Bipartite Partial Configuration Model Documentation

Release 1.0

Mika J. Straka

January 20, 2017

The Bipartite Partial Configuration Model (BiPCM) is a statistical null model for binary bipartite networks. It offers an unbiased method of analyzing node similarities and obtaining statistically validated monopartite projections [Saracco2016].

The BiPCM is related to the Bipartite Configuration Model (BiCM) [Saracco2015], but imposes only constraints on the degrees of one bipartite node layer. It belongs to a series of entropy-based null model for binary biparite networks, see also

- BiCM - Bipartite Configuration Model
- BiRG - Bipartite Random Graph

Please consult the original articles for details about the underlying methods and applications to user-movie and international trade databases [Saracco2016], [Straka2016].

An example case is illustrated in the *Tutorial*.

CHAPTER

ONE

HOW TO CITE

If you use the `bipcm` module, please cite its location on Github and the original article [Saracco2016].

1.1 References

CHAPTER

TWO

GETTING STARTED

2.1 Overview

The `bipcm` module is an implementation of the Bipartite Partial Configuration Model (BiPCM) as described in the article [Saracco2016]. The BiPCM can be used as a statistical null model to analyze the similarity of nodes in undirected bipartite networks. The similarity criterion is based on the number of common neighbors of nodes, which is expressed in terms of Λ-motifs in the original article [Saracco2016]. Subsequently, one can obtain unbiased statistically validated monopartite projections of the original bipartite network.

The construction of the BiPCM, just like the related BiCM and BiRG models, is based on the generation of a grand-canonical ensemble of bipartite graphs subject to certain constraints. The constraints can be of different types. For instance, in the Bipartite Random Graph (BiRG) the total number of edges is fixed. In the case of the BiCM the average degrees of all the nodes are constrained. In the BiPCM, on the other hand, only the degrees of one bipartite layer are constrained.

The average graph of the ensemble can be calculated analytically using the entropy-maximization principle and provides a statistical null model, which can be used for establishing statistically significant node similarities. In general, they are referred to as entropy-based null models. For more information and a detailed explanation of the underlying methods, please refer to [Saracco2016].

By using the `bipcm` module, the user can obtain the BiPCM null model which corresponds to the input matrix representing an undirected bipartite network. To address the question of node similarity, the p-values of the observed numbers of common neighbors can be calculated and used for statistical verification. For an illustration and further details, please refer to [Saracco2016] and [Straka2016].

2.1.1 Dependencies

`bipcm` is written in *Python 2.7* and uses the following modules:

- poibin Module for the Poisson Binomial probability distribution
- scipy
- numpy
- doctest For unit testing

2.2 BiPCM Quickstart

The calculation of the p-values of node similarities with the `bipcm` module is straightforward as shown below. The validated node similarities can be used to obtain an unbiased monopartite projection of the bipartite network, as illustrated in [Saracco2016].

For more detailed explanations of the methods, please refer to [Saracco2016], the *Tutorial* and the *API*.

2.2.1 Calculating the p-values of the node similarities

Be `mat` a two-dimensional binary NumPy array, which describes the biadjacency matrix of an undirected bipartite network. The nodes of the two bipartite layers are ordered along the columns and rows, respectively. In the algorithm, the two layers are identified by the boolean values `True` for the **row-nodes** and `False` for the **column-nodes**.

Import the module and initialize the Bipartite Partial Configuration Model:

```
>>> from src.bipcm import BiPCM
>>> pcm = BiPCM(bin_mat=mat, constraint=<bool>)
```

The parameter `constraint` specifies whether the degrees of the row-nodes (`constraint = True`) or the degrees of the column-nodes (`constraint = False`) should be constrained.

In order to analyze the similarity of the row-layer nodes and to save the p-values of the corresponding Λ-motifs, i.e. of the number of shared neighbors [Saracco2016], use:

```
>>> pcm.lambda_motifs_main(bip_set=True, filename=<filename>)
```

For the column-layer nodes, use:

```
>>> pcm.lambda_motifs_main(bip_set=False, filename=<filename>)
```

`bip_set` selects the bipartite node set for which the p-values should be calculated and saved. The filename *<filename>* should contain a relative path declaration. The default name of the output file is *pval_constr_<constraint>_proj_<bip_set>.csv*, where *<constraint>* and *<bip_set>* are either *rows* or *columns* depending on the degree constraint and the parameter choice in `lambda_motifs_main`. By default, the values in the file are separated by tabs, which can be changed using the `delim` keyword.

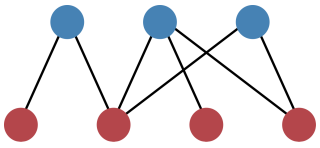
Subsequently, the p-values can be used to perform a multiple hypotheses testing and to obtain statistically validated monopartite projections [Saracco2016].

