P

Python Packages for Networks

Vladimir Batagelj
Department of Theoretical Computer Science, Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia
University of Primorska, Andrej Marušič Institute, Koper, Slovenia

Synonyms

DeepGraph; graph-tool; igraph; Library; Network description; NetworKit; NetworkX; Snap.py; Tulip; Zen

Glossary

CSV	Comma-separated values is a plain text format for storing tabular data.
JSON	JavaScript Object Notation is a data-interchange plain text format.
Large network	A network with several thousands or millions of nodes.
Network analysis	A study of networks as representations of relations between discrete objects.

Definition

Python is a high-level general-purpose programming language and easy to understand and learn. To deal with networks, different representations of networks in Python were proposed. On their basis, several Python libraries were developed to support programming of network analysis tasks.

Introduction

Python is an open-source, interpreted, interactive, object-oriented programming language (Python Software Foundation, https://www.python.org/). Its name comes from the BBC TV show Monty Python's Flying Circus. It runs on all main platforms: Windows, MAC, Linux/Unix, and Android. It is a successor of the programming language ABC that was based on ideas of structured programming (Dahl et al. 1972). Python was initially, starting in December 1989, developed by Guido van Rossum at CWI. It reached Version 1.0 in January 1994. In 2001, the Python Software Foundation (PSF) was formed. Guido remains Python's principal author – a Benevolent Dictator for Life. Python 2.0 was released on October 16, 2000, and Python 3.0 on December 3, 2008.

Python is based on the interpretation of programs that slows down their execution with respect to compiled programs. In most cases, the critical tasks are programmed in compiled languages and made available as packages (libraries). Besides standard packages, Python has a broad collection of packages for different tasks – see PyPI (the Python Package Index (https://pypi.python.org/pypi); in July 2017, it

R. Alhajj, J. Rokne (eds.), *Encyclopedia of Social Network Analysis and Mining*,
https://doi.org/10.1007/978-1-4614-7163-9_110210-1

contained 113,701 packages). Python's emphasis is on faster program development and its readability. It is also easy to learn – it is a kind of modern Basic. Besides R, Python is the main programming language used in data analysis.

A support for graphs in Python is an old problem – see a page "A Python Graph API?" (https://wiki.python.org/moin/PythonGraphApi). It starts with a question what is a graph? There are two approaches to this question: particular and general. Fathers of graph theory choose the particular approach: Berge (1958) relational/directed graphs and Harary (1969) simple undirected graphs without loops. We can find a general approach in Zykov (1969). In this entry, we shall use a general approach. In a graph, we allow both edges and arcs. A pair of nodes can be linked by multiple links. Loops are also allowed.

In most applications of graphs, we have to consider additional information about nodes or links – we are essentially dealing with networks.

Networks

A *network* is based on two sets – a set of *nodes* (vertices), which represent the selected units, and a set of *links* (lines), which represent ties between units. They determine a *graph*. A link can be *directed*, an *arc*, or *undirected*, an edge. Additional data about nodes or links may be known – their *properties* (attributes), for example, name/label, type, age, value, etc.

$$\text{Network} = \text{Graph} + \text{Data}$$

Formally, a *network* $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W})$ consists of:

- A *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{A}$ is the set of arcs, $\mathcal{E}$ is the set of edges, and $\mathcal{L} = \mathcal{E} \cup \mathcal{A}$ is the set of links. $n = \mid \mathcal{V} \mid, m = \mid \mathcal{L} \mid$
- $\mathcal{P}$ *vertex value functions* or properties: $p : \mathcal{V} \rightarrow A$
- $\mathcal{W}$ *link value functions* or weights: $w : \mathcal{L} \rightarrow B$

Types of Networks

In a *two-mode* network $\mathcal{N} = ((\mathcal{V}_1, \mathcal{V}_2), \mathcal{L}, \mathcal{P}, \mathcal{W})$, its set of nodes is split into two subsets. Each link has its end-nodes in both sets.

In a *multirelational* network $\mathcal{N} = (\mathcal{V}, (\mathcal{L}_i, i \in I), \mathcal{P}, \mathcal{W})$, the set of its links is partitioned into several mutually disjoint subsets – relations. Such networks are often obtained from text decomposed into simple sentences of the form (Subject Verb Object). Subjects and Objects are represented with nodes, and Verbs determine relations.

