NetworKit User Guide

This text is meant as an introduction to using NetworKit for network analysis. We assume that you have read the Readme and successfully built the core library and the Python module.

Startup

Start a Python 3 shell. The standard shell works fine, although IPython may be what you really want for interactive programming and data analysis. These example sessions were created with IPython.

NetworKit can be imported like a standard Python module:

```
In[1]: import NetworKit
```

or

```
In[2]: from NetworKit import *
```

In the following, we will assume the latter way for brevity.

Reading Graphs from Disk

There is a large variety of formats for storing graph data in files. For NetworKit, the currently best supported format is the METIS adjacency format. Various example graphs in this format can be found here. We download the file PGPgiantcompo.graph to disk. The needed reader class named `METISGraphReader` is in the `graphio` submodule:

```
In[3]: G = graphio.METISGraphReader().read("PGPgiantcompo.graph")
    [BEGIN] reading graph G(n=10680, m=24316) from METIS file: 10% 20.0094%
     30.0187% 40.0281% 50.0375% 60.0468% 70.0562% 80.0655% 90.0749% [DONE]
```

More reader classes can be found in the `graphio` module. However, there is also a convenient function in the top namespace which tries to guess the input format and select the appropriate reader:

```
In[4]: G = readGraph("PGPgiantcompo.graph")
    [BEGIN] reading graph G(n=10680, m=24316) from METIS file: 10% 20.0094%
     30.0187% 40.0281% 50.0375% 60.0468% 70.0562% 80.0655% 90.0749% [DONE]
```

The Graph Object

The variable `G` now contains the network represented as an undirected graph.

```
In [10]: G
Out[10]: <_NetworKit.Graph at 0x7f71c9e519c0>
```

For NetworKit, nodes are just integer indexes and edges are tuples of nodes. `G` already provides a number of methods:

```
In [11]: G.
G.addEdge             G.edges               G.hasEdge
G.markAsWeighted      G.numberOfEdges       G.removeEdge          G.toString
G.addNode             G.getName             G.isMarkedAsWeighted
G.nodes               G.numberOfNodes       G.removeNode          G.weight
```

For example, we could check if nodes 42 and 43 are connected:

```
In [20]: G.hasEdge(42,43)
Out[20]: False
```

To work with the raw list of nodes or edges, call `G.nodes()` and `G.edges()` respectively.

Using Algorithms

Connected Components

As a first example of a network analysis kernel, we determine the connected components of the graph by issuing the following commands:

```
In [22]: cc = properties.ConnectedComponents()

In [23]: cc.run(G)
```

The `cc` object has now performed the calculation and results can be requested through different method calls.

```
In [24]: cc.numberOfComponents()
Out[24]: 1

In [25]: cc.sizeOfComponent(0)
Out[25]: 10680
```

Not surprisingly, `PGPgiantcompo` has a single connected component containing all nodes.

Community Detection

NetworKit contains several algorithms for parallel community detection, contained in the `community` submodule. The `PLM` class implements a parallel heuristic known as the Louvain method. Let's apply it to 'G

```
In [26]: zeta = community.PLM().run(G)
```

Computation will not take long for a relatively small network like `PGPgiantcompo`. The resulting partition is stored in the variable `zeta`, which internally stores a mapping from node index to community index. It also provides useful methods:

```
In [27]: zeta.numberOfClusters()
Out[27]: 90
```

We see that 90 communities have been found. Let's examine their sizes:

```
In [30]: zeta.clusterSizes()
Out[30]: [215, 355, 171, 563, 385, 235, 275, 137, 17, 691,
    360, 20, 482, 295, 99, 353, 121, 140, 262, 185, 212, 40, 368,
    82, 85, 78, 356, 16, 364, 772, 271, 31, 64, 77, 313, 333,
    28, 129, 67, 17, 15, 51, 12, 23, 27, 21, 57, 7, 37, 24, 69,
    7, 26, 297, 28, 50, 45, 83, 24, 41, 47, 16, 28, 14, 25, 14,
    8, 11, 68, 33, 26, 37, 6, 16, 19, 17, 76, 29, 27, 13, 8,
    12, 14, 7, 18, 24, 14, 25, 9, 11]
```

