Package ‘BiRewire’

January 30, 2017

Version 3.4.0
Date 2016-03-20
Title High-performing routines for the randomization of a bipartite graph (or a binary event matrix), undirected and directed signed graph preserving degree distribution (or marginal totals)
Maintainer Andrea Gobbi <gobbi.andrea@mail.com>
Description Fast functions for bipartite network rewiring through N consecutive switching steps (See References) and for the computation of the minimal number of switching steps to be performed in order to maximise the dissimilarity with respect to the original network. Includes functions for the analysis of the introduced randomness across the switching steps and several other routines to analyse the resulting networks and their natural projections. Extension to undirected networks and directed signed networks is also provided. Starting from version 1.9.7 a more precise bound (especially for small network) has been implemented. Starting from version 2.2.0 the analysis routine is more complete and a visual monitoring of the underlying Markov Chain has been implemented.
License GPL-3
Depends igraph, slam, tsne, Matrix
Suggests RUnit, BiocGenerics
Author Andrea Gobbi [aut], Francesco Iorio [aut], Giuseppe Jurman [cbt], Davide Albanese [cbt], Julio Saez-Rodriguez [cbt].
URL http://www.ebi.ac.uk/~iorio/BiRewire
biocView Network
NeedsCompilation yes

R topics documented:

- BiRewire-package ................................................................. 2
- birewire.analysis.bipartite .................................................. 3
- birewire.analysis.dsg .......................................................... 5
- birewire.analysis.undirected ................................................ 7
- birewire.bipartite.from.incidence ......................................... 9
- birewire.build.dsg .............................................................. 10
- birewire.induced.bipartite .................................................. 10
- birewire.load.dsg .............................................................. 11
- birewire.rewire.bipartite .................................................... 12
**BiRewire-package**

**Description**

R package for computationally-efficient rewiring of bipartite graphs (or randomisation of 0-1 tables with prescribed marginal totals), undirected and directed signed graphs (dsg). The package provides useful functions for the analysis and the randomisation of large biological datasets that can be encoded as 0-1 tables, hence modeled as bipartite graphs by considering a 0-1 table as an incidence matrix, and for data that can be encoded as directed signed graphs such as pathways and signaling networks. Large collections of such randomised tables can be used to approximate null models, preserving event-rates both across rows and columns, for statistical significance tests of combinatorial properties of the original dataset. The package provides an interface to a sampler routine useful for generating correctly such collections. Moreover a visual monitoring for the Markov Chain underlying the switching algorithm has been implemented.

**Details**

Summary:

<table>
<thead>
<tr>
<th>Package:</th>
<th>BiRewire</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version:</td>
<td>3.1.3</td>
</tr>
<tr>
<td>Date:</td>
<td>2016-03-20</td>
</tr>
<tr>
<td>Require:</td>
<td>slam, igraph, tsne, Matrix, R&gt;=2.10</td>
</tr>
<tr>
<td>URL:</td>
<td><a href="http://www.ebi.ac.uk/~iorio/BiRewire">http://www.ebi.ac.uk/~iorio/BiRewire</a></td>
</tr>
<tr>
<td>License:</td>
<td>GPL-3</td>
</tr>
</tbody>
</table>

**Author(s)**

Andrea Gobbi [aut], Davide Albanese [cbt], Francesco Iorio [cbt], Giuseppe Jurman [cbt].

Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
References


birewire.analysis.bipartite

Analysis of Jaccard similarity trends across switching steps.

Description

This function performs a sequence of max.iter switching steps on the input bipartite graph g and compute the Jaccard similarity between g (the initial network) and its rewired version each step switching steps. This procedure is performed n.networks times and a simple explorative plot, with mean and CI, is visualized if display is set to true.

Usage

birewire.analysis.bipartite(incidence, step=10, max.iter="n",accuracy=0.00005, verbose=TRUE,MAXITER_MUL=10,exact=FALSE,n.networks=50,display=TRUE)

Arguments

incidence Incidence matrix of the initial bipartite graph g (can be extracted from an igraph bipartite graph using the get.incidence function);

step 10 (default): the interval (in terms of switching steps) at which the Jaccard index between g and the its current rewired version is computed;

max.iter "n" (default) the number of switching steps to be performed (or if exact=TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References): \( N = e/(1 - d) \ln ((e - de)/\delta) \) if exact is FALSE, \( N = e(1 - d)/2 \ln ((e - de)/\delta) \) otherwise , where e is the number of edges of g and d its edge density . This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);

accuracy 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
birewire.analysis.bipartite

verbose TRUE (default). When TRUE a progression bar is printed during computation;

MAXITER_MUL 10 (default). If exact==TRUE in order to prevent a possible infinite loop the
program stops anyway after MAXITER_MUL*max.iter iterations;

exact FALSE (default). If TRUE the program performs max.iter switching steps, oth-
erwise the program will count also the not-performed switching steps;

n.networks 50 (default), the number of independent rewiring process starting from the same
initial graph from which the mean value and the CI is computed.

display TRUE (default). If TRUE two explorative plots are displayed summarizing the
trend of the Jaccard index in terms of mean and confidence interval.

Details

This function performs max.iter switching steps (see references). In particular, at each step two
edges are randomly selected from the current version of g. Let these two edges be (a, b) and (c, d)
(where a and c belong to the first class of nodes whereas b and d belong to the second one), with
a ≠ c and b ≠ d.

If the (a, d) and (c, b) edges are not already present in the current current version of g then (a, d)
and (c, b) replace (a, b) and (c, d).

