cpnet

Release 0.0.15

Sadamori Kojaku

Jul 11, 2022
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Installation</td>
<td>3</td>
</tr>
<tr>
<td>1.1</td>
<td>cpnet</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Examples</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Indices and tables</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Python Module Index</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>21</td>
</tr>
</tbody>
</table>
This package contains algorithms for detecting core-periphery structure in networks. All algorithms are implemented in Python, with speed accelerations by numba, and can be used with a small coding effort.

See the project page for the usage of this package.
Before installing this package, make sure that you have a **Python with version 3.6 or above**.
pip is the most easiest way to install:

```
pip install cpnet
```

For conda users, although the package can be install using pip without problem in conda environement, you may want to avoid mixing pip with conda. In this case, we recommend making a link to the package:

```
git clone https://github.com/skojaku/core-periphery-detection
cd core-periphery-detection
conda develop
```

This package is under active development. If you have issues and feature requests, please raise them through Github.

### 1.1 cpnet

#### 1.1.1 cpnet package

**Submodules**

**cpnet.BE module**

```python
class cpnet.BE.BE(num_runs=10)
    Bases: CPAlgorithm
    Borgatti Everett algorithm.
```

Algorithm for finding single core-periphery pair in networks.


```python
>>> import cpnet
>>> be = cpnet.BE()  
>>> be.detect(G)
>>> pair_id = be.get_pair_id()
>>> coreness = be.get_coreness()
```

**Note:**

- [ ] weighted
• [ ] directed
• [ ] multiple groups of core-periphery pairs
• [ ] continuous core-periphery structure

detect$(G)$
Detect core-periphery structure.

Parameters

$G$ (networkx.Graph or scipy sparse matrix) – Graph

Returns
None

Return type
None

cnet.CPAAlgorithm module
class cnet.CPAAlgorithm.CPAlgorithm
Bases: object
depairing$(labels, d)$
abstract detect()
Private.
get_coreness()
Get the coreness of each node.
get_pair_id()
pairing$(labels, a)$
score$(G, c, x)$
Get score of core-periphery pairs.

cnet.Cucuringu module
class cnet.Cucuringu.LapCore$(beta=0.1)$
Bases: CPAlgorithm
LapCore algorithm.

>>> import cpnet
>>> lc = cpnet.LapCore()
>>> lc.detect(G)
>>> pair_id = lc.get_pair_id()
>>> coreness = lc.get_coreness()
Note:
- [ ] weighted
- [ ] directed
- [ ] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

detect($G$)
Detect core-periphery structure.

Parameters
$G$ (networkx.Graph or scipy sparse matrix) -- Graph

class cpnet.Cucuringu.LapSgnCore($\beta=0.1$)
Bases: CPAlgorithm
LowSgnCore algorithm.


```python
>>> import cpnet
>>> lsc = cpnet.LapSgnCore()
>>> lsc.detect(G)
>>> pair_id = lsc.get_pair_id()
>>> coreness = lsc.get_coreness()
```

Note:
- [ ] weighted
- [ ] directed
- [ ] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

detect($G$)
Detect a single core-periphery pair.

Parameters
$G$ (networkx.Graph or scipy sparse matrix) -- Graph

class cpnet.Cucuringu.LowRankCore($\beta=0.1$)
Bases: CPAlgorithm
LowRankCore algorithm.

```python
>>> import cpnet
>>> lrc = cpnet.LowRankCore()
>>> lrc.detect(G)
>>> pair_id = lrc.get_pair_id()
>>> coreness = lrc.get_coreness()
```

**Note:**

- [ ] weighted
- [ ] directed
- [ ] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

**detect**(G)

Detect a single core-periphery pair.

**Parameters**

G (networkx.Graph or scipy sparse matrix) – Graph

**cpnet.Divisive module**

**class** cpnet.Divisive.Divisive(num_runs=10)

Bases: CPAlgorithm

Divisive algorithm.