In a *temporal* network $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{T}, \mathcal{P}, \mathcal{W})$, the time $\mathcal{T}$ is added. To each node and to each link, its *activity* set is assigned. Also properties and weights can change through time – temporal quantities (Batagelj and Praprotnik 2016).

A *collection* of networks consists of some (one-mode and two-mode) networks with common subsets of nodes.

Types of networks can be combined – for example, a temporal two-mode multirelational network.

Description of Networks

How to describe a network $\mathcal{N}$? In principle the answer is simple – we list its sets $\mathcal{V}, \mathcal{L}, \mathcal{P}$, and $\mathcal{W}$. The simplest way is to describe a network $\mathcal{N}$ by providing $(\mathcal{V}, \mathcal{P})$ and $(\mathcal{L}, \mathcal{W})$ in a form of two tables. Both tables are often maintained in Excel. They can be exported as a text in CSV (comma-separated values) format.

As an example, let us describe a part of bibliographic network determined by the following works: Generalized blockmodeling (Doreian et al. 2005), Clustering with relational constraint (Ferligoj and Batagelj 1982), Partitioning signed social networks (Doreian and Mrvar 2009), and The Strength of Weak Ties (Granoveter 1973).

There are nodes of different modes (types), persons, papers, books, series, journals, and publishers, and different relations among them, author_of, editor_of, contained_in, cites, and published_by. The corresponding tables $(\mathcal{V}, \mathcal{P})$ and $(\mathcal{L}, \mathcal{W})$ are presented in Figs. 1 and 2.

```
name;mode;country;sex;year;vol;num;fPage;lPage;x;y
"Batagelj, Vladimir";person;SI;m;;;;;;809.1;653.7
"Doreian, Patrick";person;US;m;;;;;;358.5;679.1
"Ferligoj, Anuška";person;SI;f;;;;;;619.5;680.7
"Granovetter, Mark";person;US;m;;;;;;145.6;660.5
"Moustaki, Irini";person;UK;f;;;;;;783.0;228.0
"Mrvar, Andrej";person;SI;m;;;;;;478.0;630.1
"Clustering with relational constraint";paper;;;1982;47;;413;426;684.1;380.1
"The Strength of Weak Ties";paper;;;1973;78;6;1360;1380;111.3;329.4
"Partitioning signed social networks";paper;;;2009;31;1;1;11;408.0;337.8
"Generalized Blockmodeling";book;;;2005;24;;1;385;533.0;445.9
"Psychometrika";journal;;;;;;;;741.8;086.1
"Social Networks";journal;;;;;;;;321.4;236.5
"The American Journal of Sociology";journal;;;;;;;;111.3;168.9
"Structural Analysis in the Social Sciences";series;;;;;;;;310.4;082.8
"Cambridge University Press";publisher;UK;;;;;;;534.3;238.2
"Springer";publisher;US;;;;;;;884.6;174.0
```

Python Packages for Networks, Fig. 1 bibNodes.csv

```
from;relation;to
"Batagelj, Vladimir";authorOf;"Generalized Blockmodeling"
"Doreian, Patrick";authorOf;"Generalized Blockmodeling"
"Ferligoj, Anuška";authorOf;"Generalized Blockmodeling"
"Batagelj, Vladimir";authorOf;"Clustering with relational constraint"
"Ferligoj, Anuška";authorOf;"Clustering with relational constraint"
"Granovetter, Mark";authorOf;"The Strength of Weak Ties"
"Granovetter, Mark";editorOf;"Structural Analysis in the Social Sciences"
"Doreian, Patrick";authorOf;"Partitioning signed social networks"
"Mrvar, Andrej";authorOf;"Partitioning signed social networks"
"Moustaki, Irini";editorOf;"Psychometrika"
"Doreian, Patrick";editorOf;"Social Networks"
"Generalized Blockmodeling";containedIn;"Structural Analysis in the Social Sciences"
"Clustering with relational constraint";containedIn;"Psychometrika"
"The Strength of Weak Ties";containedIn;"The American Journal of Sociology"
"Partitioning signed social networks";containedIn;"Social Networks"
"Partitioning signed social networks";cites;"Generalized Blockmodeling"
"Generalized Blockmodeling";cites;"Clustering with relational constraint"
"Structural Analysis in the Social Sciences";publishedBy;"Cambridge University Press"
"Psychometrika";publishedBy;"Springer"
```

Python Packages for Networks, Fig. 2 bibLinks.csv

In large networks, to avoid the empty cells, we split a network to some subnetworks – we make a collection of networks.