At each step number of switching steps the function computes the Jaccard index between the
original graph g and its current version.

This procedure is performed n.networks times and if display is set to TRUE, two explorative plots
showing the mean value of the Jaccard Index over the SS and its CI are displayed.

Value

A list containing a data.frame data collecting all the Jacard index computed (each row is a run
of the SA), and the analytically derived lower bound N of switching steps to be performed by the
switching algorithm in order to provide the revired version of g with the maximal level of achievable
randomness (in terms of dissimilarity from the initial g).

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
Special thanks to:
Davide Albanese

References

Gobbi, A. and Iorio, F. and Dawson, K. J. and Wedge, D. C. and Tamborero, D. and Alexandrov,
randomization of large genomic datasets while preserving alteration counts Bioinformatics 2014

Jaccard, P. (1901), Étude comparative de la distribution florale dans une portion des Alpes et des

random graphs with prescribed degree sequences, eprint arXiv:cond-mat/0312028
Examples

```r
library(BiRewire)
g <- graph.bipartite(rep(0:1,length=10), c(1:10))

## get the incidence matrix of g
m<as.matrix(get.incidence(graph=g))

## set parameters
step=1
max=100*length(E(g))

## perform two different analysis using two different maximal number of switching steps
scores<-birewire.analysis.bipartite(m,step,max,n.networks=10)
scores2<-birewire.analysis.bipartite(m,step,"n",n.networks=10)
```

---

**birewire.analysis.dsg Analysis of Jaccard similarity trends across switching steps.**

**Description**

This function performs a sequence of `max.iter.pos` (and `max.iter.pos`) switching steps on the positive (and negative) part of the input dsg `g` and computes the Jaccard similarity between `g` (the initial network) and its rewired version each `step` switching steps. This procedure is performed `n.networks` times and a simple explorative plot, with mean and CI, is visualized if `display` is set to `TRUE`. The plot shows the trend of the Jaccard Index relative to the positive (and negative) part of `g`.

**Usage**

```r
birewire.analysis.dsg(dsg, step=10, max.iter.pos='n',max.iter.neg='n',accuracy=0.00005,
verbose=TRUE,MAXITER_MUL=10,exact=FALSE,n.networks=50,display=TRUE)
```

**Arguments**

- `dsg` The initial dsg object (see `birewire.induced.bipartite`). Note that the dsg must contain a list of two incidence matrices and not `igraph` bipartite graphs.
- `step` 10 (default): the interval (in terms of switching steps) at which the Jaccard index between `g` and the its current rewired version is computed;
- `max.iter.pos` "n" (default) the number of switching steps to be performed (or if `exact=TRUE` the number of successful switching steps) for the positive part of `g`. See `birewire.rewire.bipartite` for more details;
- `max.iter.neg` "n" (default) the same of `max.iter.p` but relative to the negative part;
- `accuracy` 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- `verbose` TRUE (default). When TRUE a progression bar is printed during computation;
- `MAXITER_MUL` 10 (default). If `exact=TRUE` in order to prevent a possible infinite loop the program stops anyway after `MAXITER_MUL*max.iter` iterations;
exact  FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;

n.networks  50 (default), the number of independent rewiring process starting from the same initial graph from which the mean value and the CI is computed.

display  TRUE (default). If TRUE two explorative plots are displayed summarizing the trend of the Jaccard index in terms of mean and confidence interval.

Details

This procedure acts in the same way of \texttt{birewire.analysis.bipartite} but in the case of \texttt{dsg}. The similarity is measured using \texttt{birewire.similarity.dsg}.

Value

A list containing two lists: \texttt{data} that is a list collecting all the Jaccard index computed (each row is a run of the SA) for the positive and negative part, and a list with the analytically derived lower bounds \( N \) for the positive and negative part of \( g \).

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

References


Examples

```r
library(BiRewire)
data(test_dsg)
dsg <- birewire.induced.bipartite(test_dsg,sparse=FALSE)
a=birewire.analysis.dsg(dsg,verbose=FALSE,step=1,exact=TRUE,max.iter.pos=200,max.iter.neg=50)
```
Analysis of Jaccard similarity trends across switching steps.

Description

This function performs a sequence of max.iter switching steps on the input undirected graph g and compute the Jaccard similarity between g (the initial network) and its rewired version each step switching steps. This procedure is performed n.networks times and a simple explorative plot, with mean and CI, is visualized if display is set to TRUE.

Usage

birewire.analysis.undirected(adjacency, step=10, max.iter="n", accuracy=0.00005, verbose=TRUE, MAXITER_MUL=10, exact=FALSE, n.networks=50, display=TRUE)

Arguments

adjacency            Incidence matrix of the initial bipartite graph g (can be extracted from an igraph undirected graph using the get.adjacency function);
step                10 (default): the interval (in terms of switching steps) at which the Jaccard index between g and its current rewired version is computed;
max.iter             "n" (default): the number of switching steps to be performed (or if exact==TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References): N = e/(2d^3 - dd^2 + 2d + 2) ln ((e - de)/δ) if exact is FALSE, N = e(1 - d)/2 ln ((e - de)/δ) otherwise , where e is the number of edges of g and d its edge density . This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);
accuracy             0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
verbose              TRUE (default). When TRUE a progression bar is printed during computation;
MAXITER_MUL          10 (default). If exact==TRUE in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;
exact                FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;
n.networks            50 (default), the number of independent rewiring process starting from the same initial graph from which the mean value and the CI is computed.
display              TRUE (default). If TRUE two explorative plots are displayed summarizing the trend of the Jaccard index in terms of mean and confidence interval.

