

Introduction to iBBiG

Aedin Culhane, Daniel Gusenleitner

April 27, 2020

1 iBBiG

Iterative Binary Bi-clustering of Gene sets (iBBiG) is a bi-clustering algorithm optimized for discovery of overlapping biclusters in sparse binary matrices of data (Gusenleitner *et al.* in review).

We have optimized this method for the discovery of modules in matrices of discretized p -values from gene set enrichment analysis (GSA) of hundreds of datasets. However, it could be applied to any binary (1,0) matrix, such as discretized p -values from any sources of binary data. We apply iBBiG to meta-GSA to enable integrated analysis over hundreds of gene expression datasets. By integrating data at the levels of GSA results, we avoid the need to match probes/genes across multiple datasets, making large scale data integration a tractable problem.

iBBiG scales well with the dimensions of meta-datasets and is tolerant to noise characteristic of genomic data. It outperformed other traditional clustering approaches (Hierarchical clustering, k-means) or biclustering methods (bimax, fabia, coalesce) when applied to simulated data.

2 Application to simulated dataset

To demonstrate iBBiG, we will use a simulated binary dataset of 400 rows x 400 columns (as described by Gusenleitner *et al.*), in which a 1 indicates a positive association (or $p < 0.05$) between a gene set (row) and the results of a pairwise test between clinical covariates (column), and a 0 represents a lack of association.

To simulate random noise characteristically observed in genomic data, 10% random background noise (value of 1) was introduced into the matrix.

The matrix was seeded with seven artificial modules or bi-clusters (M1-M7; Figure 1) by assigning associations (value of 1) to its column and row pairs. To replicate the expected properties of real data, seeded modules partially overlapped in columns, in rows and in both rows and columns simultaneously. M1 gene sets overlap with most other modules with the exception of M3. M2 has overlapping pairwise tests with modules M4-7.

Artificial modules also have highly varying sizes and aspect ratios, including "wide" modules driven by a large number of pairwise tests and only a few gene sets and "tall" modules like M1 which consist of 25 pairwise tests and a large number of gene sets ($n = 250$). This latter type of module might represent a complex, well-characterized biological process such as proliferation.

In a real data set the signal strength will vary both between and within modules. Variance between modules was simulated by imposing random noise (1 -> 0 replacement) with different signal strengths on the modules (Figure 1). Within a module, we expect to see a few strong signals (gene sets associated with all pairwise tests) and many weaker signals. Therefore within each module, a noise gradient was also applied so that the first gene sets had the greatest number of associations (Figure 1). This overlaid noise gradient ranged from 10 to 60% and varied between modules (Table 1).

To create this simulated data as described in Gusenleitner, *et al.* use the function `makeArtificial` which creates an object of class `iBBiG`, an extension of `biclust`.

```
> library(iBBiG)
> binMat<-makeArtificial()
```

```
[1] "***** Summary of Design Matrix *****"
```

	Rows	Cols	DensityLow	DenistyHigh
M1	250	25	0.4	0.9
M2	75	175	0.4	0.8
M3	50	50	0.5	0.8
M4	40	40	0.4	0.9
M5	30	30	0.4	0.8
M6	20	20	0.6	0.9
M7	40	40	0.5	0.6

