BiRewire

Andrea Gobbi, Francesco Iorio

Contents

1 Overview

BiRewire is an R package implementing high-performing routines for the randomisation of bipartite graphs preserving their node degrees (i.e. Network Rewiring), through the Switching Algorithm (SA) [5].
This package is particularly useful for the randomisation of '0-1' tables (or presence-absence matrices) in which the distributions of non-null entries (i.e. presence distributions) must be preserved both across rows and columns. By considering these tables as incidence matrices of bipartite graphs then this problem reduces to bipartite network rewiring.
For example, by modeling a genomic dataset as a binary event matrix (BEM),

in which rows correspond to samples, columns correspond to genes and the (i, j) entry is non-null if the i-th sample harbours a mutation in the j-th gene, then with BiRewire is possible to randomise the dataset preserving its mutation rates both across samples and genes. This is crucial to preserve tumour specific alterations, dependencies between gene-mutations and heterogeneity in mutation/copy-number-alteration rates across patients.
Large collections of such randomised tables can be then used to approximate samples from the uniform distribution of all the possible genomic datasets with the same mutation-rates of the initial one. Finally this data can be used as null model to test the statistical significance of several combinatorial properties of the original dataset: for example the tendency of a group of genes to be co- or mutually-mutated [7].

Moreover, with se same routines, it is possible to generate a rewired version of a given directed signed network (DSG), encoding for example a patway or a signaling network (for details see section 5 and [2]). Similar procedures have been implemented in order to manage undirected networks. Since version 3.6.0, the SA can be performed also on matrices containing NAs. In this case the SA works as usual but the position of NA will be preserved. This feature is available if the graph is encoded whith its incidence/adjacence matrix and not in the case of DSG.

Specifically, with *BiRewire* users can:

1. create bipartite graphs from genomic BEMs (or, generally, from any kind of presence-absence matrix);

2. perform an analysis, which consists of studying the trend of Jaccard Similarity between the original network and its rewired versions across the switching steps (by using a user-defined sampling time), and analytically estimating the number of steps at which this similarity reaches a plateau (i.e. the maximal level of randomness is achieved) according to the lower bound derived in [1];

3. generate rewired versions of a bipartite graph with the analytically derived bound as number of switching steps or a user-defined one;

4. derive projections of the starting network and its rewired version and perform different graph-theory analysis on them;

5. generate a set of netwoks correctly drawn from the suitable null-model starting from the initial BEM;

6. monitoring the behaviour of the Markov chain underlying the SA

7. perform the same analysis described in point 1,2,3,5 and 6 for undirected graphs and directed signed graphs (DGS).

All the functions of the package are written in C-code and R-wrapped. A reduced version of the packages has been also implemented in Python `https://github.com/andreagobbi/pyBiRewire`.

2 Installation

It is possible to download the package from `http://www.ebi.ac.uk/~iorio/BiRewire` and install it with the shell-command:

```
R CMD INSTALL BiRewire_xx.yy.zz.tar.gz
```

or with BiocManager::install() directly in R:

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("BiRewire")
```

Moreover, the sources of the developement version are available here `http://www.bioconductor.org/packages/devel/bioc/html/BiRewire.html`. Alternatively, the source files can be cloned from the github repositories: `https://github.com/andreagobbi/BiRewire` and `https://github.com/andreagobbi/BiRewire--release` using the command

```
git clone git@github.com:andreagobbi/BiRewire--release.git
git clone git@github.com:andreagobbi/BiRewire.git
```

We suggest to use the BiocManager::install() function from R in order to have the last working relase package (build and check procedure).

To load BiRewire use the following commands:

```
> library(BiRewire)
```

3 Package Dependencies

BiRewire requires the R packages **Matrix igraph** [6], **slam** [11] and **Rtsne** [12] available at the CRAN repository.

4 Notation

Let $\mathcal{G}$ be a bipartite graph, i.e. a graph containing two classes of nodes V_r and V_c such that every edge $e \in E$ connects one node in the first class to a node in the second class.
Let $\mathcal{B}$ be the incidence matrix of $\mathcal{G}$, i.e. the $|V_r| \times |V_c|$ binary matrix whose generic entry $m_{i,j}$ is not null if an only if $(i, j) \in E$.
The number of edges is indicated with $e = |E|$ and the edge density with $d = \frac{e}{|V_r||V_c|}$.

The SA performs N Switching Steps (SSs), in which:

1. two edges (a, b) and (c, d) both $\in E$ are randomly selected,
2. if $a \neq c$, $b \neq d$, $(a, d) \notin E$ and $(b, d) \notin E$ then:
   (a) the edges (a, d) and (b, d) are added to E and
   (b) the edges (a, b) and (c, d) are removed from E.

Notice that we count a SS only if it is successfully performed.

The *Jaccard Index* (JI, [10]) is used to quantify the similarity between the original graph and its rewired version at the k-th SS. Since the SA preserves the degree distribution and does not alter the number of nodes, the JI, indicated with $s^{(k)}$, can be computed as

$$s^{(k)} = \frac{x^{(k)}}{2e - x^{(k)}},$$

where $x^{(k)}$ is the number of edges in common between the two graphs. Fixed a small error δ, the number N of SSs providing the rewired version of a network with the maximally achievable level of randomness (in terms of average dissimilarity from the original network) is asymptotically equal to

$$\frac{e(1-d)}{2} \ln \frac{1-d}{\delta}.$$

More detailed, we analitically derived the fixed point of the underlying Markov chain $\bar{x}$; fixed a δ we can extimate the distance of the current state of the chain respect to this fixed point in terms of fraction of edges δ. For large netowrk we can assumne that a distance less than one edge is satisfiable (see [**?**]), so the bound reads:

$$\frac{e(1-d)}{2} \ln e(1-d),$$

but in order to manage smaller network the bound with the parameter δ results to be more general. This bound is much lower than the empirical one proposed in [5] (see Reference for details).

5 Directed Signed Network

A directed signed network (DSG) $\mathcal{G}$ is a directed network in which the edges are encoded with a triplet $(a, b, *)$ where a denotes the source node, b the target node and $*$ the sign of the relation (the sign of the edge). In our case (pathways and signaling) $*$ can be positive $+$ and negative $-$. In [2] we show how to create a correspondence between a DSG and a couple (B^+, B^-) of bipartite networks. This correspondence f, and its inverse f^{-1} are useful for the creation of a rewired version of G: we rewire independently B^+ and B^- and we rebuild the final DSG G^* using f^{-1}. A DGS is usually encoded in a SIF format (Simple Interaction File see `http://wiki.cytoscape.org/Cytoscape_User_Manual/Network_Formats` for more informations). In the case of DSG a suitable SIF file has 3 columns: the first encodes the source nodes, the second the sign and the last the source nodes.

6 Function Description

In this section all the functions implemented in BiRewire are described with a simple practical example in which a real breast cancer dataset is modeled as a bipartite network, and randomised preserving the mutation-rate both across samples and genes (i.e. the corresponding bipartite network is rewired). In each of the following functions it is possible to perform N **successful** switching

steps (see [1] for more details about this more general bound) using the flag exact=TRUE. To prevent a possible indinite loop, the program performs at maximum MAXITER_MUL*max.iter iterations.

6.1 birewire.analysis.bipartite and.undirected

First of all, we create a bipartite network modeling a genomic breast cancer dataset downloaded from the Cancer Genome Atlas (TCGA) projects data portal `http://tcga.cancer.gov/dataportal/`, used in [1] From this dataset germline mutations were filtered out with state-of-the-art softwares; synonymous mutations and mutations identified as benign and tolerated were also removed. The resulting bipartite graph has $n_r = 757$ nodes (corresponding to samples), $n_c = 9,757$ nodes (corresponding to genes), and $e = 19,758$ edges connecting a node in n_r to a node in n_c if the gene corresponding to the node in n_r is mutated to the samples corresponding to the node in n_C. The edge density of this network is 0.27%.

The genomic dataset (in the form of a binary matrix in which rows correspond to samples, columns correspond to genes and the (i, j) entry is non null if the i-th sample harbours a mutation in the j-th gene) can be loaded and modeled as a bipartite graph, with the following commands:

```
> data(BRCA_binary_matrix)##loads an binary genomic event matrix for the
>
> g=birewire.bipartite.from.incidence(BRCA_binary_matrix)##models the dataset
>
```

Once the bipartite graph is created it is possible to conduct the analysis by calling the **birewire.analysis.bipartite** function, using the following commands:

```
> step=5000
> max=100*sum(BRCA_binary_matrix)
> scores<-birewire.analysis.bipartite(BRCA_binary_matrix,step,
+                  verbose=FALSE,max.iter=max,n.networks=5,display=F)
>
```

The function **birewire.analysis.bipartite** returns the Jaccard similarity sampled every *step* SSs (in the example above step is equal to 5000). The SA is independentyl applyed on the initial data for *n.networks* times in order to extimate the mean value of thr JI and the relative CI (as $1.96 \pm \sigma/\sqrt{n.networks}$). A plot such information is displayed if the paramenter *ndisplay* is set to true. The routine returns a list of two element: **$N** is the analitically derived bound and **$data** the similarity score table.

The same analysis can be performed on general undirected networks.

```
> g.und<-erdos.renyi.game(directed=F,loops=F,n=1000,p.or.m=0.01)
> m.und<-get.adjacency(g.und,sparse=FALSE)
> step=100
> max=100*length(E(g.und))
> scores.und<-birewire.analysis.undirected(m.und,step=step,
+                          verbose=FALSE,max.iter=max,n.networks=5)
>
```

6.2 birewire.rewire.bipartite

To rewire a bipartite graph two modalities are available. Both of them can be used with the analytical bound N as number of switching steps or with a user defined value. The function takes in input an incidence matrix $\mathcal{B}$ or the an *igraph* bipartite graph.

```
> m2<-birewire.rewire.bipartite(BRCA_binary_matrix,verbose=FALSE)
> g2<-birewire.rewire.bipartite(g,verbose=FALSE)
```

The first function recives in output the incidence matrix of the rewired graph while the second one a bipartite *igraph* graph. See documentation for further details.

6.3 birewire.rewire.undirected

To rewire a general undirected graph the following functions can be used:

```
> m2.und<-birewire.rewire.undirected(m.und,verbose=FALSE)
> g2.und<-birewire.rewire.undirected(g.und,verbose=FALSE)
```

6.4 birewire.similarity

This function computes the Jaccard index between two incidence matrices with same dimensions and node degrees. It is possible also to use directly two suitable graphs.

```
> sc=birewire.similarity(BRCA_binary_matrix,m2)
> sc=birewire.similarity(BRCA_binary_matrix,t(m2))#also works
```

6.5 birewire.rewire.bipartite.and.projections

The following functions execute the Switching Algorithm and computes similarity trends across its switching steps for the two natural projections of the starting bipartite graph

```
> #use a smaller graph!
> gg <- graph.bipartite( rep(0:1,length=10), c(1:10))
> result=birewire.rewire.bipartite.and.projections(gg,step=10,
+         max.iter="n",accuracy=0.00005,verbose=FALSE)
> plot(result$similarity_scores.proj2,type='l',col='red',ylim=c(0,1))
> lines(result$similarity_scores.proj1,type='l',col='blue')
> legend("top",1, c("Proj2","Proj1"), cex=0.9, col=c("blue","red"), lty=1:1,lwd=3)
```

6.6 birewire.sampler.bipartite

This function uses the SA to generate a set of K bipartite networks drawn from the null model given by an initial bipartite graph. The function creates a main folder (*path* input parameter) and a set of subfolders in order to have maximum 1000 files per folder. Notice that the initial graph is used only for the first rewiring process, and the output of the fist process is used as inoput for the second and so on.

```
> #use a smaller graph!
> gg <-graph.bipartite(rep(0:1,length=10), c(1:10))
>  ## NOT RUN
>  ##birewire.sampler.bipartite(get.incidence(g),K=10,path='TESTBIREWIRE',verbose=F)
>  ##unlink('TESTBIREWIRE',recursive = T)
```

6.7 birewire.visual.monitoring.bipartite and .undirected

These functions allow to visualize the Markov Chain underlying the SA. More in detail, given a sequence of steps to test, we sample from the SA each indicated step generating a brunch of networks. We compute the pairwise Jaccard distance among them, i.e. the Jaccad index is defined as 1 minus the Jaccad similarity. Then we perform a dimensional scaling using *Rtsne* [12] and plot the result.