This algorithm partitions a network into communities using the Louvain algorithm. Then, it partitions each community into a core and a periphery using the BE algorithm. The quality of a community is computed by that equipped with the BE algorithm.


```python
>>> import cpnet
>>> alg = cpnet.Divisive()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```

**Note:**

- [x] weighted
- [ ] directed
- [x] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

**detect**(G)

Detect core-periphery structure.
Parameters

\( G(\text{networkx.Graph or scipy sparse matrix}) \) – Graph

Returns

None

Return type

None

cpnet.KM_ER module

class cpnet.KM_ER.KM_ER(num_runs=10)

Bases: CPAlgorithm

Kojaku-Masuda algorithm with the Erdos-Renyi random graph.
This algorithm finds multiple core-periphery pairs in networks.


```python
>>> import cpnet
>>> alg = cpnet.KM_ER()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```

Note:

- [x] weighted
- [ ] directed
- [x] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

detect\((G)\)

Detect core-periphery structure.

Parameters

\( G(\text{networkx.Graph or scipy sparse matrix}) \) – Graph

Returns

None

Return type

None

significance()
**cpnet.KM_config module**

class cpnet.KM_config.KM_config(num_runs=10)

Bases: CPAlgorithm

Kojaku-Masuda algorithm with the configuration model.

This algorithm finds multiple core-periphery pairs in networks.


```python
>>> import cpnet
>>> alg = cpnet.KM_config()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```

**Note:**

- [x] weighted
- [] directed
- [x] multiple groups of core-periphery pairs
- [] continuous core-periphery structure

**detect(G)**

Detect core-periphery structure.

**Parameters**

- G (networkx.Graph or scipy sparse matrix) – Graph

**Returns**

- None

**Return type**

- None

**significance()**

---

**cpnet.Lip module**

class cpnet.Lip.Lip

Bases: CPAlgorithm

Lip’s algorithm.


```python
>>> import cpnet
>>> alg = cpnet.Lip()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```
Note:

• [ ] weighted
• [ ] directed
• [ ] multiple groups of core-periphery pairs
• [ ] continuous core-periphery structure

\texttt{detect}(G)

Detect core-periphery structure.

\textbf{Parameters}

\texttt{G (networkx.Graph or scipy sparse matrix)} – Graph

\textbf{Returns}

None

\textbf{Return type}

None

cpnet.MINRES module

class \texttt{cpnet.MINRES}(num_runs=10)

Bases: \texttt{CPAlgorithm}

MINRES algorithm.


\begin{verbatim}
>>> import cpnet
>>> alg = cpnet.MINRES()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
\end{verbatim}

Note:

• [x] weighted
• [ ] directed
• [ ] multiple groups of core-periphery pairs
• [x] continuous core-periphery structure

\texttt{detect}(G)

Detect core-periphery structure.

\textbf{Parameters}

\texttt{G (networkx.Graph or scipy sparse matrix)} – Graph

\textbf{Returns}

None
Return type
None

**cpnet.Rombach module**

class `cpnet.Rombach.Rombach`(*num_runs=10, alpha=0.5, beta=0.8, algorithm='ls')

Bases: `CPAlgorithm`

Rombach’s algorithm for finding continuous core-periphery structure.


```python
>>> import cpnet

>>> alg = cpnet.Rombach()

>>> alg.detect(G)

>>> pair_id = alg.get_pair_id()

>>> coreness = alg.get_coreness()
```

**Note:**
- [x] weighted
- [ ] directed
- [ ] multiple groups of core-periphery pairs
- [x] continuous core-periphery structure

`detect(G)`
Detect core-periphery structure.