Details

This function performs max.iter switching steps (see references). In particular, at each step two edges are randomly selected from the current version of g. Let these two edges be (a, b) and (c, d), with a ≠ c, b ≠ d, a ≠ d, b ≠ c . If the (a, d) and (c, b) (or (a, d) and (b, d)) edges are not already present in the current version of g then (a, d) and (c, b) replace (a, b) and (c, d) (or (a, b) and (c, d) replace (a, c) and (b, d)). If both
of the configurations are allowed, then one of them is randomly selected.

At each step number of switching steps the function computes the Jaccard index between the original graph \( g \) and its current version. This procedure is performed \( n\text{.networks} \) times and if display is set to TRUE, two explorative plots showing the mean value of the Jaccard Index over the SS and its CI are displayed.

**Value**

A list containing a data.frame \( data \) collecting all the Jaccard index computed (each row is a run of the SA), and the analytically derived lower bound \( N \) of switching steps to be performed by the switching algorithm in order to provide the revised version of \( g \) with the maximal level of achievable randomness (in terms of dissimilarity from the initial \( g \)).

**Author(s)**

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
Special thanks to:
Davide Albanese

**References**


**Examples**

library(BiRewire)
g <- erdos.renyi.game(1000,0.1)
## get the incidence matrix of g
m<-as.matrix(get.adjacency(graph=g,sparse=FALSE))

## set parameters
step=1000
max=100*length(E(g))

## perform two different analysis using two different numbers of switching steps
scores<-birewire.analysis.undirected(m,step,max,n.networks=10,verbose=FALSE)
scores2<-birewire.analysis.undirected(m,step,"n",n.networks=10,verbose=FALSE)
birewire.bipartite.from.incidence

Converts an incidence matrix into a bipartite graph.

Description

This function creates an igraph bipartite graph from an incidence matrix.

Usage

birewire.bipartite.from.incidence(matrix, directed=FALSE)

Arguments

matrix    incidence matrix: an (n-by-m) binary matrix where rows correspond to vertices
           in the first class while columns correspond to vertices in the second one;
directed  Logical, if TRUE a directed graph is created.

Details

The function calls graph.incidence of package igraph. See igraph documentation for more
details.

Value

Bipartite igraph graph.

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

References

Csardi, G. and Nepusz, T (2006) The igraph software package for complex network research, Inter-

Examples

library(igraph)
library(BiRewire)
g <- graph.bipartite( rep(0:1,length=10), c(1:10))
## gets the incidence matrix of g
m<-as.matrix(get.incidence(graph=g))
## rewire the current graph
m2=birewire.rewire.bipartite(m,100)
# create the rewired bipartite graph
g2<-birewire.bipartite.from.incidence(m2,TRUE)
birewire.build.dsg  Transform a dsg object in a SIF file.

Description
The routine transforms the initial dsg (two bipartite graphs) into SIF dsg format.

Usage
birewire.build.dsg(dsg, delimitators=list(negative='-', positive='+'))

Arguments
- dsg: The dsg to be converted;
- delimitators: list(negative='-', positive='+')(default): a list with 'positive' and 'negative' names identifying the character encoding the relation;

Details
This function converts the dsg object into a SIF format that can be saved using birewire.write.dsg, an internal function, using the given delimitators for encoding the relations. It is the inverse function of birewire.induced.bipartite.

Value
A dsg in SIF format.

Examples
```
data(test_dsg)
dsg=birewire.induced.bipartite(test_dsg)
tmp=birewire.rewire.dsg(dsg, verbose=FALSE)
dsg2=birewire.build.dsg(tmp)
```

birewire.induced.bipartite  Transform a SIF data frame into a dsg object (a list of positive and negative incidence matrix).

Description
The routine transforms the initial dsg graph in SIF format into a dsg object made of two bipartite graphs: one for positive edges and the other for negative edges.

Usage
birewire.induced.bipartite(g, delimitators=list(negative='-', positive='+'), sparse=FALSE)
**Arguments**

- **g**
  - A dataframe in SIF format describing a dsg (for example the output of `birewire.load.dsg`);

- **delimitators**
  - list(negative='-',positive='+')(default): a list with `positive` and `negative` names identifying the character encoding the relation;

- **sparse**
  - FALSE (default): if TRUE the two bipartite graphs are saved as `igraph` bipartite graphs;

**Details**

This function extract the positive and negative part of `g` and create a dsg object that can be used for example in the rewiring algorithm. It is the inverse function of `birewire.build.dsg`.

**Value**

A list of two incidence matrix or bipartite `igraph` objects.

**References**


**Examples**

```r
data(test_dsg)
dsg=birewire.induced.bipartite(test_dsg)
```

---

**Description**

The routine reads a SIF file and return a R table.

**Usage**

`birewire.load.dsg(path)`

**Arguments**

- **path**
  - Path to the SIF file.

**Value**

A R table that can be transformed into a dsg using `birewire.induced.bipartite`
birewire.rewire.bipartite

Efficient rewiring of bipartite graphs

Description

Optimal implementation of the switching algorithm. It returns the rewired version of the initial bipartite graph or its incidence matrix.