If the p-values should not be saved but returned by `lambda_motifs_main`, use:

```
>>> pcm.lambda_motifs_main(bip_set=True, write=False)
```

2.3 Tutorial

The tutorial will take you step by step from the biadjacency matrix of a real-data network to the calculation of the p-values. Our example bipartite network will be the following:



The structure of the network can be caught in the biadjacency matrix. In our case, the matrix is

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that the nodes of the layers of the bipartite network are ordered along the rows and the columns, respectively. In the algorithms, the two layers are identified by the boolean values `True` for the **row-nodes** and `False` for the

column-nodes. In our example image, the row-nodes are colored in blue (top layer) and the column-nodes in red (bottom layer).

Note: Tutorial has to be finished.

2.4 Testing

The methods in the `bipcm` module have been implemented using doctests. To run the tests, execute:

```
>>> python -m doctest bipcm_tests.txt
```

from the folder *src* in the command line. If you want to run the tests in verbose mode, use:

```
>>> python -m doctest -v bipcm_tests.txt
```

Note that *bipcm.py* and *bipcm_tests.txt* have to be in the same directory to run the test.

2.5 API

API for the methods in the `bipcm` module.

class `bipcm.`**`BiPCM`**(*bin_mat*, *constraint*)

Bipartite Partial Configuration Model for binary bipartite networks.

This class implements the Bipartite Partial Configuration Model (BiPCM), which can be used as a null model for the analysis of undirected and binary bipartite networks. The class provides methods to calculate the biadjacency matrix of the null model and to quantify node similarities in terms of p-values.

static `check_constraint`(*constraint*)

Check that the constraint parameter is either `True` of `False`.

Parameters **constraint** (*bool*) – constrains the degrees of either the row-nodes (`True`) or column-nodes (`False`)

Raises **AssertionError** raise an error if the constraint is neither `True` nor `False`

`check_input_matrix_is_binary`()

Check that the entries of the input matrix are 0 or 1.

Raises **AssertionError** raise an error if the input matrix is not binary

`get_edge_prob_seq`()

Return an array with the link probabilities of the BiPCM null model.

In the first part of the array, the row degrees are fixed. In the second part, the column degrees are fixed.

Returns array of link probabilities

Return type numpy.array

`get_lambda_motif_matrix`(*mm*, *bip_set=False*)

Return the number of Λ-motifs as found in `mm`.

Given the binary input matrix `mm`, count the number of Λ-motifs between node couples of the bipartite layer specified by `bip_set`.

Parameters

- **mm** (*numpy.array*) – binary matrix

- **bip_set** (*bool*) – selects row-nodes (`True`) or column-nodes (`False`)

Returns square matrix of observed Λ-motifs

Return type numpy.array

Raises

- **NameError** – raise an error if the parameter `bip_set` is neither `True` nor `False`
- **AssertionError** – raise an error if shape of the probability matrix is not correct

get_lambda_pvalues(*plam_mat*, *nlam_mat*, *bip_set=False*)

Return the p-values for the Λ-motifs in `nlam_mat`.

Calculate the p-values for the numbers of observed Λ-motifs as given in the parameter `nlam_mat` for the bipartite node layer `bip_set`. The probabilities for the single Λ-motifs are given in `plam_mat`.

If `bip_set` corresponds to the constrained bipartite node set, the Λ-motifs follow a Binomial probability distribution. Otherwise, all the node pairs follow the same Poisson Binomial probability distribution. The p-values are calculated as

$$p_{val}(k) = Pr(X >= k) = 1 - Pr(X < k) = 1 - cdf(k) + pmf(k)$$

Note: The lower triangular part (including the diagonal) of the returned matrix is set to zero.

Parameters

- **plam_mat** (*numpy.array*) – matrix of Λ-motif probabilities
- **nlam_mat** (*numpy.array*) – matrix of observed number of Lambda motifs
- **bip_set** (*bool*) – selects row-nodes (`True`) or column-nodes (`False`)

Returns matrix of the p-values for the Λ-motifs

Return type numpy.array

Raises

- **NameError** – raise an error if the parameter `bip_set` is neither `True` nor `False`
- **AssertionError** – raise an error if shapes of the probability matrix and the matrix with the number of Λ-motifs are not equal

get_plambda_matrix()

Return the Λ-motif probability matrix.

Return a square matrix M of Lambda probabilities for the nodes given the degree constraints on the node set self.const_set.

Note: If N_i are the nodes with constrained degrees, $M_{ij} = p(\Lambda_{ij})$ is the probability of nodes $i, j \in N_i$ sharing one common neighbor, whereas M_{ii} is the probability that two nodes of the opposite layer have node $i \in N_i$ as a common neighbor. The lower triangular part of M excluding the diagonal is set to 0 since the matrix is symmetric.