**Parameters**

- `G` (networkx.Graph or scipy sparse matrix) – Graph

**Returns**
None

**Return type**
None

class `cpnet.Rombach.SimAlg`(A, x, alpha, beta)

Bases: `Annealer`

corevector(*x, alpha, beta*)

detect(*G*)

Calculate state’s energy

eval(od)

move()

Swaps two nodes.
**cpnet.Rossa module**

```python
>>> import cpnet
>>> alg = cpnet.Rossa()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```

**Note:**
- [x] weighted
- [x] directed
- [ ] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

**detect(G)**

Detect core-periphery structure.

Parameters

- `G` *(networkx.Graph or scipy sparse matrix)* – Graph

Returns

None

Return type

None

**cpnet.Surprise module**

```python
>>> import cpnet
>>> alg = cpnet.Surprise()
>>> alg.detect(G)
>>> pair_id = alg.get_pair_id()
>>> coreness = alg.get_coreness()
```

**Note:**
- [x] weighted
- [x] directed
- [ ] multiple groups of core-periphery pairs
- [ ] continuous core-periphery structure

**cpnet.Surprise**(num_runs=10)

Core-periphery detection by Surprise.

J. van Lith de Jeude, G. Caldarelli, T. Squartini. Detecting Core-Periphery Structures by Surprise. EPL, 125, 2019
• [ ] weighted
• [ ] directed
• [ ] multiple groups of core-periphery pairs
• [ ] continuous core-periphery structure

**detect**(*G*)
Detect core-periphery structure.

**Parameters**

* *G* (networkx.Graph or scipy sparse matrix) – Graph

**Returns**

None

**Return type**

None

---

**cpnet.adam module**

**class** cpnet.adam.ADAM
Bases: object

**update**(*theta, grad, lasso_penalty*, **positiveConstraint=False*)
Ascending.

**cpnet.qstest module**

**cpnet.qstest.config_model**(*G*)

**cpnet.qstest.erdos_renyi**(*G*)

**cpnet.qstest.qstest**(*pair_id*, coreness, *G*, cpa, significance_level=0.05, **null_model=<function config_model>, sfunc=<function sz_n>, num_of_rand_net=100, q_tilde=[], s_tilde=[], **params*)
(q,s)-test for core-periphery structure.

**Parameters**

• **pair_id** (dict) – node i belongs to pair pair_id[i]
• **coreness** (dict) – node i is a core (x[i]=1) or a periphery (x[i]=0)
• *G* (networkx.Graph or scipy sparse matrix) – Network
• **cpa** (CPAlgorithm) – algorithm that detects the core-periphery structure in question
• **significance_level** (float, optional) – Significance level, defaults to 0.05
• **null_model** (func, optional) – funcion to produce a null model, defaults to config_model
• **sfunc** (func, optional) – Size function that calculates the size of a community, defaults to sz_n
• **num_of_thread** (int, optional) – Number of threads, defaults to 4
- `num_of_rand_net (int, optional)` – Number of random networks, defaults to 300
- `q_tilde (list, optional)` – pre-computed sampled of strength \( q \) of core-periphery structure, defaults to \([]\)
- `s_tilde (list, optional)` – pre-computed sample of the size of a core-periphery pair, defaults to \([]\)