Cluster sizes in new iBBiG (Biclust) data object

Number of Modules: 7

Rows	250	75	50	40	30	20	40
Columns	25	175	50	40	30	20	40

```
> binMat
```

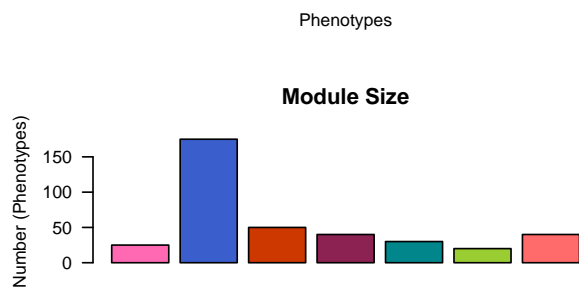
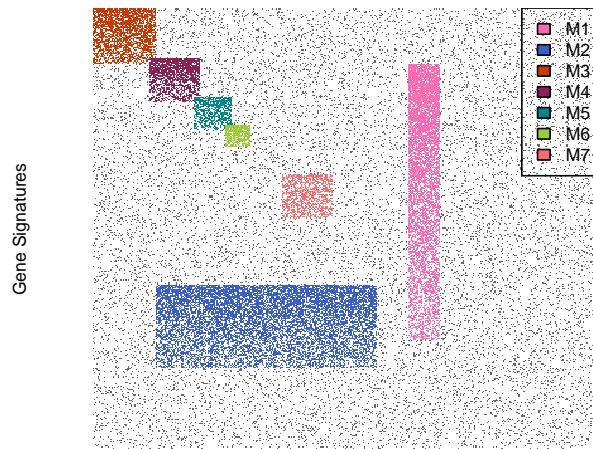
An object of class iBBiG

Number of Clusters found: 7

First 5 Cluster scores and sizes:

	[,1]	[,2]	[,3]	[,4]	[,5]
Cluster Score	NA	NA	NA	NA	NA
Number of Rows:	250	75	50	40	30
Number of Columns:	25	175	50	40	30

```
> plot(binMat)
```



The class *BiClust* contains the number of clusters and two logical matrices which indicate whether a row or column are present in the cluster.

```
> str(binMat)
```

```
Formal class 'iBBiG' [package "iBBiG"] with 8 slots
..@ Seeddata      : num [1:400, 1:400] 1 1 1 1 1 1 0 0 1 1 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:400] "sig_1" "sig_2" "sig_3" "sig_4" ...
.. .. ..$ : chr [1:400] "cov_1" "cov_2" "cov_3" "cov_4" ...
..@ RowScorexNumber: num[0 , 0 ]
..@ Clusterscores  : num(0)
..@ Parameters     :List of 3
.. ..$ designMatrix: num [1:7, 1:6] 251 51 1 46 81 106 151 51 251 1 ...
.. .. ..- attr(*, "dimnames")=List of 2
.. .. .. ..$ : chr [1:7] "M1" "M2" "M3" "M4" ...
.. .. .. ..$ : chr [1:6] "startC" "startR" "endC" "endR" ...
.. ..$ nRow      : num 400
.. ..$ nCol      : num 400
..@ RowxNumber    : logi [1:400, 1:7] FALSE FALSE FALSE FALSE FALSE FALSE ...
..@ NumberxCol    : logi [1:7, 1:400] FALSE FALSE TRUE FALSE FALSE FALSE ...
```

```

..@ Number      : int 7
..@ info        : list()

> Number(binMat)

[1] 7

> RowxNumber(binMat) [1:2, ]

      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,] FALSE FALSE TRUE  FALSE FALSE FALSE FALSE
[2,] FALSE FALSE TRUE  FALSE FALSE FALSE FALSE

> NumberxCol(binMat) [,1:2]

      [,1] [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,]  TRUE  TRUE
[4,] FALSE FALSE
[5,] FALSE FALSE
[6,] FALSE FALSE
[7,] FALSE FALSE

```

The matrix *RowxNumber* is a logical matrix having a number of rows equal to that of *binMat* and a number of columns equal to the number of detected clusters. *NumberxCol* is reversed; this class has a row count equal to the number of clusters and a column count of the number of columns of *binMat*.

To run *iBBiG* on this artificial binary matrix, simply call the function *iBBiG*. The function *plot* and *statClust* will provide a visual representation and statistical summary of the results of the cluster analysis.

