent.

5.1.1 Which graph class should I use?

Dynamic Graph Type	DyNetx Class
Undirected	DynGraph
Directed	DynDiGraph

5.1.2 Basic graph types

Undirected Dynamic Graphs

Overview

Adding and removing nodes and edges

DynGraph.__init__
DynGraph.add_interaction
DynGraph.add_interactions_from
DynGraph.add_star
DynGraph.add_path
DynGraph.add_cycle

Iterating over nodes and edges

DynGraph.interactions
DynGraph.interactions_iter
DynGraph.neighbors
DynGraph.neighbors_iter
DynGraph.nodes
DynGraph.nodes_iter

Information about graph structure

DynGraph.has_interaction
DynGraph.number_of_interactions
DynGraph.degree
DynGraph.degree_iter
DynGraph.size
DynGraph.order
DynGraph.has_node
DynGraph.number_of_nodes
DynGraph.to_directed
DynGraph.update_node_attr
DynGraph.update_node_attr_from

Dynamic Representation: Access Snapshots and Interactions

DynGraph.stream_interactions
DynGraph.time_slice
DynGraph.temporal_snapshots_ids
DynGraph.interactions_per_snapshots
DynGraph.inter_event_time_distribution

Directed Dynamic Graphs

Overview

Adding and removing nodes and edges

DynDiGraph.__init__

Continued on next page

Table 5 – continued from previous page

DynDiGraph.add_interaction
DynDiGraph.add_interactions_from

Iterating over nodes and edges

DynDiGraph.interactions
DynDiGraph.interactions_iter
DynDiGraph.in_interactions
DynDiGraph.in_interactions_iter
DynDiGraph.out_interactions
DynDiGraph.out_interactions_iter
DynDiGraph.neighbors
DynDiGraph.neighbors_iter
DynDiGraph.successors
DynDiGraph.successors_iter
DynDiGraph.predecessors
DynDiGraph.predecessors_iter
DynDiGraph.nodes
DynDiGraph.nodes_iter

Information about graph structure

DynDiGraph.has_interaction
DynDiGraph.has_successor
DynDiGraph.has_predecessor
DynDiGraph.number_of_interactions
DynDiGraph.degree
DynDiGraph.degree_iter
DynDiGraph.in_degree
DynDiGraph.in_degree_iter
DynDiGraph.out_degree
DynDiGraph.out_degree_iter
DynDiGraph.size
DynDiGraph.order
DynDiGraph.has_node
DynDiGraph.number_of_nodes
DynDiGraph.to_undirected
DynDiGraph.update_node_attr
DynDiGraph.update_node_attr_from

Dynamic Representation: Access Snapshots and Interactions

DynDiGraph.stream_interactions
DynDiGraph.time_slice
DynDiGraph.temporal_snapshots_ids
DynDiGraph.interactions_per_snapshots
DynDiGraph.inter_event_time_distribution

Continued on next page

Table 8 – continued from previous page

DynDiGraph.inter_in_event_time_distribution
DynDiGraph.inter_out_event_time_distribution

5.2 Algorithms

Dynetx implements standard temporal network measures

5.2.1 Paths

Compute the time respecting paths between nodes in the graph.

These algorithms work with undirected and directed graphs.

Time respecting paths

time_respecting_paths
all_time_respecting_paths
annotate_paths
path_duration
path_length

Temporal Directed Acyclic Graph

temporal_dag

5.3 Functions

Functional interface to graph methods and assorted utilities.

5.3.1 Graph

degree
degree_histogram
density
create_empty_copy
is_directed
add_star
add_path
add_cycle

5.3.2 Nodes

nodes

Continued on next page

Table 12 – continued from previous page

number_of_nodes
all_neighbors
non_neighbors

5.3.3 Interactions

interactions
number_of_interactions
non_interactions

5.3.4 Freezing graph structure

freeze
is_frozen

5.3.5 Snapshots and Interaction Stream

stream_interactions
time_slice
temporal_snapshots_ids
interactions_per_snapshots
inter_event_time_distribution

5.4 Reading and writing graphs

5.4.1 Edge List

Interaction Graph

write_interactions
read_interactions

Snapshot Graphs

write_snapshots
read_snapshots

5.4.2 JSON

JSON data

Generate and parse JSON serializable data for DyNetx graphs.

`node_link_data`
`node_link_graph`