```
> ggg <- graph.bipartite( rep(0:1,length=10), c(1:10))
> tsne = birewire.visual.monitoring.bipartite(ggg,display=F,n.networks=10,perplexity=2)

[1] "K= 1"
[1] "K= 5"
[1] "K= 100"
[1] "K= n"

> g <- erdos.renyi.game(1000,0.1)
> tsne = birewire.visual.monitoring.undirected(g,display=F,n.networks=10,perplexity=2)

[1] "K= 1"
[1] "K= 5"
[1] "K= 100"
[1] "K= n"
```

6.8 birewire.load.dsg and birewire.save.dsg

The first function reads a SIF DSG from a given path and the second writes a DSG in a specific path.

6.9 birewire.induced.bipartite and birewire.build.dsg

These two functions encoded the correspondence between a DSG and a ordered couple of bipartite graphs (B^+, B^-). The first takes a SIF object, loaded with **birewire.load.dsg** and produces a list with the positive and negative bipartite graph, the second one build a SIF object starting from a list of two bipartite networks.

```
> data(test_dsg)
> dsg=birewire.induced.bipartite(test_dsg,delimitators=list(negative='-',positive='+'))
> tmp=birewire.build.dsg(dsg,delimitators=list(negative='-',positive='+'))
```

6.10 birewire.rewire.dsg

Function for generating a rewired version of a given DSG G. The parameters are quite the same of **birewire.rewire.bipartite**: in this case it is possible to control the number of SS independently for the positive**max.iter.pos** and negative **max.iter.neg** part of G.

```
> dsg2=birewire.rewire.dsg(dsg=dsg)

DONE in 0 seconds
DONE in 0 seconds

> tmp=birewire.build.dsg(dsg2,delimitators=list(negative='-',positive='+'))
```

6.11 birewire.similarity.dsg

Computes the Jaccard index between two DSGs.

```
>  birewire.similarity.dsg(dsg,dsg2)

[1] 0.2222222
```

6.12 birewire.sampler.dsg

This function uses the SA to generate a set of K DSG in SIF format drawn from the null model given by an initial DSG. The function creates a main folder (*path* input parameter) and a set of subfolders in order to have maximum 1000 files per folder. See **birewire.sampler.bipartite** from more details.

```
> ##NOT RUN
> ##birewire.sampler.dsg(dsg,K=10,path='TESTBIREWIREDSG',verbose=F,
> ##                      delimitators=list(negative='-',positive='+'))
> ##unlink('TESTBIREWIREDSG',recursive = T)
```

Since verison 3.27.1 it is possible to add a flag in order to not generate positive and negative loops between two nodes. Be aware that the process could be slower since the generated dsg is checked after the rewiring procedure and saved only if it does not contain positive-negative simulataneous loops.

```
> ##NOT RUN
> ##birewire.sampler.dsg(dsg,K=10,path='TESTBIREWIREDSG',verbose=F,
> ##                      delimitators=list(negative='-',positive='+'),check_pos_neg=T)
> ##unlink('TESTBIREWIREDSG',recursive = T)
```

7 Directed graphs

Notice that a directed graph can be encoded as a DSG with only positive (negative) part. All the routines involving DSG could be used also for directed graphs building a DSN as a R list with just one element named **positive**.

8 Example

Here we collect the functionalities of the package in a single example. The output plot of the analysis is showed in Fig.1 on the left side and the output of the monitoring procedure is displayed in Fig.1 on the right side.