```python
>>> import cpnet
g = cpnet.KM_config()
g.detect(G)
pair_id = km.get_pair_id()
coreness = km.get_coreness()
sig_pair_id, sig_coreness, significance, p_values = cpnet.qstest(pair_id,
                                                          coreness, G, km)
```

**cpnet.qstest.sampling** \((G, cpa, sf, n, null_model)\)

**cpnet.qstest.sz_degree** \((network, c, x)\)

**cpnet.qstest.sz_n** \((network, c, x)\)

**cpnet.utils module**

- `cpnet.utils.calc_node_pos` \((G, layout_algorithm)\)
- `cpnet.utils.classify_nodes` \((G, c, x, max_num=None)\)
- `cpnet.utils.draw` \((G, c, x, ax, draw_edge=True, font_size=0, pos=None, cmap=None, max_group_num=None,
                          draw_nodes_kwd={}, draw_edges_kwd={'edge_color': '#adadad'}, draw_labels_kwd={},
                          layout_algorithm=None)\)

Plot the core-periphery structure in the networks.

**Parameters**

- `G (networkx.Graph)` – Graph
- `c (group membership c[i] of i)` – dict
- `x (dict)` – core (\( x[i]=1 \)) or periphery (\( x[i]=0 \))
- `ax (matplotlib.pyplot.ax)` – axis
- `draw_edge (bool, optional)` – whether to draw edges, defaults to True
- `font_size (int, optional)` – font size for node labels, defaults to 0
- `pos (dict, optional)` – pos[i] is the xy coordinate of node i, defaults to None
- `cmap (matplotlib.cmap, optional)` – colormap defaults to None
- `max_group_num (int, optional)` – Number of groups to color, defaults to None
- `draw_nodes_kwd (dict, optional)` – Parameter for networkx.draw_networkx_nodes, defaults to {}\n- `draw_edges_kwd (dict, optional)` – Parameter for networkx.draw_networkx_edges, defaults to \{"edge_color": "#adadad"\} \n- `draw_labels_kwd (dict, optional)` – Parameter for networkx.draw_networkx_labels, defaults to {}
• layout_kwd (dict, optional) – layout keywords, defaults to {}

Returns
(ax, pos)

Return type
matplotlib.pyplot.ax, dict

cpnet.utils.draw_interactive(G, c, x, hover_text=None, node_size=10.0, pos=None, cmap=None)
cpnet.utils.set_node_colors(c, x, cmap, colored_nodes)
cpnet.utils.to_adjacency_matrix(net)
cpnet.utils.to_nxgraph(net)

Module contents
CHAPTER TWO

EXAMPLES

- Example 1 (Detection of core-periphery structure)
- Example 2 (Statistical test)
- Example 3 (Case study: Political blog network)
- Example 4 (Case study: Airport network)
CHAPTER
THREE

INDICES AND TABLES

• genindex
• modindex
• search


PYTHON MODULE INDEX

C
  cpnet, 14
  cpnet.adam, 12
  cpnet.BE, 3
  cpnet.CPAlgorithm, 4
  cpnet.Cucuringu, 4
  cpnet.Divisive, 6
  cpnet.KM_config, 8
  cpnet.KM_ER, 7
  cpnet.Lip, 8
  cpnet.MINRES, 9
  cpnet.qstest, 12
  cpnet.Rombach, 10
  cpnet.Rossa, 11
  cpnet.Surprise, 11
  cpnet.utils, 13
M
MINRES (class in cpnet.MINRES), 9
module
cpnet, 14
cpnet.adam, 12
cpnet.BE, 3
cpnet.CPAlgorithm, 4
cpnet.Cucuringu, 4
cpnet.Divisive, 6
cpnet.KM_config, 8
cpnet.KM_ER, 7
cpnet.Lip, 8
cpnet.MINRES, 9
cpnet.qstest, 12
cpnet.Rombach, 10
cpnet.Rossa, 11
cpnet.Surprise, 11
cpnet.utils, 13
move() (cpnet.Rombach.SimAlg method), 10

P
pairing() (cpnet.CPAlgorithm.CPAlgorithm method), 4

Q
qstest() (in module cpnet.qstest), 12

R
Rombach (class in cpnet.Rombach), 10
Rossa (class in cpnet.Rossa), 11

S
sampling() (in module cpnet.qstest), 13
score() (cpnet.CPAlgorithm.CPAlgorithm method), 4
set_node_colors() (in module cpnet.utils), 14
significance() (cpnet.KM_config.KM_config method), 8
significance() (cpnet.KM_ER.KM_ER method), 7
SimAlg (class in cpnet.Rombach), 10
Surprise (class in cpnet.Surprise), 11
sz_degree() (in module cpnet.qstest), 13
sz_n() (in module cpnet.qstest), 13

T
to_adjacency_matrix() (in module cpnet.utils), 14
to_nxgraph() (in module cpnet.utils), 14

U
update() (cpnet.adam.ADAM method), 12