Factorization and Description of Large Networks

To save space and improve the computing efficiency, we often replace values of categorical variables with integers. In R this encoding is called a *factorization*. We enumerate all possible values of a given categorical variable (coding table) and afterwards replace each its value by the corresponding index in the coding table. This approach is used in most programs dealing with large networks. Unfortunately the coding table is often a kind of meta-data.

In Fig. 3 (in two columns), a Pajek's NET file **bib.net** corresponding to our example network is presented (de Nooy et al. 2012). Pajek's data format is based on factorization. The items in link descriptions are a relation number, from-node, to-node, weight, and label.

In Fig. 4 the mode (first column) and the sex (second column) partitions of nodes are given. The mode coding table is 1 = book, 2 = journal, 3 = paper, 4 = person, 5 = publisher, and 6 = series; and the sex coding table is 0 = not applicable, 1 = female, and 2 = male.

In Fig. 5 a visualization, produced with Pajek, of our example network is presented.

```
*vertices  16
1 "Batagelj, Vladimir"
2 "Doreian, Patrick"
3 "Ferligoj, Anuška"
4 "Granovetter, Mark"
5 "Moustaki, Irini"
6 "Mrvar, Andrej"
7 "Clustering with relational constraint"
8 "The Strength of Weak Ties"
9 "Partitioning signed social networks"
10 "Generalized Blockmodeling"
11 "Psychometrika"
12 "Social Networks"
13 "The American Journal of Sociology"
14 "Structural Analysis in the Social Sciences"
15 "Cambridge University Press"
16 "Springer"
*arcs :1 "authorOf"
*arcs :2 "cites"
*arcs :3 "containedIn"
*arcs :4 "editorOf"
*arcs :5 "publishedBy"
```

```
*arcs
1: 1 10 1 l "authorOf"
1: 2 10 1 l "authorOf"
1: 3 10 1 l "authorOf"
1: 1 7 1 l "authorOf"
1: 3 7 1 l "authorOf"
1: 4 8 1 l "authorOf"
4: 4 14 1 l "editorOf"
1: 2 9 1 l "authorOf"
1: 6 9 1 l "authorOf"
4: 5 11 1 l "editorOf"
4: 2 12 1 l "editorOf"
3: 10 14 1 l "containedIn"
3: 7 11 1 l "containedIn"
3: 8 13 1 l "containedIn"
3: 9 12 1 l "containedIn"
2: 9 10 1 l "cites"
2: 10 7 1 l "cites"
5: 14 15 1 l "publishedBy"
5: 11 16 1 l "publishedBy"
```

Python Packages for Networks, Fig. 3 bib.net

```
*vertices  16
4
4
4
4
4
4
3
3
3
1
2
2
2
6
5
5
```

```
*vertices  16
2
2
1
2
1
2
0
0
0
0
0
0
0
0
0
0
```

Python Packages for Networks, Fig. 4 bibMode.clu and bibSex.clu

netJSON

JSON (JavaScript Object Notation) is becoming very popular for describing and exchanging structured objects among programs and web applications. It is human-readable and preserves the structure of complex data objects. In all main programming languages, efficient libraries exist for processing JSON files.

We developed **netJSON** – a JSON-based format for description of networks. In Fig. 6 a description of our example network in **netJSON** is presented.

In general ids in a **netJSON** description can be any immutable data objects such that each node/link gets different id.

In the description in Fig. 6, we could omit the ids. In such a case, they are determined implicitly as index of the item position in a list. Since the value of info.org is 1, the counting starts with 1.