Usage

birewire.rewire.bipartite(incidence, max.iter="n", accuracy=0.00005, verbose=TRUE, MAXITER_MUL=10, exact=FALSE)

Arguments

incidence Incidence matrix of the initial bipartite graph \( g \) (can be extracted from an igraph bipartite graph using the get.incidence function; or the entire bipartite igraph graph

max.iter "n" (default) the number of switching steps to be performed (or if exact=TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References): \( N = e/2(1-d)\ln((e-de)/\delta) \) if exact is FALSE, \( N = e(1-d)/2\ln((e-de)/\delta) \) otherwise , where \( e \) is the number of edges of \( g \) and \( d \) its edge density . This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);

accuracy 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;

verbose TRUE (default). When TRUE a progression bar is printed during computation.

MAXITER_MUL 10 (default). If exact=TRUE in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;

exact FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;

Details

Main function of the package. It performs at most max.iter switching steps producing a rewired version of an initial bipartite graph.

Value

Incidence matrix of the rewired graph or the igraph corresponding object depending on the input type.

Author(s)

Andrea Gobbi

Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
References


Examples

```r
library(igraph)
library(BiRewire)
g <- graph.bipartite( rep(0:1,length=10), c(1:10))

## gets the incidence matrix of g
m <- as.matrix(get.incidence(graph=g))

## rewiring
m2 <- birewire.rewire.bipartite(m,100*length(E(g)))
## creates the corresponding bipartite graph
g2 <- birewire.bipartite.from.incidence(m2,directed=TRUE)
```

**birewire.rewire.bipartite.and.projections**

Analysis and rewiring function processing a bipartite graphs and its two projections

**Description**

This function performs the same analysis of `birewire.analysis.bipartite` but additionally it provides in output a rewired version of the two networks resulting from the natural projections of the initial graph, together with the corresponding Jaccard index trends.

**Usage**

```r
birewire.rewire.bipartite.and.projections(graph, step=10, max.iter="n", accuracy=0.00005, verbose=TRUE, MAXITER_MUL=10)
```

**Arguments**

- `graph`: A bipartite graph g;
- `max.iter`: "n" (default) the number of successful switching steps to be performed. If equal to "n" then this number is considered equal to the analytically derived lower bound $N = e(1 - d)/2 \ln ((e - de)/\delta)$ presented in Gobbi et al. (see References);
step 10 (default): the interval (in terms of switching steps) at which the Jaccard index between $g$ and its current rewired version is computed;

accuracy 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;

verbose TRUE (default) boolean value. If TRUE print a processing bar during the rewiring algorithm.

MAXITER_MUL 10 (default). Since $N$ indicates the number of successful switching steps, in order to prevent a possible infinite loop the program stops anyway after MAX-ITER_MUL*max.iter iterations;

Details
See birewire.analysis.bipartite for details.

Value
A list containing the three sequences of Jaccard index values (similarity_scores, similarity_scores.proj1, similarity_scores.proj2) for the three resulting graphs respectively (rewired, rewired.proj1, rewired.proj2). The first one is the rewired version of the initial graph $g$, while the second and the third one are rewired versions of its natural projections.

Author(s)
Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

References

Examples
library(igraph)
library(BiRewire)
g <- simplify(graph.bipartite( rep(0:1,length=100),
c(c(1:100),seq(1,100,3),seq(1,100,7),100,seq(1,100,13),
seq(1,100,17),seq(1,100,19),seq(1,100,23),100))
##gets the incidence matrix of g
m<-as.matrix(get.incidence(graph=g))
## rewires g and its projections
result=birewire.rewire.bipartite.and.projections(g,step=10,max.iter="n",accuracy=0.00005)
Efficient rewiring of directed signed graphs

Description
Optimal implementation of the switching algorithm. It returns the rewired version of the initial directed signed graph (dsg).

Usage
\begin{verbatim}
birewire.rewire.dsg(dsg,exact=FALSE,verbose=1,max.iter.pos='n',max.iter.neg='n',accuracy=0.00005,MAXITER_MUL=10,path=NULL,delimitators=list(positive='+',negative='-'))
\end{verbatim}

Arguments
- **dsg**: A dsg object: is a list of two incidence matrices (see References), "positive" and "negative", encoding the positive edges and negative edges. This list can be obtained reading a SIF file using \texttt{birewire.load.dsg} function and converting the resulting dataframe using \texttt{birewire.induced.bipartite};
- **exact**: FALSE (default). If TRUE the program performs \texttt{max.iter} successful switching steps, otherwise the program will count also the not-performed switching steps;
- **verbose**: TRUE (default). When TRUE a progression bar is printed during computation;
- **max.iter.pos**: "n" (default) the number of switching steps to be performed on the positive part of \texttt{dsg} (or if \texttt{exact}=TRUE the number of \text{\textbf{successful}} switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi \textit{et al.} (see References): \[ N = \frac{e}{2(1 - d)} \ln \left( \frac{(e - de)}{\delta} \right) \] if exact is FALSE, \[ N = \frac{e(1 - d)}{2} \ln \left( \frac{(e - de)}{\delta} \right) \] otherwise, where \( e \) is the number of edges of \( g \) and \( d \) its edge density. This bound is much lower than the empirical one proposed in Milo \textit{et al.} 2003 (see References);
- **max.iter.neg**: "n" (default) the number of switching steps to be performed on the negative part of \texttt{dsg} (or if \texttt{exact}=TRUE the number of \text{\textbf{successful}} switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi \textit{et al.} (see References): \[ N = \frac{e}{2(1 - d)} \ln \left( \frac{(e - de)}{\delta} \right) \] if exact is FALSE, \[ N = \frac{e(1 - d)}{2} \ln \left( \frac{(e - de)}{\delta} \right) \] otherwise, where \( e \) is the number of edges of \( g \) and \( d \) its edge density. This bound is much lower than the empirical one proposed in Milo \textit{et al.} 2003 (see References);
- **accuracy**: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- **MAXITER_MUL**: 10 (default). If \texttt{exact=TRUE} in order to prevent a possible infinite loop the program stops anyway after \texttt{MAXITER_MUL}*\texttt{max.iter} iterations;
- **path**: NULL (default). If not NULL, the dsg is saved in \texttt{path} in SIF format;
- **delimitators**: list(positive='+',negative='-') (default). If \texttt{save.file} is true, the dsg is saved using \texttt{delimitators} as characters encoding the relations. See \texttt{birewire.build.dsg} for more details.

