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, but will be phased out (see build instructions below).

Help

E-Mail List

For questions regarding NetworKit, subscribe to our e-mail list and feel free to ask.

Features

Data Structures

- undirected graph (with optional weights) for parallel read access

Handling Data

- Graph Input/Output Formats
  - METIS
  - Edge List
  - GraphViz

Algorithms

- Community Detection
  - Modularity
  - Parallel Label Propagation (PLP)
  - Parallel Louvain Method (PLM)
- Components
  - Connected Components
- Generators
  - Erdös-Renyi
  - Barabasi-Albert
- Clustering

  - Local Clustering Coefficient
- Traversal
  - Breadth-First Search
- Partitioning
- Independent Sets
  - Luby's parallel independent set algorithm
- Coarsening
- Matching
- Subgraphs

NetworkX Compatibility

- conversion
  - `NetworKit.nk2nx`: `NetworKit.Graph` to `networkx.Graph`
  - `NetworKit.nx2nk`: `networkx.Graph` to `NetworKit.Graph`

Roadmap

Data Structures

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

Algorithms

- Cores
  - k-Core Decomposition
- Centrality
  - approximated Betweenness/Closeness

Release History

2.0

- NetworKit becomes a Python extension module

1.0

- static global community detection algorithms

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

External Code

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

License

The source code of this program is released under the MIT License. We ask you to cite us if you use this code in your project. Feedback is also welcome.

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
- `Googletest` for unit testing
- `log4cxx` for logging

The following preprocessor definitions (using the `-D` compiler flag) remove these dependencies, but also some functionality:

- `NOLOGGING` removes all log statements
- `NOLOG4CXX` replaces log statements with `std::cout` output if `log4cxx` is not available - loglevel `TRACE` becomes fixed.

Building the Base

We recommend SCons for building the C++ kernels 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.

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

Next, verify that the code was built correctly: Run all unit tests with

```
./NetworKit-Tests --tests --gtest_filter=*Test.test*
```

The expression after --gtest_filter= selects the tests.

Run performance tests with

```
./NetworKit-Tests --tests --gtest_filter=*Benchmark*
```

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 or newer - Cython

Additionally, the module uses the following external Python packages:

- networkx
- tabulate

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
```

The next step is to compile NetworKit as a library with

```
scons --optimize=O --target=Core
```

After everything is compiled, switch to the cython folder and use the build script (build.sh) or manually call

```
python3 setup.py build_ext --inplace
```

This will create the extension module NetworKit. The resulting module 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, here are some examples:

```
>>> G = readGraph("../input/jazz.graph")
>>> ccs = ConnectedComponents()
>>> ccs.run(G)
>>> ccs.numberOfComponents()
>>> GraphProperties.minMaxDegree(G)
>>> GraphProperties.averageLocalClusteringCoefficient(G)
```

Running the 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 --summary=path/to
```

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 Graphs},
    Year = {2013}}
```