NetworKit

NetworKit is a growing open-source toolkit for high-performance network analysis. Its aim is to provide tools for the analysis of large networks in the size range from thousands to billions of edges. For this purpose, it implements efficient graph algorithms, many of them parallel to utilize multicore architectures. These are meant to compute standard measures of network analysis, such as degree sequences, clustering coefficients and centrality (see next sections of this document for a list of existing features and a roadmap of planned features). In this respect, NetworKit is comparable to packages such as NetworkX, albeit with a focus on parallelism and scalability. NetworKit is also a testbed for algorithm engineering and contains a few novel algorithms from recently published research, especially in the area of community detection (see list of publications below).

As of release 2.0, NetworKit is meant to be built into a Python extension module. High-performance algorithms are written in C++ and exposed to Python via Cython. Python in turn gives us the ability to work interactively and a rich environment of tools for data analysis and scientific computing, such as `pandas`, `matplotlib`, `networkx`, `numpy` and `scipy`. A static command line program for community detection can still be built (see build instructions below), but we focus our efforts on the interactive Python environment.

Help

Documentation

In addition to this `Readme`, the `UserGuide` provides an introduction to the NetworKit tools. The `DevGuide` is meant for developers who would like to contribute. When using NetworKit as a Python module, refer to the docstrings of classes, methods and functions. C++ sources are also documented in Doxygen format.

To convert the documentation to PDF install the pandoc utility and call the script `docs2pdf.sh`.

E-Mail List

For questions regarding NetworKit, subscribe to our e-mail list (`networkit@ira.uka.de`) and feel free to ask.

Features

A not necessarily complete list of features:

Data Structures

- undirected graph with optional weights

Algorithms

- high-performance community detection
  - Modularity index (`community.Modularity`)
  - parallel Label propagation algorithm (`community.PLP`)
  - parallel Louvain method (`community.PLM`)
- various network properties
  - degree distribution (`properties.GraphProperties.averageLocalClusteringCoefficient`)
  - connected components (`properties.ConnectedComponents`)
  - local clustering coefficient (e.g. `properties.GraphProperties.averageLocalClusteringCoeffici`
- graph generators
  - Erdös-Renyi (`generators.ErdosRenyiGenerator`)
  - Barabasi-Albert (`generators.BarabasiAlbertGenerator`)
- traversal
  - breadth-first search (`graph.BFS`)
  - Dijkstra's algorithm (`graph.Dijkstra`)

Input/Output

- graph input/output formats
  - METIS
  - edge list
  - GraphViz
- `NetworkX` compatibility

Roadmap

A list of features likely to appear in future releases:

Data Structures

- directed graph
- graphs safe for concurrent modification
- improved support for attributed graphs

Algorithms

- core decomposition
- centrality
- dynamic community detection
- selective/local community detection
- coarsening
- matching
- partitioning
- layout and visualization

Input/Output

- file formats: gexf (Gephi), ...

Release History

2.0 (11/11/2013)

- NetworKit's becomes a Python extension module
- network analysis tools: degree distribution, clustering coefficient, connected components...
- added graph generators
- added I/O formats
- basic traversal algorithms

1.0 (20/03/2013)

- parallel community detection algorithms: PLP, PLM, EPP
- community detection metrics
- basic data structures

Credits

Main Developers

- Christian Staudt - `christian.staudt @ kit.edu` - Homepage
- Henning Meyerhenke - `meyerhenke @ kit.edu` - Homepage

Contributors

- Andreas Bilke
- Yassine Marrakchi
- Aleksejs Sazonovs
- Maximilian Vogel
- Miriam Beddig
- Stefan Bertsch
- Florian Weber
- Patrick Flick
- Daniel Hoske
- Jörg Weisbarth
- Guido Brückner

External Code

This program includes the *The Lean Mean C++ Option Parser* by Matthias S. Benkmann.

License

Requirements

Compiler:

A C++ compiler supporting C++11 (we use GCC 4.8). The compiler and linker flags `-fopenmp -std=c++11` are required.

Libraries:

- `OpenMP` for parallelism

To avoid possible binary incompatibilities, try to build these libraries with the same compiler that will be used to build NetworKit.

Building the Base

We recommend SCons for building the C++ part of NetworKit. Individual settings for your environment will be read from a configuration file. As an example, the file `build.conf.example` is provided. Copy this to `build.conf` and edit your environment settings. Then call `scons`.

The call to SCons has the following options:

```
scons --optimize=<level> --target=<target>
```

where <`level`> can be

- `D` debug
- `O` optimized
- `P` profiling

and <`target`> can be

- `Core` build NetworKit as a library, required by the Python shell
- `Tests` build executable for the unit tests
- `CommunityDetection` build executable for static, global community detection algorithms (deprecated)

For example, to build NetworKit as an optimized library, run scons –optimize=O –target=Core

To speed up the compilation on a multicore machine, you can append `-jX` where X denotes the number of threads to compile with.

Logging is enabled by default. If you want to disable logging functionality, add the following to your scons call:

```
--logging=no
```

Alternatively, the project can be built with Eclipse. Our Eclipse and CDT project files are included as examples in the `[eclipse/` directory. Copy them to the project file location, import the project into Eclipse and modify depending on your needs.