Details
This function runs \texttt{birewire.rewire.bipartite} on the positive and negative part of \texttt{dsg}. See references for more details.
birewire.rewire.undirected

Value

Rewired dsg.

Author(s)

Andrea Gobbi: <gobbi.andrea@mail.com>

References


Examples

library(BiRewire)
data(test_dsg)
dsg=birewire.induced.bipartite(test_dsg)
tmp= birewire.rewire.dsg(dsg,verbose=FALSE)

Description

Optimal implementation of the switching algorithm. It returns the rewired version of the initial undirected graph or its adjacency matrix.

Usage

birewire.rewire.undirected(adjacency, max.iter="n",accuracy=0.00005, verbose=TRUE,MAXITER_MUL=10,exact=FALSE)

Arguments

adjacency An igraph undirected graph g or its adjacency matrix (can be extracted from g using get.adjacency);
max.iter  "n" (default) the number of switching steps to be performed (or if exact==TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References): \[ N = \frac{e}{(2d^3 - 6d^2 + 2d + 2)} \ln(e - de) \] if exact is FALSE, \[ N = \frac{e(1 - d)}{2 \ln((e - de)/\delta)} \] otherwise , where \(e\) is the number of edges of \(g\) and \(d\) its edge density. This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);

accuracy 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;

verbose TRUE (default) boolean value. If TRUE print a processing bar during the rewiring algorithm.

MAXITER_MUL 10 (default). If exact==TRUE in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;

exact FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;

Details
Performs at most max.iter number of rewiring steps producing a rewired version of an initial undirected graph.

Value
Adjacency matrix of the rewired graph or the relative igraph object depending on the input type.

Author(s)
Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
Special thanks to: Davide Albanese

References

Examples

library(igraph)
library(BiRewire)
g <- erdos.renyi.game(1000, 0.1)
##gets the incidence matrix of g
m<-as.matrix(get.adjacency(graph=g,sparse=FALSE))

## sets parameters
step=1000
max=100*length(E(g))

##rewiring
m2=birewire.rewire.undirected(m,100*length(E(g)))
##creates the corresponding bipartite graph
g2<graph.adjacency(m2,mode="undirected")

---

birewire.sampler.bipartite

Efficient generation of a null model for a given bipartite graph

Description

The routine samples correctly from the null model of a given bipartite graph creating a set of randomized version of the initial bipartite graph.

Usage

birewire.sampler.bipartite(incidence,K,path,max.iter="n", accuracy=0.00005,
verbose=TRUE,MAXITER_MUL=10,exact=FALSE,write.sparse=TRUE)

Arguments

- **incidence**: Incidence matrix of the initial bipartite graph;
- **K**: The number of networks that has to be generated;
- **path**: The directory in which the routine stores the outputs;
- **max.iter**: "n" (default) the number of switching steps to be performed (or if `exact=TRUE` the number of **successful** switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in *Gobbi et al.* (see References): \[N = \frac{e}{2(1-d)} \ln \left( \frac{(e-dc)/\delta}{e} \right)\] if exact is FALSE, \[N = \frac{e(1-d)/2 \ln \left( \frac{(e-dc)/\delta}{e} \right)}{\delta}\] otherwise, where \(e\) is the number of edges of \(g\) and \(d\) its edge density. This bound is much lower than the empirical one proposed in *Milo et al. 2003* (see References);
- **accuracy**: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- **verbose**: TRUE (default). When TRUE a progression bar is printed during computation.
- **MAXITER_MUL**: 10 (default). If `exact=TRUE` in order to prevent a possible infinite loop the program stops anyway after `MAXITER_MUL`*`max.iter` iterations;
- **exact**: FALSE (default). If TRUE the program performs `max.iter` switching steps, otherwise the program will count also the not-performed switching steps;
- **write.sparse**: TRUE (default). If FALSE the table is written as an R data.frame (long time and more space needed)
Details

The routine creates, starting from the given path, different subfolders in order to have maximum 1000 files for folder. Moreover the incidence matrices are saved using `write_stm_CLUTO` (sparse matrices) that can be loaded using `read_stm_CLUTO`. The set is generated calling `birewire.rewire.bipartite` on the last generated graph starting from the input graph.

Author(s)

Andrea Gobbi: <gobbi.andrea@mail.com>

References


description

Efficient generation of a null model for a given dsg. The routine samples correctly from the null model of a given dsg creating a set of randomized dsgs.

