
METIS for Python Documentation

Release 0.1

Ken Watford

Jan 18, 2018

Contents

1	Installation	3
2	Example	5
3	Indices and tables	9
	Python Module Index	11

Wrapper for the METIS library for partitioning graphs (and other stuff).

This library is unrelated to PyMetis, except that they wrap the same library. PyMetis is a Boost Python extension, while this library is pure python and will run under PyPy and interpreters with similarly compatible ctypes libraries.

[NetworkX](#) is recommended for representing graphs for use with this wrapper, but it isn't required. Simple adjacency lists are supported as well.

The function of primary interest in this module is `part_graph()`.

Other objects in the module may be of interest to those looking to mangle their graph datastructures into the required format. Examples of this include the `networkx_to_metis()` and `adjlist_to_metis()` functions. These are automatically called by `part_graph()`, so there is little need to call them manually.

See the BitBucket [repository](#) for updates and issue reporting.

CHAPTER 1

Installation

It's on PyPI, so installation should be as easy as:

```
pip install metis
    -or-
easy_install metis
```

METIS itself is not included with this wrapper. Get it [here](#).

Note that the shared library is needed, and isn't enabled by default by the configuration process. Turn it on by issuing:

```
make config shared=1
```

Your operating system's package manager might know about METIS, but this wrapper was designed for use with METIS 5. Packages with METIS 4 will not work.

This wrapper uses a few environment variables:

METIS_DLL

This wrapper uses Python's ctypes module to interface with the METIS shared library. If it is unable to automatically locate the library, you may specify the full path to the library file in this environment variable.

METIS_IDXTYPEWIDTH

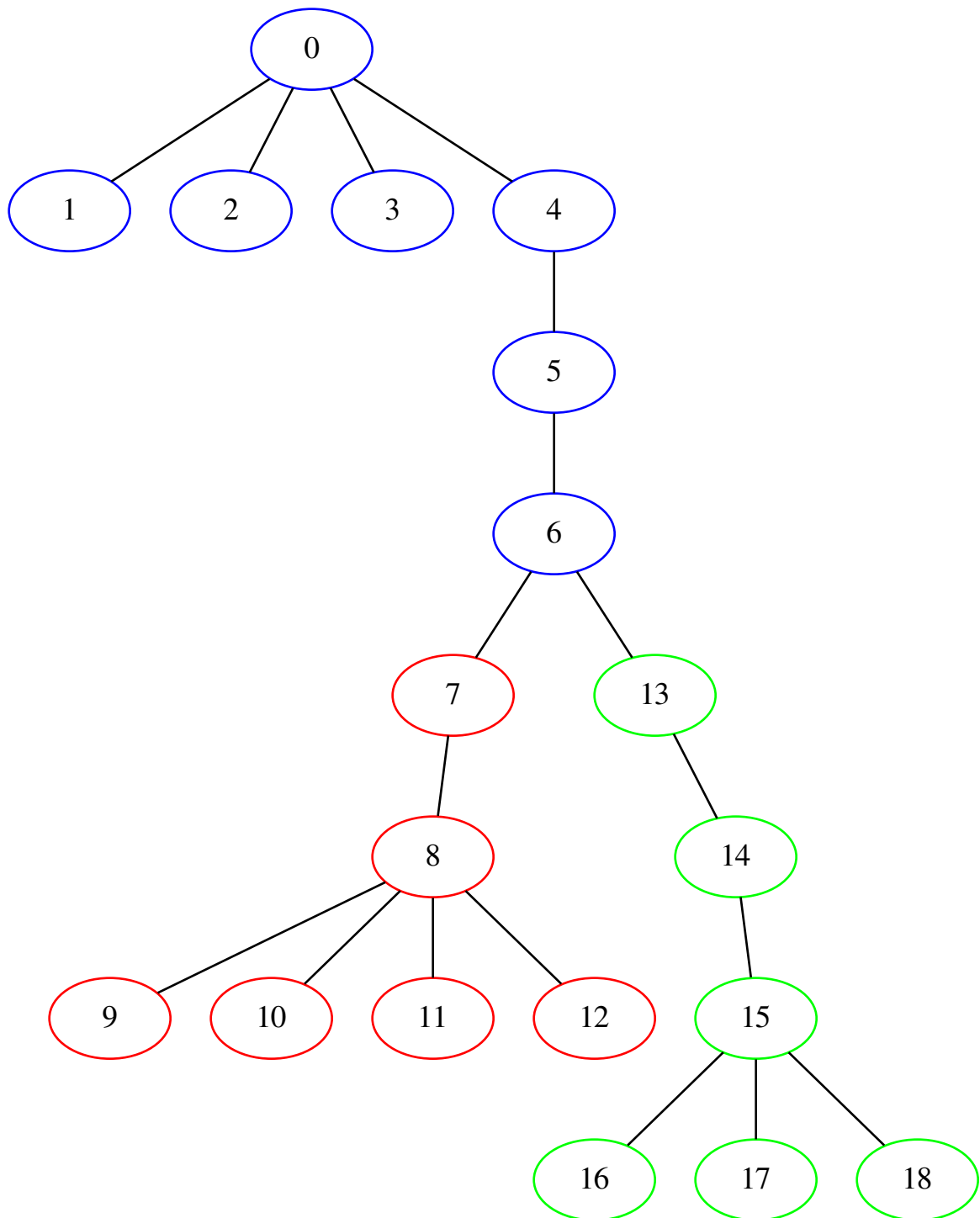
METIS_REALTYPEWIDTH

The sizes of the `idx_t` and `real_t` types are not easily determinable at runtime, so they can be provided with these environment variables. The default value for each of these (at both compile time and in this library) is 32, but they may be set to 64 if desired. If the values do not match what was used to compile the library, Bad Things(TM) will occur.

