NetworkX-METIS Documentation

Release 1.0

NetworkX Developers

August 18, 2015

Contents

Contents:

CHAPTER 1

Overview

NetworkX-METIS is an add-on for the NetworkX python package using METIS for graph partitioning.

NetworkX is a Python package for the creation, manipulation and study of the structure, dynamics, and functions of complex networkx. METIS is a C library written for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. NetworkX-METIS uses Cython to wrap the METIS library to make it available in Python.

1.1 Free software

NetworkX-METIS is free software; you can redistribute it and/or modify it under the terms of the **:doc:‘Apache License </reference/legal>‘_**. We welcome contributions from the community. Information on NetworkX development is found at the NetworkX Developer Zone at Github https://github.com/networkx/networkx-metis

1.2 History

NetworkX-METIS was born in 2014. The original version of the wrapper was designed and written by Yingchong Situ. The first public release as an add-on for NetworkX was made after a Google Summer of Code 2015 project, Implementing Add-on system of NetworkX.

1.2.1 What Next

- Installing
- Reference

CHAPTER 2

Download

2.1 Software

Source and binary releases: https://pypi.python.org/pypi/networkx-metis/

Github (latest development): https://github.com/networkx/networkx-metis/

Installing

Before installing NetworkX-METIS, you need to have setuptools , Cython and NetworkX installed.

3.1 Quick install

Get NetworkX-METIS from the Python Package Index at http://pypi.python.org/pypi/networkx-metis

or install it with

```
pip install networkx-metis
```

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 (at github.com) with manully checking out

```
https://github.com/networkx/networkx-metis
```

3.2 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 git source code repository.

Installation on Windows is largely the same as on Linux/Mac except that no "platform compiler" is pre-installed. So, an extra `--compiler` flag may be necessary to specify a compiler. A simple guide for installing and setting up the compiler is available here.

3.2.1 Source archive file

1. Download the source (tar.gz or zip file) from https://pypi.python.org/pypi/networkx-metis/ or get the latest development version from https://github.com/networkx/networkx-metis/
2. Unpack and change directory to the source directory (it should have the setup.py on top level).
3. Run

```
python setup.py build
```

   to build, and

```
python setup.py install
```

to install.

4. (Optional) Run `nosetests` to execute the tests if you have nose installed.

3.2.2 GitHub

1. Clone the networkx-metis repostitory

```
git clone https://github.com/networkx/networkx-metis.git
```

(see https://github.com/networkx/networkx-metis/ for other options)

2. Change directory to `networkx-metis`
3. Run

```
python setup.py build
```

to build, and

```
python setup.py install
```

to install.

4. (Optional) Run `nosetests` to execute the tests if you have nose installed.

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.

3.3 Requirements

3.3.1 Python

To use NetworkX-METIS you need Python 2.7, 3.2 or later.

3.3.2 NetworkX

To use NetworkX-METIS you need NetworkX 2.0 or later installed.

3.3.3 Cython

For NetworkX-METIS to work, you need Cython installed.

The easiest way to get Python and most optional packages is to install the Enthought Python distribution “Canopy”.

There are several other distributions that contain the key packages you need for scientific computing. See http://scipy.org/install.html for a list.

CHAPTER 4

Reference

Contents:

4.1 Copyright & License Notice

4.2 METIS Wrapper

4.2.1 METIS

Wrappers of METIS graph partitioning functions.

node_nested_dissection(G[, weight, options])	Compute a node ordering of a graph that reduces fill when the Laplacian ma
partition(G, nparts[, node_weight, ...])	Partition a graph using multilevel recursive bisection or multilevel multiway
vertex_separator(G[, weight, options])	Compute a vertex separator that bisects a graph.

nxmetis.node_nested_dissection

nxmetis.**node_nested_dissection**(*G*, *weight='weight'*, *options=None*)

Compute a node ordering of a graph that reduces fill when the Laplacian matrix of the graph is LU factorized. The algorithm aims to minimize the sum of weights of vertices in separators computed in the process.

Parameters

- **G** (*NetworkX graph*) – A graph.
- **weight** (*object, optional*) – The data key used to determine the weight of each node. If None, each node has unit weight. Default value: 'weight'.