JSON allows that the field values are structured objects. Therefore, for example, we can use temporal quantities as property values or weights

```
{ "id": "Borgatti_S", "nPapers":
[[1991, 1992, 1], [1994, 1995, 1],
[1997, 1998, 1], [1999, 2000, 2],
[2003, 2004, 1], [2005, 2007, 2],
[2007, 2008, 1]]] }
```

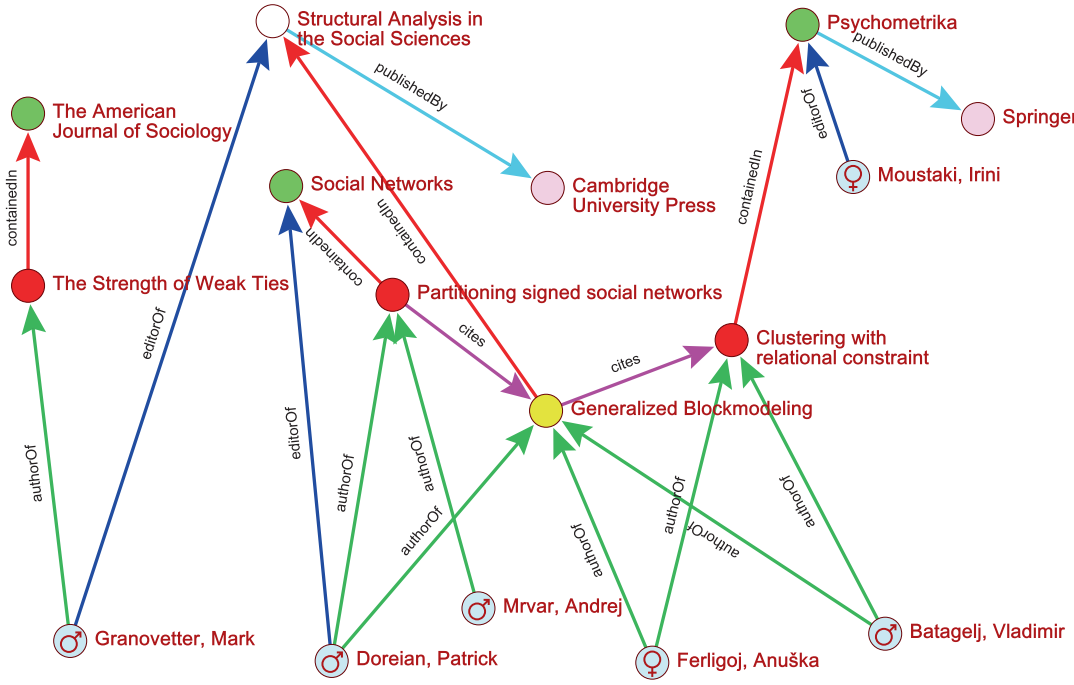
or specify a function that transforms a property value or a weight.

A general netJSON format that will support a description of collections of (linked) networks is still under development.

Representations of Networks in Python

A representation of a network in Python depends on network size, variability (static/dynamic), and intended operations on it. Selecting the right representation can improve the efficiency of network processing. In libraries a representation that leads to efficient algorithms for most of the tasks is usually selected.

A "classic" graph implementation as a "dictionary of list" was proposed by Guido van Rossum (1998).



Python Packages for Networks, Fig. 5 Example bibliographic network – Pajek's picture

For example, a graph $\mathcal{V} = \{A, B, C, D, E, F\}$ and $\mathcal{A} = \{(A, B), (A, C), (B, C), (B, D), (C, D), (D, C), (E, F), (F, C)\}$ is represented as

```
graph = {'A': ['B', 'C'],
         'B': ['C', 'D'],
         'C': ['D'],
         'D': ['C'],
         'E': ['F'],
         'F': ['C'] }
```

Even better is a "dictionary of dictionary" representation that is a basis of **PADS** collection of Python algorithms and data structures implemented by David Eppstein of the University of California, Irvine (http://www.ics.uci.edu/~eppstein/PADS/). Another interesting approach to network representation is proposed in a library **graphABC** (http://www.linux.it/~della/GraphABC/). See also the book by Hetland (2010).

For implementing in Python prototype network analysis algorithms and programs, we developed in 2009 a Python library **Nets** that supports basic operations with networks based on an elaborated "dictionary of dictionary" representation (https://github.com/bavla/Nets). In **Nets** each node/link has its *id*. If a link id is not specified by a user, it is determined by **Nets**.