```
> ##NOT RUN
> #ggg <- bipartite.random.game(n1=100,n2=40,p=0.2)
> #For recovering quickly the bound N we can perform a short analysis
> #N=birewire.analysis.bipartite(get.incidence(ggg,sparse=F),max.iter=2,step=1)$N
> #Now we can perform the real analysis
> #res=birewire.analysis.bipartite(get.incidence(ggg,sparse=F),max.iter=10*N,n.networks=10
> #and monitoring the markov chain
> #tsne = birewire.visual.monitoring.bipartite(ggg,display=T,n.networks=75,sequence=c(1,10
> #Now we can generate a null model
> #birewire.sampler.bipartite(ggg,K=10000,path="TESTBIREWIREBIPARTITE")
```

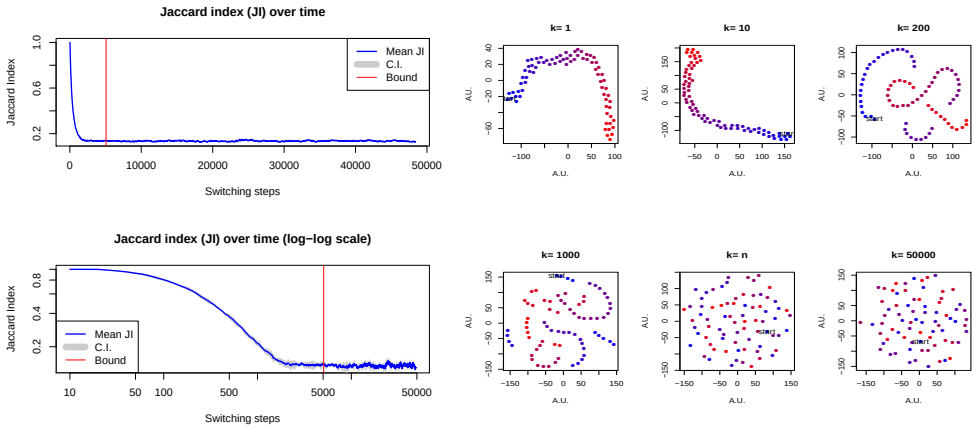

Figure 1: The output plots of **birewire.analyis.bipartite**(left side) and of **birewire.visual.monitoring.bipartite**(right side) relative to the example in section 8. The gradient of the colour from blue to red indicates the position of the sampling network respect the others. The starting network (blue) is marked with the text *start*.

References

[1] Gobbi, A. and Iorio, F. and Dawson, K. J. and Wedge, D. C. and Tamborero, D. and Alexandrov, L. B. and Lopez-Bigas, N. and Garnett, M. J. and Jurman, G. and Saez-Rodriguez, J. (2014) *Fast randomization of large genomic datasets while preserving alteration counts* Bioinformatics 2014 30 (17): i617-i623 doi: 10.1093/bioinformatics/btu474.

[2] Iorio, F. and and Bernardo-Faura, M. and Gobbi, A. and Cokelaer, T.and Jurman, G.and Saez-Rodriguez, J. (2016) *Efficient randomization of biologicalnetworks while preserving functionalcharacterization of individual nodes* Bioinformatics 2016 1 (17):542 doi: 10.1186/s12859-016-1402-1.

[3] Gobbi, A. and Jurman, G. (2013) *Theoretical and algorithmic solutions for null models in network theory* (Doctoral dissertation) `http://eprints-phd.biblio.unitn.it/1125/` .

[4] Jaccard, P. (1901), *Etude comparative de la distribution florale dans une portion des Alpes et des Jura*, Bulletin de la Societe Vaudoise des Sciences Naturelles 37:547–579.

[5] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, U. Alon (2003), *On the uniform generation of random graphs with prescribed degree sequences*, eprint arXiv:cond-mat/0312028.

[6] Csardi, G. and Nepusz, T (2006) *The igraph software package for complex network research*, InterJournal, Complex Systems `http://igraph.sf.net`.

[7] Ciriello, G. and Cerami, E. and Sander, C. and Schultz, N.(2012) Mutual exclusivity analysis identifies oncogenic network modules, *Genome Research*, **22**, 398-406.

[8] Miklòs I, Podani J. Randomization of presence-absence matrices: comments and new algorithms. Ecology. Eco Soc America; 2004;85(1):86–92.

[9] Gotelli NJ. Null model analysis of species co-occurrence patterns. Ecology. 2000.

[10] Jaccard, Paul. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. Bulletin del la Société Vaudoise des Sciences Naturelles; 1901; 37:547–579.

[11] Hornik, K. and Meyer, D, and Buchta, C. (2014) *slam: Sparse Lightweight Arrays and Matrices*, R package `http://CRAN.R-project.org/package=slam`.

[12] Van der Maaten, L.J.P. and Hinton, G.E. Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008