- **options** (*MetisOptions, optional*) – METIS options. If None, the default options are used. Default value: None.

Returns **perm** – The node ordering.

Return type list of nodes

Raises `NetworkXError` – If the parameters cannot be converted to valid METIS input format, or METIS returns an error status.

nxmetis.partition

`nxmetis.`**`partition`**(*G*, *nparts*, *node_weight='weight'*, *node_size='size'*, *edge_weight='weight'*, *tpwgts=None*, *ubvec=None*, *options=None*, *recursive=False*)

Partition a graph using multilevel recursive bisection or multilevel multiway partitioning.

Parameters

- **G** (*NetworkX graph*) – An undirected graph.
- **nparts** (*int*) – Number of parts to partition the graph. It should be at least 2.
- **node_weight** (*object, optional*) – The data key used to determine the weight of each node. If None, each node has unit weight. Default value: 'weight'.
- **node_size** (*object, optional*) – The data key used to determine the size of each node when computing the total communication volumne. If None, each node has unit size. Default value: 'size'
- **edge_weight** (*object, optional*) – The data key used to determine the weight of each edge. If None, each edge has unit weight. Default value: 'weight'.
- **tpwgts** (*list of lists of floats, optional*) – The target weights of the partitions and the constraints. The target weight of the i-th partition and the j-th constraint is given by `tpwgts[i][j]` (the numbering for both partitions and constraints starts from zero). For each constraint the sum of the `tpwgts[][]` entries must be 1.0 (i.e., $\sum_i \mathrm{tpwgts}[i][j] = 1.0$). If None, the graph is equally divided among the partitions. Default value: None.
- **ubvec** (*list of floats, optional*) – The allowed load imbalance tolerance for each constraint. For the i-th and the j-th constraint, the allowed weight is the `ubvec[j] * tpwgts[i][j]` fraction of the j-th constraint's total weight. The load imbalances must be greater than 1.0. If None, the load imbalance tolerance is 1.001 if there is exactly one constraint or 1.01 if there are more. Default value: None.
- **options** (*MetisOptions, optional.*) – METIS options. If None, the default options are used. Default value: None.
- **recursive** (*bool, optional*) – If True, multilevel recursive bisection is used. Otherwise, multileve multilevel multiway partitioning is used. Default value: False.

Returns

- **objval** (*int*) – The edge-cut or the total communication volume of the partitioning solution. The value returned depends on the partitioining's objective function.
- **parts** (*lists of nodes*) – The partitioning.

Raises

- `NetworkXNotImplemented` – If the graph is directed or is a multigraph.
- `NetworkXError` – If the parameters cannot be converted to valid METIS input format, or METIS returns an error status.

nxmetis.vertex_separator

`nxmetis.`**`vertex_separator`**(*G*, *weight='weight'*, *options=None*)

Compute a vertex separator that bisects a graph. The algorithm aims to minimize the sum of weights of vertices in the separator.

Parameters

- **G** (*NetworkX graph*) – A graph.
- **weight** (*object, optional*) – The data key used to determine the weight of each node. If None, each node has unit weight. Default value: 'weight'.
- **options** (*MetisOptions, optional*) – METIS options. If None, the default options are used. Default value: None.

Returns **sep, part1, part2** – The separator and the two parts of the bisection represented as lists.

Return type lists of nodes

Raises `NetworkXError` – If the parameters cannot be converted to valid METIS input format, or METIS returns an error status.

4.2.2 Enums

MetisPType	Partitioning method.
MetisObjType	Type of objective.
MetisCType	Catching scheme to be used during coarsening.
MetisIPType	Algorithm used during initial partitioning.
MetisRType	Algorithm used for refinement.
MetisNumbering	Numbering scheme is used for the adjacency structure of a graph or the element-node structure of a mesh.
MetisDbgLvl	Amount of progress/debugging information will be printed during the execution of the algorithms.

nxmetis.enums.MetisPType

class `nxmetis.enums.`**`MetisPType`**

Partitioning method.

Attributes

`default`	Default partitioning method.
`kway`	Multilevel k-way partitioning.
`rb`	Multilevel recursive bisectioning.

nxmetis.enums.MetisObjType

class `nxmetis.enums.`**`MetisObjType`**

Type of objective.