The library **Nets** is based on a network object containing three dictionaries:

- `_info` – keys are general properties of a network. System properties: `network`, `title`, `simple`, `directed`, `multirel`, `mode`, `temporal`, `meta`, `nNodes`, `nArcs`, `nEdges`, `time`, etc. User properties such as `nWeak`, `planar`, etc. can be also included.
- `_nodes` – keys are ids of nodes. A value is a list of four dictionaries: edgeStar, inArcStar, outArcStar, and nodeProperties. Each star is again a dictionary that has for keys ids of neighboring nodes and for values lists of link ids.
- `_links` – keys are ids of links. A value is a list [nodeId1, nodeId2, directed, relId, linkProperties] where linkProperties is a dictionary of weights.

```
{
  "netJSON": "basic",
  "info": {
    "org":1, "nNodes":16, "nArcs":19, "nEdges":0,
    "simple":True, "directed":True, "multirel":True, "mode":6, "temporal":False,
    "network":"bib", "title":"Example bibliographic network",
    "meta":[{ "au":"Vladimir Batagelj", "ti":"Network creation", "date":"June 10, 2016" }]
  },
  "nodes": [
    { "id":1, "lab":"Batagelj, Vladimir", "mode":"person", "country":"SI", "sex":"male" },
    { "id":2, "lab":"Doreian, Patrick", "mode":"person", "country":"US", "sex":"male" },
    { "id":3, "lab":"Ferligoj, Anuška", "mode":"person", "country":"SI", "sex":"female" },
    { "id":4, "lab":"Granovetter, Mark", "mode":"person", "country":"US", "sex":"male" },
    { "id":5, "lab":"Moustaki, Irini", "mode":"person", "country":"UK", "sex":"female" },
    { "id":6, "lab":"Mrvar, Andrej", "mode":"person", "country":"SI", "sex":"male" },
    { "id":7, "lab":"Clustering with relational constraint", "mode":"paper", "year":1982,
              "vol":47, "fPage":413, "lPage":426 },
    { "id":8, "lab":"The Strength of Weak Ties", "mode":"paper", "year":1973, "vol":78,
              "num":6, "fPage":1360, "lPage":1380 },
    { "id":9, "lab":"Partitioning signed social networks", "mode":"paper", "year":2009,
              "vol":31, "num":1, "fPage":1, "lPage":11 },
    { "id":10, "lab":"Generalized Blockmodeling", "mode":"book", "year":2005, "vol":24,
               "fPage":1, "lPage":385 },
    { "id":11, "lab":"Psychometrika", "mode":"journal" },
    { "id":12, "lab":"Social Networks", "mode":"journal" },
    { "id":13, "lab":"The American Journal of Sociology", "mode":"journal" },
    { "id":14, "lab":"Structural Analysis in the Social Sciences", "mode":"series" },
    { "id":15, "lab":"Cambridge University Press", "mode":"publisher", "country":"UK" },
    { "id":16, "lab":"Springer", "mode":"publisher", "country":"US" }
  ]
  "links": [
    { "id":1, "type":"arc", "n1":1, "n2":10, "rel":"authorOf" },
    { "id":2, "type":"arc", "n1":2, "n2":10, "rel":"authorOf" },
    { "id":3, "type":"arc", "n1":3, "n2":10, "rel":"authorOf" },
    { "id":4, "type":"arc", "n1":1, "n2":7, "rel":"authorOf" },
    { "id":5, "type":"arc", "n1":3, "n2":7, "rel":"authorOf" },
    { "id":6, "type":"arc", "n1":4, "n2":8, "rel":"authorOf" },
    { "id":7, "type":"arc", "n1":4, "n2":14, "rel":"editorOf" },
    { "id":8, "type":"arc", "n1":2, "n2":9, "rel":"authorOf" },
    { "id":9, "type":"arc", "n1":6, "n2":9, "rel":"authorOf" },
    { "id":10, "type":"arc", "n1":5, "n2":11, "rel":"editorOf" },
    { "id":11, "type":"arc", "n1":2, "n2":12, "rel":"editorOf" },
    { "id":12, "type":"arc", "n1":10, "n2":14, "rel":"containedIn" },
    { "id":13, "type":"arc", "n1":7, "n2":11, "rel":"containedIn" },
    { "id":14, "type":"arc", "n1":8, "n2":13, "rel":"containedIn" },
    { "id":15, "type":"arc", "n1":9, "n2":12, "rel":"containedIn" },
    { "id":16, "type":"arc", "n1":9, "n2":10, "rel":"cites" },
    { "id":17, "type":"arc", "n1":10, "n2":7, "rel":"cites" },
    { "id":18, "type":"arc", "n1":14, "n2":15, "rel":"publishedBy" },
    { "id":19, "type":"arc", "n1":11, "n2":16, "rel":"publishedBy" }
  ]
}
```