Usage

`birewire.sampler.dsg(dsg,K,path,delimitators=list(negative='-',positive='+'),exact=FALSE, verbose=TRUE, max.iter.pos='n',max.iter.neg='n', accuracy=0.00005,MAXITER_MUL=10)`

Arguments

dsg    A dsg object: is a list of two incidence matrices (see References), "positive" and "negative", encoding the positive edges and negative edges. This list can be obtained reading a SIF file using `birewire.load.dsg` function and converting the resulting dataframe using `birewire.induced.bipartite`.

max.iter.pos    "n" (default) the number of switching steps to be performed on the positive part of dsg (or if exact==TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References): $N = e/(1-d) \ln ((e - de)/\delta)$ if exact is FALSE, $N = e(1 - d)/2 \ln ((e - de)/\delta)$ otherwise , where e is the number of edges of g and d its edge density . This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);
max.iter.neg   "n" (default) the number of switching steps to be performed on the negative part of dsg (or if exact==TRUE the number of successful switching steps). If equal to "n" then this number is considered equal to the analytically derived lower bound presented in Gobbi et al. (see References):  \( N = \frac{e}{2(1 - d)} \ln \left( \frac{(e - de)}{\delta} \right) \) if exact is FALSE,  \( N = \frac{e(1 - d)}{2} \ln \left( \frac{(e - de)}{\delta} \right) \) otherwise , where \( e \) is the number of edges of g and \( d \) its edge density . This bound is much lower than the empirical one proposed in Milo et al. 2003 (see References);

accuracy     0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;

verbose      TRUE (default). When TRUE a progression bar is printed during computation.

MAXITER_MUL  10 (default). If exact==TRUE in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;

exact        FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;

path         The directory in which the routine stores the outputs;

K             The number of network that has to be generated;

delimitators list(negative='-',positive='+')(default):a list with 'positive' and 'negative' names identifying the character encoding the relation used for writing the output with birewire.build.dsg;

Details

The routine creates, starting from a given path, different subfolders in order to have maximum 1000 files for folder; the SIF files are saved using birewire.write.dsg, an internal routine. The set is generated calling birewire.rewire.dsg on the last generated dsg starting from the input dsg.

Author(s)

Andrea Gobbi: <gobbi.andrea@mail.com>

References


Efficient generation of a null model for a given undirected graph

**Description**

The routine samples correctly from the null model of a given undirected graph creating a set of randomized version of the initial undirected graph.

**Usage**

```r
birewire.sampler.undirected(adjacency,K,path,max.iter="n", accuracy=0.00005, verbose=TRUE,MAXITER_MUL=10,exact=FALSE,write.sparse=TRUE)
```

**Arguments**

- `adjacency`: Adjacency matrix of the initial undirected graph;
- `K`: The number of networks that has to be generated;
- `path`: The directory in which the routine stores the outputs;
- `max.iter`: "n" (default) see `birewire.rewire.undirected` for references
- `accuracy`: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- `verbose`: TRUE (default). When TRUE a progression bar is printed during computation.
- `MAXITER_MUL`: 10 (default). If `exact=TRUE` in order to prevent a possible infinite loop the program stops anyway after `MAXITER_MUL*max.iter` iterations;
- `exact`: FALSE (default). If TRUE the program performs `max.iter` switching steps, otherwise the program will count also the not-performed switching steps;
- `write.sparse`: TRUE (default). If FALSE the table is written as an R data.frame (long time and more space needed)

**Details**

The routine creates, starting from the given path, different subfolders in order to have maximum 1000 files for folder. Moreover the incidence matrices are saved using `write_stm_CLUTO` (sparse matrices) that can be loaded using `read_stm_CLUTO`. The set is generated calling `birewire.rewire.undirected` on the last generated graph starting from the input graph.

**Author(s)**

Andrea Gobbi: <gobbi.andrea@mail.com>
References


---

birewire.similarity

Compute the Jaccard similarity index between two binary matrices with the same number of non-null entries and the same row- and column-wise sums.

**Description**

Compute the Jaccard similarity index between two binary matrices with the same number of non-null entries and the same row- and column-wise sums. The function accepts also two igraph objects.

**Usage**

```r
birewire.similarity(m1, m2)
```

**Arguments**

- `m1`: First matrix or graph;
- `m2`: Second matrix or graph.

**Details**

The **Jaccard** index between two sets $M$ and $N$ is defined as:

$$\frac{|M \cup N|}{|M \cap N|}$$

With $M$ and $N$ binary matrices, the Jaccard index is computed as:

$$\frac{\sum_i N_{i,j} \land M_{i,j}}{\sum_i N_{i,j} \lor M_{i,j}}.$$ 

The Jaccard index ranges between 0 and 1.

**Value**

Returns the Jaccard similarity index between the objects.

**Author(s)**

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>
Examples

```r
library(igraph)
library(BiRewire)
g <- graph.bipartite(rep(0:1, length=10), c(1:10))
g2 = birewire.rewire.bipartite(g)
birewire.similarity(get.incidence(g, sparse=FALSE), get.incidence(g2, sparse=FALSE))
birewire.similarity(g, g2)
```

---

**birewire.similarity.dsg**

*Compute the Jaccard similarity index between dsg.*

**Description**

Compute the Jaccard similarity index between dsg objects described in the same way (matrices of graphs).

**Usage**

```r
birewire.similarity.dsg(m1, m2)
```

**Arguments**

- `m1`: First dsg;
- `m2`: Second dsg.

**Details**

See `birewire.similarity` for more details.

**Value**

Returns the Jaccard similarity index between the objects.