Attributes

`cut`	Edge-cut minimization.
`default`	Default type of objective.
`vol`	Total communication volume minimization.

nxmetis.enums.MetisCType

class `nxmetis.enums.`**`MetisCType`**

Catching scheme to be used during coarsening.

Attributes

`default`	Default catching scheme.
`rm`	Random matching.
`shem`	Sorted heavy-edge matching.

nxmetis.enums.MetisIPType

class `nxmetis.enums.`**`MetisIPType`**

Algorithm used during initial partitioning.

Attributes

`default`	Default method for initial partitioning.
`edge`	Derive a separator from an edge cut.
`grow`	Grow a bisection using a greedy strategy.
`node`	Grow a bisection using a greedy node-based strategy.
`random`	Compute a bisection at random followed by a refinement.

nxmetis.enums.MetisRType

class `nxmetis.enums.`**`MetisRType`**

Algorithm used for refinement.

Attributes

`default`	Default method used for refinement.
`fm`	FM-based cut refinement.
`greedy`	Greedy-based cut and volume refinement.
`sep1sided`	One-sided node FM refinement.
`sep2sided`	Two-sided node FM refinement.

nxmetis.enums.MetisNumbering

class `nxmetis.enums.`**`MetisNumbering`**

Numbering scheme is used for the adjacency structure of a graph or the element-node structure of a mesh.

Attributes

`default`	Default numbering scheme.
`one`	Fortran-style one-based numbering.
`zero`	C-style zero-based numbering.

nxmetis.enums.MetisDbgLvl

class `nxmetis.enums.`**`MetisDbgLvl`**

Amount of progress/debugging information will be printed during the execution of the algorithms. Can be combined by bit-wise OR.

Attributes

`coarsen`	Display various statistics during coarsening.
`conninfo`	Display information related to the minimization of subdomain connectivity.
`contiginfo`	Display information related to the elimination of connected components.
`default`	Display default statistics.
`info`	Print various diagnostic messages.
`ipart`	Display various statistics during initial partitioning.
`moveinfo`	Display detailed information about vertex moves during refinement.
`refine`	Display various statistics during refinement.
`sepinfo`	Display information about vertex separators.
`time`	Perform timing analysis.

4.2.3 Types

`MetisOptions`(**kwargs)	Options controlling behaviors of METIS algorithms.

nxmetis.types.MetisOptions

class `nxmetis.types.`**`MetisOptions`**`(`***kwargs*`)`

Options controlling behaviors of METIS algorithms.

`__init__``(`***kwargs*`)`

Initializes a MetisOptions object. Values can be provided for some parameters as arguments.

Example

```
>>> options = MetisOptions(ncuts=2, niter=100)
```

Methods

`__init__`(**kwargs)	Initializes a MetisOptions object.

Attributes

`ccorder`	A boolean to detect & order connected components separately.
`compress`	A boolean to compress graph prior to ordering.
`contig`	A boolean to create contigous partitions.
`ctype`	Matching scheme to be used during coarsening.
`dbglvl`	Amount of progress/debugging information will be printed during the execution of the algorithms.
`iptype`	Algorithm used during initial partitioning.
`minconn`	Number of mimimum connectivity.
`ncuts`	Number of cuts.
`niter`	Number of refinement iterations.
`no2hop`	A boolean to perform a 2-hop matching.
`nseps`	Number of separators.
`numbering`	Numbering scheme is used for the adjacency structure of a graph or the element-node structure of a mesh.
`objtype`	Type of objective.
`pfactor`	Prunning factor for high degree vertices.
`ptype`	Types of Partitioning method.
`rtype`	Algorithm used for refinement.
`seed`	Random number seed.
`ufactor`	User-supplied ufactor.

CHAPTER 5

Bibliography

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

CHAPTER 7

Indices and tables

- genindex
- modindex
- search
- Glossary

Bibliography

[Langtangen04] H.P. Langtangen, “Python Scripting for Computational Science.”, Springer Verlag Series in Computational Science and Engineering, 2004.

[Martelli03] A. Martelli, “Python in a Nutshell”, O’Reilly Media Inc, 2003.