Python Packages for Networks, Fig. 6 bib.JSON

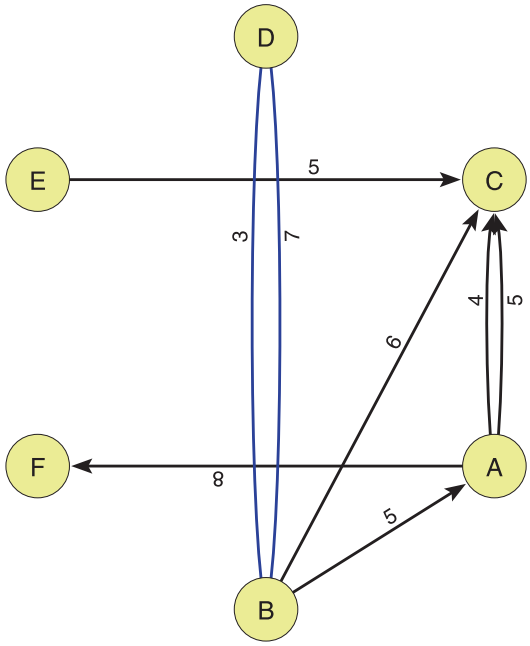
For an example network from Fig. 7, the corresponding **Nets** representation is presented in Fig. 8.

Python Packages

There exist several Python packages that support network analysis. They can be classified in two main classes:

- Pure-Python packages: easy to install, platform independent, less efficient, and in reasonable time can deal with networks with up to some millions of nodes
- Compiled libraries with Python interface: sometimes difficult to install, very efficient, and can deal with very large networks

A support for network analysis is available also in the **SageMath** (http://www.sagemath.org/;



Python Packages for Networks, Fig. 7 Example 2 network

Joyner et al. 2013) – a Python-based-free open-source mathematics software system licensed under the GPL.

NetworkX

NetworkX (https://networkx.github.io/) is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It is based on "dictionary of dictionary" data structure and is written in pure Python. It was created by Aric Hagberg, Dan Schult, and Pieter Swart in 2002 and 2003 and released in April 2005. Its background was described in a paper (Hagberg et al. 2008).

NetworkX is the most popular among Python packages for network analysis. It contains many network analysis algorithms and is very well documented. Some books (Caldarelli and Chessa 2016; Al-Taie and Kadry 2017; Fouss et al. 2016; Tsvetovat and Kouznetsov 2011) are based on it. It is the de facto standard for the analysis in Python of small- to medium-size (up to some millions of nodes) networks.

NetworkX is a cross-platform (Linux/Unix, Mac OS X, Windows) package and runs with Python 2.7/3.4 or later.

DeepGraph

The package **DeepGraph** was developed by Dominik Traxl and made public in 2016 (https://github.com/deepgraph/deepgraph/). He describes it as follows: **DeepGraph** is a scalable, general-purpose data analysis package. It implements a network representation based on **Pandas** DataFrames – the nodes and edges are each represented by a DataFrame. It provides methods to construct, partition, and plot graphs, to interface with popular network packages, and more. Since it provides interfacing methods to **NetworkX**, **scipy** sparse matrices, and **graph-tool**, a user can easily exploit the different advantages of these packages. Its theoretical background was published in Traxl et al. (2016).

DeepGraph is a cross-platform (Linux/Unix, Mac OS X, Windows) package and runs with Python 2.7/3.4 or later.

Zen

Zen (https://github.com/networkdynamics/zenlib, http://zen.networkdynamics.org/), developed in 2012 by Derek Ruths, is a library that provides a high-speed, easy-to-use API for loading, analyzing, visualizing, and manipulating networks in Python. By using a hybrid of Python and Cython code, it combines the speed and low memory overhead of C with the ease of use of Python. The result is a library that truly makes working with networks easy and fast. Many operations in **Zen** are over 100 times faster than the identical operation in **NetworkX**.