**Author(s)**

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

**Examples**

```r
library(BiRewire)
data(test_dsg)
dsg <- birewire.induced.bipartite(test_dsg, sparse=FALSE)
birewire.similarity.dsg(dsg, birewire.rewire.dsg(dsg))
dsg <- birewire.induced.bipartite(test_dsg, sparse=TRUE)
birewire.similarity.dsg(dsg, birewire.rewire.dsg(dsg))
```
birewire.slum.to.sparseMatrix

The function transforms a triplet sparse matrix from slum package to a Matrix sparse matrix.

Description

Transform a triplet sparse matrix from slum package to a Matrix sparse matrix that can be used by igraph for creating a network. This function could be used in order to analyze graphs obtained from samplers routines (birewire.sampler.undirected, birewire.sampler.dsg and birewire.sampler.bipartite).

Usage

birewire.slum.to.sparseMatrix( simple_triplet_matrix_sparse)

Arguments

simple_triplet_matrix_sparse
A triplet sparse matrix, usually the object coming from read_stm_CLUTO.

Value

Returns an Matrix sparse matrix that could be used for building an igraph graph using graph.adjacency.

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

birewire.visual.monitoring.bipartite

Visual monitoring of the Markov chain underlying the SA for directed graphs.

Description

This function generates a cascade-sampling from the model at different switching steps given in sequence. For each step the routine computes the pairwise Jaccard distance (1-JI) among the samples and performs, on the resulting matrix, a dimensional scaling reduction (using tsne). If display is set to TRUE the relative plot is displayed.

Usage

birewire.visual.monitoring.bipartite(data, accuracy=0.00005, verbose=FALSE, MAXITER_MUL=10, exact=FALSE, n.networks=100, perplexity=15, sequence=c(1, 5, 100, "n"), ncol=2, nrow=length(sequence)/ncol, display=TRUE)
Arguments

- **data**: The initial bipartite graph, either an incidence matrix or an `igraph` bipartite graph object;
- **accuracy**: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- **verbose**: TRUE (default). When TRUE a progression bar is printed during computation.
- **MAXITER_MUL**: 10 (default). If `exact==TRUE` in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;
- **exact**: FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;
- **n.networks**: 100 (default): the number of network generated for each step defined in sequence;
- **perplexity**: 15 (default): the value of perplexity passed to the function `tsne`;
- **sequence**: `c(1,5,100,"n")` (default) the sequence of step for which generating a sampler (see `birewire.sampler.bipartite`);
- **ncol**: 2 (default). The number of column in the plot;
- **nrow**: length(sequence)/ncol (default). The number of row in the plot;
- **display**: TRUE (default). If TRUE the result is displayed.

Details

For each value \( p \) in sequence (it that can also contain the special character "n", see `birewire.rewire.bipartite`), the routine generates n.networks sampled each \( p \) SS from the SA initialized with the given data. Pairwise distance are computed using the Jaccard distance and the resulting matrix is the input for the dimensional scaling performed by the function `tsne`. An explorative plot is displayed if `display` is set to TRUE.

Value

A list containing the list containing the distance matrices `dist` and the list containing the tsne results `tsne`.

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

References


**Examples**

```r
library(BiRewire)
g <- graph.bipartite( rep(0:1,length=100), c(1:100))
birewire.visual.monitoring.bipartite(g,display=FALSE,n.networks=10)
```

**Description**

This function generates a cascade-sampling from the model at different switching steps given in `sequence`. For each step the routine computes the pairwise Jaccard distance (1-JI) among the samples and performs, on the resulting matrix, a dimensional scaling reduction (using `tsne`). If `display` is set to `TRUE` the relative plot is displayed.

**Usage**

```r
birewire.visual.monitoring.dsg(data,accuracy=0.00005,verbose=FALSE,MAXITER_MUL=10,exact=FALSE,n.networks=100,perplexity=15,
sequence.pos=c(1,5,100,"n"),
sequence.neg=c(1,5,100,"n"),ncol=2,nrow=length(sequence.pos)/ncol,display=TRUE)
```

**Arguments**

- **data**: The initial dsg either in matrix or graph formulation (see `birewire.induced.bipartite`).
- **accuracy**: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- **verbose**: TRUE (default). When TRUE a progression bar is printed during computation.
- **MAXITER_MUL**: 10 (default). If `exact==TRUE` in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;
- **exact**: FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed switching steps;
- **n.networks**: 100 (default): the number of network generated for each step defined in `sequence`;
- **perplexity**: 15 (default): the value of perplexity passed to the function `tsne`;
- **sequence.pos**: `c(1,5,100,"n")`(default) the sequence of step for which generating a sampler (see `birewire.sampler.dsg`) for the positive part of `data`
- **sequence.neg**: same as `sequence.pos` but for the negative part
- **ncol**: 2 (default). The number of column in the plot;
- **nrow**: length(sequence)/ncol (default). The number of row in the plot;
- **display**: TRUE (default). If TRUE the result of `tsne` is displayed.
Details

See `birewire.visual.monitoring.bipartite` for more details.

Value

A list containing the list containing the distance matrices `dist` and the list containing the tsne results `tsne`.