Returns Λ-motif probability matrix

Return type numpy.array

get_proj_pmat (*plam_mat*, *nlam_mat*, *bip_set=False*)
Return the probabilities of the observed Λ-motifs.

The probabilities of the Λ-motifs between the nodes specified by `bip_set` in the input matrix are calculated and returned.

If the node set `bip_set` is the same as the constrained one, the Λ-motifs follow a Binomial probability distribution. Otherwise, all the node pairs follow the same Poisson Binomial distribution.

The probability mass function is given by

$$pmf(k) = Pr(X = k)$$

Note: The lower triangular part including the diagonal is set to 0 since the matrix is symmetric.

Parameters

- **plam_mat** (*numpy.array*) – matrix of Lambda motif probabilities
- **nlam_mat** (*numpy.array*) – matrix of observed number of Lambda motifs
- **bip_set** (*bool*) – select row-nodes (`True`) or column-nodes (`False`)

Returns matrix containing the probabilities of the Λ-motifs

Return type numpy.array

Raises

- **NameError** – raise an error if the parameter `bip_set` is neither `True` nor `False`
- **AssertionError** – raise an error if shapes of the probability matrix and the matrix with the number of Λ-motifs are not equal

lambda_loglike (*bip_set=False*)
Return the log-likelihood of the number of Λ-motifs.

The total log-likelihood of the number of observed Λ-motifs in the input matrix is calculated according to the BiPCM null model.

Parameters **bip_set** (*bool*) – analyze Λ-motifs of row-nodes (`True`) or column-nodes (`False`)

lambda_motifs_main (*bip_set=False*, *write=True*, *filename=None*, *delim='t'*)
Calculate and save the p-values of the Λ-motifs.

For each node couple in the bipartite layer specified by `bip_set`, Λ-motifs and calculate the corresponding p-value.

Parameters

- **bip_set** (*bool*) – select row-nodes (`True`) or column-nodes (`False`)
- **write** (*bool*) – if `True`, the p-values are saved in the specified file
- **filename** (*str*) – name of the file which will contain the p-values, default is *pval_constr_<constraint>_proj_<rows OR columns>.csv*
- **delim** (*str*) – delimiter between entries in file, default is tab

Returns matrix of p-values if `write==True`

Return type numpy.array

Raises NameError raise an error if the parameter `bip_set` is neither `True` nor `False`

static save_matrix(*mat*, *filename*, *delim='t'*, *binary=False*)
Save the matrix `mat` in the file `filename`.

The matrix can either be saved as a binary NumPy `.npy` file or as a human-readable CSV file.

Note: The relative path has to be provided in the filename, e.g. *../data/pvalue_matrix.csv*

Parameters

- **mat** (*numpy.array*) – two-dimensional matrix
- **filename** (*str*) – name of the output file
- **delim** (*str*) – delimiter between values in file
- **binary** (*bool*) – if `True`, save as binary `.npy`, otherwise as a CSV file

set_degree_seq()
Return the node degree sequence of the input matrix.

Returns node degree sequence [degrees row-nodes, degrees column-nodes]

Return type numpy.array

Raises AssertionError raise an error if the length of the returned degree sequence does not correspond to the total number of nodes

static triumat2flat_idx(*i*, *j*, *n*)
Convert an matrix index couple to a flattened array index.

Given a square matrix of dimension n and an index couple (i, j) *of the upper triangular part* of the matrix, the function returns the index which the matrix element would have in a flattened array.

Note:

•$i \in [0, ..., n-1]$

•$j \in [i+1, ..., n-1]$

•returned index $\in [0,\ n(n-1)/2-1]$

Parameters

- **i** (*int*) – row index
- **j** (*int*) – column index
- **n** (*int*) – dimension of the square matrix

Returns flattened array index

Return type int

2.6 License

2.7 Contact

For questions or input, please write to mika.straka@imtlucca.it.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

[Saracco2015] F. Saracco, R. Di Clemente, A. Gabrielli, T. Squartini, Randomizing bipartite networks: the case of the World Trade Web, Scientific Reports 5, 10595 (2015)

[Saracco2016] F. Saracco, M. J. Straka, R. Di Clemente, A. Gabrielli, G. Caldarelli, T. Squartini, Inferring monopartite projections of bipartite networks: an entropy-based approach, arXiv preprint arXiv:1607.02481

[Straka2016] M. J. Straka, F. Saracco, G. Caldarelli, Product Similarities in International Trade from Entropy-based Null Models, Complex Networks 2016, 130-132 (11 2016), ISBN 978-2-9557050-1-8

B

C

G

L

S

T