Zen requires Python 2.7 or later (but not Python 3). In principle it should be a cross-platform, but not easy to install.

```
>>> N._info
{'simple': False, 'mode': 1, 'multirel': False, 'temporal': False,
 'network': 'test', 'title': 'Test'}
>>> N._nodes
{'B': [{'D': [1, 6]}, {}, {'A': [2], 'C': [4]}, {'x': 0.5, 'y': 0.95}],
 'A': [{}, {'B': [2]}, {'C': [3, 8], 'F': [7]}, {'x': 0.89, 'y': 0.725}],
 'C': [{}, {'A': [3, 8], 'B': [4], 'E': [5]}, {}, {'x': 0.89, 'y': 0.275}],
 'D': [{'B': [1, 6]}, {}, {}, {'x': 0.5, 'y': 0.05}],
 'E': [{}, {}, {'C': [5]}, {'x': 0.11, 'y': 0.275}],
 'F': [{}, {'A': [7]}, {}, {'x': 0.11, 'y': 0.725}]}
>>> N._links
{1: ['B', 'D', False, None, {'w': 3}],
 2: ['B', 'A', True, None, {'w': 5}],
 3: ['A', 'C', True, None, {'w': 4}],
 4: ['B', 'C', True, None, {'w': 6}],
 5: ['E', 'C', True, None, {'w': 5}],
 6: ['B', 'D', False, None, {'w': 7}],
 7: ['A', 'F', True, None, {'w': 8}],
 8: ['A', 'C', True, None, {'w': 5}]}
>>>
```

Python Packages for Networks, Fig. 8 Example 2 network representation with library **Nets**

igraph

igraph (http://igraph.org) is a network analysis library written in C and designed for extremely large networks. It is a collection of network analysis tools with the emphasis on efficiency, portability, and ease of use. It can be programmed in R, Python, and C/C++. Very popular is the **igraph/R** package. **igraph** was developed by Gábor Csárdi and Tamás Nepusz (2006) and first released in 2006.

igraph/Python is a cross-platform (Linux/Unix, Mac OS X, Windows) package and runs with Python 2.7/3.4 or later. Installing **igraph/Python** is relatively simple (Gohlke 2011). Basic documentation is provided.

graph-tool

graph-tool (https://graph-tool.skewed.de/) is an efficient Python module for manipulation and statistical analysis of networks. The core data structures and algorithms are implemented in C++, based heavily on the **Boost Graph Library** (BGL, http://www.boost.org/libs/graph/doc/index.html), and can be used to work with very large networks. Many algorithms are implemented in parallel using OpenMP, which provides excellent performance on multi-core architectures. It was developed by Tiago de Paula Peixoto and released in 2006.

graph-tool package runs on Linux/Unix and Mac OS X with Python 2.7/3.4 or later. **graph-tool** is quite complicated to install.

NetworKit

NetworKit (https://networkit.iti.kit.edu/) is an open-source toolkit for large-scale (up to billions of links) network analysis. It is a Python package, with performance-critical algorithms implemented in C++/OpenMP. **NetworKit** is maintained by the Research Group Parallel Computing of the Institute of Theoretical Informatics at Karlsruhe Institute of Technology (KIT). It started as a collection of community detection algorithms developed in C++ by Henning Meyerhenke and Christian L. Staudt. It was first released in March 2013. Its background is described in the paper (Staudt et al. 2016).

NetworKit is comparable to packages such as **NetworkX**, albeit with a focus on parallelism and scalability. It is a hybrid combining the performance of kernels written in C++ with a convenient Python frontend.

NetworKit is a cross-platform (Linux/Unix, Mac OS X, Windows) package and runs with Python 3.3 or later. Some installing problems were reported. Basic documentation is provided.

Snap.py

Stanford Network Analysis Platform (**SNAP**) (Leskovec and Sosič 2016) is a general-purpose, high-performance system for analysis and manipulation of large networks. It is written in C++ and optimized for maximum performance and compact graph representation. It easily scales to massive networks with hundreds of millions of nodes and billions of edges. **SNAP** was originally developed by Jure Leskovec in the course of his PhD studies. The first release was made available in November 2009. **Snap.py** (https://snap.stanford.edu/snappy/) is a Python interface for **SNAP**. It provides performance benefits of **SNAP**, combined with flexibility of Python.

Installation packages for Mac OS X, Linux (as CentOS), and Windows 64-bit are available. **Snap.py** sticks to Python 2.7 or later (but not Python 3). Basic documentation is provided.