Author(s)

Andrea Gobbi
Maintainer: Andrea Gobbi <gobbi.andrea@mail.com>

References


Examples

```r
library(BiRewire)
data(test_dsg)
##bigger dsg
test_dsg_2=test_dsg
test_dsg_2[,1]=paste(test_dsg_2[,1],"_",sep="")
test_dsg_2[,3]=paste(test_dsg_2[,3],"_",sep="")
dsg <- birewire.induced.bipartite(rbind(test_dsg,test_dsg_2),sparse=FALSE)

a=birewire.visual.monitoring.dsg(dsg,exact=TRUE,sequence.pos=c(1,2,"n",100),sequence.neg=c(1,2,"n",60),n.networks=50)
```
**Description**

This function generates a cascade-sampling from the model at different switching steps given in sequence. For each step the routine computes the pairwise Jaccard distance (1-JI) among the samples and performs, on the resulting matrix, a dimensional scaling reduction (using tsne). If display is set to TRUE the relative plot is displayed.

**Usage**

```r
birewire.visual.monitoring.undirected(data, accuracy=0.00005, verbose=FALSE, MAXITER_MUL=10, exact=FALSE, n.networks=100, perplexity=15, sequence=c(1,5,100,"n"), ncol=2, nrow=length(sequence)/ncol, display=TRUE)
```

**Arguments**

- **data**: The initial undirected graph, either an adjacency matrix or an igraph undirected graph object;
- **accuracy**: 0.00005 (default) is the desired level of accuracy reflecting the average distance between the Jaccard index at the N-th step and its analytically derived fixed point in terms of fraction of common edges;
- **verbose**: TRUE (default). When TRUE a progression bar is printed during computation.
- **MAXITER_MUL**: 10 (default). If exact==TRUE in order to prevent a possible infinite loop the program stops anyway after MAXITER_MUL*max.iter iterations;
- **exact**: FALSE (default). If TRUE the program performs max.iter switching steps, otherwise the program will count also the not-performed swithcing steps;
- **n.networks**: 100 (default): the number of network generated for each step defined in sequence;
- **perplexity**: 15 (default): the value of perplexity passed to the function tsne;
- **sequence**: c(1,5,100,"n") (default) the sequence of step for wich generating a sampler (see birewire.sampler.undirected);
- **ncol**: 2 (default). The number of column in the plot;
- **nrow**: length(sequence)/ncol (default). The number of row in the plot;
- **display**: TRUE (default). If TRUE the result of tsne is displayed.

**Details**

For each value p in sequence (it that can also contain the special character "n", see birewire.rewire.bipartite), the routine generates n.networks sampled each p SS from the SA initialized with the given data. Parwise distance are computed using the Jaccard distance and the resulting matrix is the input for the dimensional scaling performed by the function tsne. An explorative plot is displayed if display is set to TRUE.
**BRCA_binary_matrix**

**Value**

A list containing the list containing the distance matrices *dist* and the list containing the tsne results *tsne*.

**Author(s)**

Andrea Gobbi
Maintainer: Andrea Gobbi &lt;gobbi.andrea@mail.com&gt;

**References**


**Examples**

```r
library(BiRewire)
g <- erdos.renyi.game(1000,0.1)
birewire.visual.monitoring.undirected(g,display=FALSE,n.networks=10)
```

---

**BRCA_binary_matrix**  
**TCGA Brest Cancer data**

**Description**

Breast cancer samples and their respective mutations downloaded from the Cancer Cancer Genome Atlas (TCGA), used in *Gobbi et al.* Germline mutations were filtered out of the list of reported mutations; synonymous mutations and mutations identified as benign and tolerated were also removed from the dataset. The bipartite graph resulting when considering this matrix as an incidence matrix has \( n_r = 757, n_c = 9757, e = 19758 \) for an edge density equal to 0.27%.

**Usage**

```r
data(BRCA_binary_matrix)
```

**Source**

http://tcga.cancer.gov/dataportal/
References


---

test_dsg  Tool example of dsg

Description

A simple dsg for testing routines.

Usage

data(test_dsg)
Index

*Topic **bipartite graph, incidence matrix**
  - birewire.bipartite.from.incidence, 9

*Topic **bipartite graph, projection, rewire**
  - birewire.rewire.bipartite.and.projections, birewire.induced.bipartite, 5, 10, 11, 15, 19, 26

*Topic **bipartite graph, rewire**
  - birewire.analysis.bipartite, 3, 6, 13, 14
  - birewire.load.dsg, 11, 15, 19

*Topic **birewire.analysis.dsg**, 5
  - birewire.analysis.undirected, 12, 15, 25

*Topic **birewire.load.dsg**, 11
  - birewire.rewire.bipartite, 5, 12, 15, 25, 28

*Topic **birewire.sampler.bipartite**, 18
  - birewire.rewire.bipartite, 12

*Topic **birewire.slum.to.sparseMatrix**, 24
  - birewire.similarity, 22

*Topic **birewire.similarity.dsg**, 6
  - birewire.slum.to.sparseMatrix, 24

*Topic **birewire.visual.monitoring.dsg**, 26
  - birewire.similarity.dsg, 6, 23

*Topic **birewire.visual.monitoring.undirected**, 28
  - birewire.similarity.dsg, 23

*Topic **BRCA_binary_matrix**, 29
  - birewire.slum.to.sparseMatrix, 24

*Topic **datasets**
  - BRCA_binary_matrix, 29
  - test_dsg, 30

*Topic **directed graph, rewire, pathway, signaling**
  - birewire.build.dsg, 10
  - birewire.induced.bipartite, 10
  - birewire.load.dsg, 11
  - birewire.rewire.dsg, 15

*Topic **igraph**, 3, 5, 7, 9, 11, 12, 16, 24
  - birewire.rewire.undirected, 16

*Topic **Birewire**
  - BiRewire-package, 2
  - BiRewire-package, 2

BiRewire (BiRewire-package), 2