Test

You actually don't need to build and run our unit tests. However if you experience any issues with NetworKit, you might want to check, if NetworKit runs properly. Please refer to the `Unit Tests and Testing` section in our DevGuide.pdf

Building NetworKit as a Python Module

As of version 2.0 NetworKit is meant to be used as a Python extension module. To build the module, the following is required:

- Python 3 (>= 3.3 recommended)
- Cython

Additionally, the module uses the following external Python packages:

- `networkx`
- `tabulate`
- `scipy`
- `matplotlib`

The functionality of NetworKit available to Python is greatly reduced without these packages. These are best installed via `easy_install` or `pip`. (If you have multiple Python installations, be sure you use the commands matching your Python 3 version. In the following examples, `python3` and `pip3` will be used.)

```
pip3 install package_name
```

Then, switch to the top folder and run the script `setup.py` with the following options:

```
python3 setup.py build_ext --inplace [--optimize=V] [-jX]
```

The script will call scons to compile NetworKit as a library and then build the extensions in the folder `src/python`. By default, NetworKit will be build with the amount of available cores in optimized mode. It is possible the add the options `--optimize=V` and `-jN` the same way it can be done to a manual scons call. The setup script provides more functionality:

```
python3 setup.py develop [--uninstall] [--optimize=V] [-jX]
```

will compile NetworKit, build the extensions and on top of that temporarily install NetworKit so that it is available on the whole system. This can be undone by adding `--uninstall`.

```
python3 setup.py clean [--optimize=V]
```

will remove the extensions and its build folder as well as call scons to remove the NetworKit library and its build folder specified by `--optimize=V`.

Assuming the extension module `NetworKit` exists, it can be imported in python:

```
python3
>>> import NetworKit
```

Interactive Work with NetworKit

With NetworKit as a Python extension module, you get access to native high-performance code and can at the same time work interactively in the Python ecosystem. Although the standard Python interpreter works fine, we recommend IPython as a great environment for scientific computing. IPython can also be installed via `pip` or `easy_install`. For tab completion in ipython you may also need to install `readline`. The following should work: `pip3 install readline ipython`.

After the requirements are satisfied, start IPython and import NetworKit.

```
ipython3
>>> from NetworKit import *
```

Now you should be able to use NetworKit interactively. For usage examples, refer to the `UserGuide`.

IPython Notebook

We recommend that you familiarize yourself with NetworKit through experimenting with the interactive IPython Notebook `NetworKit_UserGuide.ipynb` located in the folder `NetworKit/cython/Notebooks`. To display and work with these notebooks, you have to start a local notebook server from the terminal with:

```
ipython3 notebook --pylab inline
```

Your default browser should open a web interface named "IPython Dashboard". You can either add `NetworKit_UserGuide.ipynb` from the above mentioned location, or you can point IPython to the location by starting it with

```
ipython3 notebook --pylab inline --notebook-dir=NetworKit/cython/Notebooks
```

The notebook appears in the list and you can start it by clicking on it.

Running Executables

Warning: As of version 2.0, NetworKit is meant to be used as a Python module, so the following interface will be phased out and is deprecated.

NetworKit-CommunityDetection

Required options for community detection are `--algorithm` and `--graph`.

Main algorithms:

```
PLP     Parallel Label Propagation
PLM     Parallel Louvain Method
EPP     Ensemble Preprocessing
```

Example calls:

```
./NetworKit-CommunityDetection-O --algorithm=PLP --graph=path/to/a.graph
./NetworKit-CommunityDetection-O --algorithm=EPP:4*PLP+PLM --graph=path/to/a.graph
./NetworKit-CommunityDetection-O --algorithm=PLM --graph=path/to/a.graph --runs=10 --sum
```

The `--graph` option accepts graph files in a format known as the METIS file format, a simple adjacency list format. Many example files can be found in the collection of the 10th DIMACS Implementation Challenge.

The default loglevel is INFO, add `--loglevel=DEBUG` for more or `--loglevel=ERROR` for less verbose output.

By default, all available threads will be used. To explicitly set the number of threads, use

```
--threads=8
```

To perform 42 runs of the algorithm per graph, add

```
--runs=42
```

To append key result data to a CSV file, add

```
--summary=/path/to/file.csv
```

To save the clustering produced, add

```
--saveClustering=/path/to/file.clust
```

Contribute

We would like to encourage contributions to the NetworKit source code. See the development guide (`DevGuide.mdown`) for instructions. For support please contact `christian.staudt @ kit.edu`.

Publications

The following is a list of publications on the basis of NetworKit. We ask you to cite the appropriate ones if you found NetworKit useful for your own research.

```
@inproceedings{sm2013ehpcdh,
    Author = {Christian L. Staudt and Henning Meyerhenke},
    Booktitle = {proceedings of the 2013 International Conference on Parallel Processing
    Date-Added = {2013-10-01 08:13:23 +0000},
    Date-Modified = {2013-10-01 08:13:23 +0000},
    Publisher = {Conference Publishing Services (CPS)},
    Title = {Engineering High-Performance Community Detection Heuristics for Massive Gra
    Year = {2013}}
```