CHAPTER 2

Example

```
>>> import networkx as nx
>>> import metis
>>> G = metis.example_networkx()
>>> (edgecuts, parts) = metis.part_graph(G, 3)
>>> colors = ['red', 'blue', 'green']
>>> for i, p in enumerate(parts):
...     G.node[i]['color'] = colors[p]
...
>>> nx.write_dot(G, 'example.dot') # Requires pydot or pygraphviz
```



`metis.part_graph(graph, nparts=2, tpwgts=None, ubvec=None, recursive=False, **opts)`

Perform graph partitioning using k-way or recursive methods.

Returns a 2-tuple (*objval*, *parts*), where *parts* is a list of partition indices corresponding and *objval* is the value of the objective function that was minimized (either the edge cuts or the total volume).

Parameters

- **graph** – may be a NetworkX graph, an adjacency list, or a METIS_Graph named tuple. To use the named tuple approach, you'll need to read the METIS manual for the meanings of the fields.

See `networkx_to_mmetis()` for help and details on how the graph is converted and how node/edge weights and sizes can be specified.

See `adjlist_to_mmetis()` for information on the use of adjacency lists. The extra `nodew` and `nodesz` keyword arguments of that function may be given directly to this function and will be forwarded to the converter. Alternatively, a dictionary can be provided as `graph` and its items will be passed as keyword arguments.

- **nparts** – The target number of partitions. You might get fewer.
- **tpwgts** – Target partition weights. For each partition, there should be one (float) weight for each node constraint. That is, if *nparts* is 3 and each node of the graph has two weights, then *tpwgts* might look like this:

```
[(0.5, 0.1), (0.25, 0.8), (0.25, 0.1)]
```

This list may be provided flattened. The internal tuples are for convenience. The partition weights for each constraint must sum to 1.

- **ubvec** – The load imbalance tolerance for each node constraint. Should be a list of floating point values each greater than 1.
- **recursive** – Determines whether the partitioning should be done by direct k-way cuts or by a series of recursive cuts. These correspond to `METIS_PartGraphKway()` and `METIS_PartGraphRecursive()` in METIS's C API.

Any additional METIS options may be specified as keyword parameters.

For k-way clustering, the appropriate options are:

```
objtype   = 'cut' or 'vol'
ctype     = 'rm' or 'shem'
iptype    = 'grow', 'random', 'edge', 'node'
rtype     = 'fm', 'greedy', 'sep2sided', 'sep1sided'
ncuts     = integer, number of cut attempts (default = 1)
niter     = integer, number of iterations (default = 10)
ufactor   = integer, maximum load imbalance of (1+x)/1000
minconn   = bool, minimize degree of subdomain graph
contig    = bool, force contiguous partitions
seed      = integer, RNG seed
numbering = 0 (C-style) or 1 (Fortran-style) indices
dbg1vl    = Debug flag bitfield
```

For recursive clustering, the appropriate options are:

```
ctype     = 'rm' or 'shem'
iptype    = 'grow', 'random', 'edge', 'node'
rtype     = 'fm', 'greedy', 'sep2sided', 'sep1sided'
ncuts     = integer, number of cut attempts (default = 1)
```

```
niter      = integer, number of iterations (default = 10)
ufactor    = integer, maximum load imbalance of (1+x)/1000
seed       = integer, RNG seed
numbering  = 0 (C-style) or 1 (Fortran-style) indices
dbg1vl     = Debug flag bitfield
```

See the METIS manual for specific meaning of each option.

`metis.networkx_to_metis(G)`

Convert NetworkX graph into something METIS can consume. The graph may specify weights and sizes using the following graph attributes:

- `edge_weight_attr`
- `node_weight_attr` (multiple names allowed)
- `node_size_attr`

For example:

```
>>> G.adj[0][1]['weight'] = 3
>>> G.node[0]['quality'] = 5
>>> G.node[0]['specialness'] = 8
>>> G.graph['edge_weight_attr'] = 'weight'
>>> G.graph['node_weight_attr'] = ['quality', 'specialness']
```

If `node_weight_attr` is a list instead of a string, then multiple node weight labels can be provided.

All weights must be integer values. If an attr label is specified but a node/edge is missing that attribute, it defaults to 1.

If a graph attribute is not provided, no default is used. That is, if `edge_weight_attr` is not set, then 'weight' is not used as the default, and the graph will appear unweighted to METIS.

`metis.adjlist_to_metis(adjlist, nodew=None, nodesz=None)`

Rudimentary adjacency list converter. Primarily of use if you don't have or don't want to use NetworkX.

Parameters

- **adjlist** – A list of tuples. Each list element represents a node or vertex in the graph. Each item in the tuples represents an edge. These items may be single integers representing neighbor index, or they may be an (index, weight) tuple if you want weighted edges. Default weight is 1 for missing weights.

The graph must be undirected, and each edge must be represented twice (once for each node). The weights should be identical, if provided.

- **nodew** – is a list of node weights, and must be the same size as *adjlist* if given. If desired, the elements of *nodew* may be tuples of the same size (≥ 1) to provide multiple weights for each node.
- **nodesz** – is a list of node sizes. These are relevant when doing volume-based partitioning.

Note that all weights and sizes must be non-negative integers.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

m

metis, [1](#)

A

`adjlist_to_metis()` (in module `metis`), 8

E

environment variable

`METIS_DLL`, 3

`METIS_IDXTYPEWIDTH`, 3

`METIS_REALTYPEWIDTH`, 3

M

`metis` (module), 1

N

`networkx_to_metis()` (in module `metis`), 8

P

`part_graph()` (in module `metis`), 6