It becomes clear that the communities are quite unevenly sized. We can also ask for the community of a specific node:

```
In [14]: zeta.clusterOf(42)
Out[14]: 10

In [15]: zeta.getMembers(10)
        # output omitted
```

To get an overview of the community detection result, use the function `communities.inspectCommunities`. This displays a table with information about community sizes as well as the modularity value, a measure of the goodness of the partition and the presence of modular structure in the network.

```
In [5]: community.inspectCommunities(zeta, G)
-------------------  ---------
# communities         82
min community size     6
max community size   888
avg. community size  130.244
imbalance              6.77863
modularity             0.87673
-------------------  ---------
```

To target very large graphs (in the order of billions of edges), we recommend the `PLP` parallel label propagation method, which is faster and scales very well with the number of processors. Be aware however that the two methods can arrive at quite different results:

```
In [7]: community.inspectCommunities(community.PLP().run(G), G)
-------------------  ----------
# communities        979
min community size     2
max community size   411
avg. community size   10.9091
imbalance             37.3636
modularity             0.799206
-------------------  ----------
```

Compatibility with NetworkX

NetworkX is a Python package with a large collection of network analysis methods and graph algorithms. NetworKit provides functions to convert graph objects and thereby connects the two modules.

```
In [6]: import networkx
```

The function `nk2nx` converts a `NetworKit.Graph` to a `networkx.Graph`:

```
In [4]: nxG = nk2nx(G)

In [5]: nxG
Out[5]: <networkx.classes.graph.Graph at 0x7fb1a453f390>
```

Now we can also use some of the numerous NetworkX functions, e.g.

```
In [13]: networkx.degree_assortativity_coefficient(nxG)
Out[13]: 0.23821137170818882
```

The function `nx2nk` handles the other direction. For example, we can use graph generators not implemented by NetworKit:

```
In [3]: wsG = nx2nk(networkx.generators.watts_strogatz_graph(1000, 2, 0.1))
```

This opens up a wide range of possibilities which are not yet or will never be implemented within NetworKit. Note however that NetworkX is written mostly in pure Python, its data structures are more memory-intensive and its algorithms do not target very large graphs. You are likely to reach limits of your machine for graphs with millions of edges, while NetworKit aims for good performance for three more orders of magnitude.

Overview of Network Properties

Let us now inspect more basic properties of the graph. The functions for this are mostly located in the `properties` module (though module structure is subject to change). For example, this function gives us a tuple containing the number of nodes and the number of edges:

```
In [7]: properties.nm(G)
Out[7]: (10680, 24316)
```

To see the most important features of a network at a glance, try the function `properties.showProperties`. It prints a tabular overview (and can also display histograms on the terminal). For example, this is what it tells us about `PGPgiantcompo`:

```
In [11]: properties.showProperties(G)

    Network Properties
    ==================
    Basic Properties
    --------------  ------------
    nodes (n)       10680
    edges (m)       24316
    min. degree         1
    max. degree       205
    avg. degree         4.55356
    isolated nodes      0
    self-loops          0
    density             0.000426
    --------------  ------------
    Path Structure
```

```
-------------------------  -----
connected components           1
size of largest component  10680
diameter
avg. eccentricity
-------------------------  -----
Miscellaneous
--------------------  ------------
degree assortativity      0.238211
cliques               13814
--------------------  ------------
Community Structure
---------------------------------  -----------  --------
avg. local clustering coefficient               0.265945
PLP community detection
                                   communities  1196
                                   modularity   0.787809
PLM community detection
                                   communities  90
                                   modularity   0.872310
---------------------------------  -----------  --------
```

Some properties have not been calculated because they would take too much time. For some features, it is currently still necessary to convert the graph to NetworkX. The `showProperties` function is work in progress and will be continually improved.

Settings

Verbosity

The C++ kernel of NetworKit outputs text to the terminal via the log4cxx library. The amount of detail is controlled by the current log level, which is one of `TRACE`, `DEBUG`, `INFO`, `WARN`, and `ERROR`. By default, the log level is `ERROR`. You can set the log level at runtime and do this from the Python shell:

```
>>> NetworKit.configureLogging(loglevel="DEBUG")
>>> NetworKit.currentLogLevel()
    'ERROR'
```