Tulip Python

Tulip was originally developed in 2001 by David Auber at LaBRI, University of Bordeaux. At its web site (http://tulip.labri.fr/Documentation/current/tulip-python/html/), we find the following description: **Tulip** is an information visualization framework written in C++ dedicated to the analysis and visualization of graphs. **Tulip Python** is a set of modules that exposes to Python almost all the content of the **Tulip C ++** API. The main features provided by the bindings are creation and manipulation of graphs, storage of data on graph elements (float, integer, Boolean, color, size, coordinate, list, etc.), application of algorithms of different types on graphs (layout, metric, clustering, etc.), and the ability to write **Tulip** plug-ins in pure Python. The bindings can be used inside the **Tulip** software GUI in order to run scripts on the current visualized graph. Starting from **Tulip** 3.6, the bindings can also be used outside **Tulip** through the classical Python interpreter.

For details about **Tulip**, see the essay Tulip 5.

Tulip Python is a cross-platform (Linux/Unix, Mac OS X, Windows) package and runs with Python 3.3 or later. Basic documentation is provided.

Cross-References

- ▶ Network Data File Formats
- ▶ Pajek and PajekXXL
- ▶ R Packages for Social Network Analysis
- ▶ Tulip 5
- ▶ Sources of Network Data

Acknowledgments The work was partially supported by Slovenian Research Agency (ARRS) projects J7-8279 and J1-6720 and grant P1-0294.

References

Al-Taie MZ, Kadry S (2017) Python for graph and network analysis. Springer, Cham

Batagelj V, Praprotnik S (2016) An algebraic approach to temporal network analysis based on temporal quantities. Soc Netw Anal Min 6(1):1–22

Berge C (1958) Théorie des graphes et ses applications. Dunod, Paris. The theory of graphs. Courier Co., 1962

Caldarelli G, Chessa A (2016) Data science and complex networks: real cases studies with Python. Oxford University Press, Oxford

Csárdi G, Nepusz T (2006) The igraph software package for complex network research. InterJournal Complex Systems, 1695

Dahl OJ, Dijkstra EW, Hoare CAR (1972) Structured programming. Academic, London

de Nooy W, Mrvar A, Batagelj V (2012) Exploratory network analysis using Pajek. Cambridge University Press, Cambridge

Doreian P, Mrvar A (2009) Partitioning signed social networks. Soc Networks 31(1):1–11

Doreian P, Batagelj V, Ferligoj A (2005) Generalized blockmodeling. Cambridge University Press, New York

Ferligoj A, Batagelj V (1982) Clustering with relational constraint. Psychometrika 47(4):413–426

Fouss F, Saerens M, Shimbo M (2016) Algorithms and models for network data and link analysis. Cambridge University Press, Cambridge, UK

Gohlke C (2011) Unofficial windows binaries for Python extension packages. http://www.lfd.uci.edu/~gohlke/pythonlibs/

Granoveter M (1973) The strength of weak ties. Am J Sociol 78(6):1360–1380

Hagberg A, Schult D, Swart P (2008) Exploring network structure, dynamics, and function using NetworkX. In: Varoquaux G, Vaught T, Millman J (eds) Proceedings

of the 7th Python in science conference (SciPy 2008), pp 11–15
Harary F (1969) Graph theory. Addison-Wesley, Reading
Hetland ML (2010) Python Algorithms: mastering basic algorithms in the Python Language. Apress, New York
Joyner D, Nguyen MV, Phillips D (2013) Algorithmic graph theory and Sage. https://code.google.com/archive/p/graphbook/
Leskovec J, Sosič R (2016) SNAP: a general purpose network analysis and graph mining library. ACM Trans Intell Syst Technol 8(1): 1 https://dl.acm.org/citation.cfm?id=2898361
Staudt C, Sazonovs A, Meyerhenke H (2016) NetworKit: a tool suite for large-scale complex network analysis. Netw Sci 4(4):508–530
Traxl D, Boers N, Kurths J (2016) Deep graphs – a general framework to represent and analyze heterogeneous complex systems across scales. Chaos 26(6):065303
Tsvetovat M, Kouznetsov A (2011) Social network analysis for startups: finding connections on the social web. O'Reilly, Sebastopol
van Rossum G (1998) Python patterns – implementing graphs. https://www.python.org/doc/essays/graphs/
Zykov AA (1969) Teorija konechnyh grafov I. Nauka, Novosibirsk