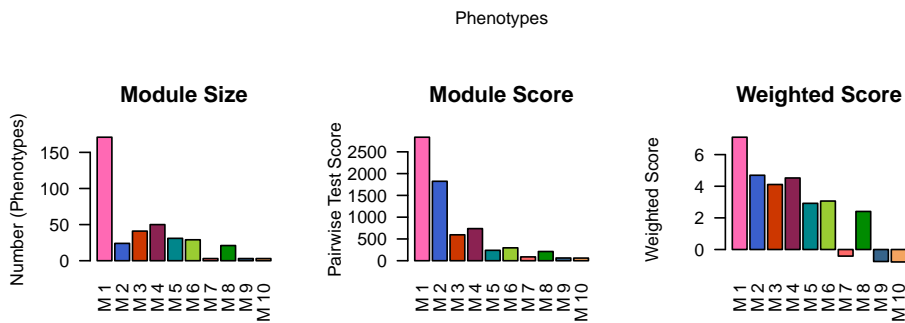
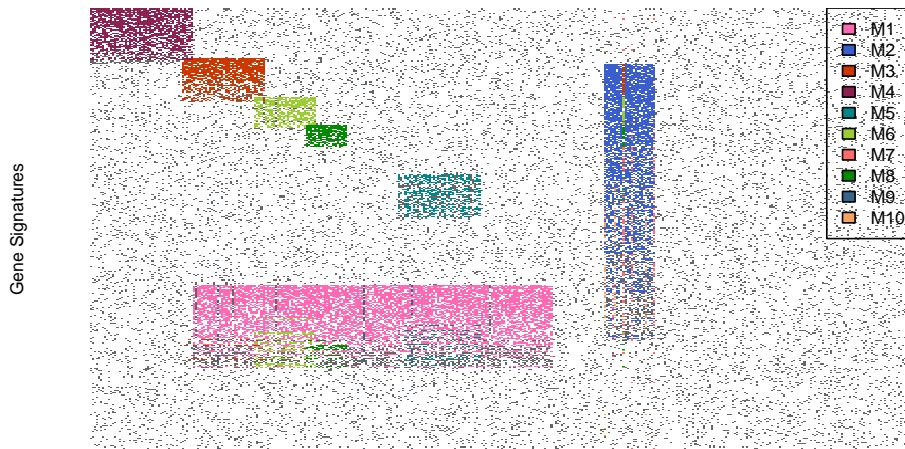
```

> res<- iBBiG(binMat@Seeddata, nModules=10)

Module: 1 ... done
Module: 2 ... done
Module: 3 ... done
Module: 4 ... done
Module: 5 ... done
Module: 6 ... done
Module: 7 ... done
Module: 8 ... done
Module: 9 ... done
Module: 10 ... done

> plot(res)

```



If you wish to compare two *iBBiG* or *Biclust* results, for example a prediction and a gold standard (GS), the function `JIdist` will calculate the Jaccard Index distance between two *Biclust* or *iBBiG* result objects. By default, it calculates the distances between each column. Setting `margin = row` or `margin = both` will cause the function to calculate instead the JI distance between the rows, or an average of rows/columns.

By default, `RfunctionJIdist` returns a *data.frame* with 2 columns, the column `n` indicating which cluster was the best match (maximum JI) to each cluster of the second *iBBiG* object (GS). The column `JI` contains the Jaccard Index distance between the columns of these two clusters. If `best = FALSE`, the function will return the distance matrix instead of the best match.

```
> JIdist(res, binMat)
```

```

      n  JI
GS_1 2 0.960
GS_2 1 0.966
GS_3 4 1.000
GS_4 3 0.976
GS_5 6 0.903
GS_6 8 0.952
GS_7 5 0.732

```

```
> JIdist(res, binMat, margin="col", best=FALSE)
```

	GS_1	GS_2	GS_3	GS_4
M 1	0.005128205	0.96590909	0.00000000	0.18539326
M 2	0.960000000	0.00000000	0.00000000	0.00000000
M 3	0.015384615	0.19337017	0.05813953	0.97560976
M 4	0.000000000	0.00000000	1.00000000	0.05882353
M 5	0.018181818	0.17045455	0.00000000	0.00000000
M 6	0.018867925	0.15909091	0.00000000	0.06153846
M 7	0.120000000	0.00000000	0.00000000	0.00000000
M 8	0.022222222	0.11363636	0.00000000	0.00000000
M 9	0.000000000	0.01714286	0.00000000	0.02380952
M 10	0.120000000	0.00000000	0.00000000	0.00000000

	GS_5	GS_6	GS_7
M 1	0.16860465	0.1169591	0.23391813
M 2	0.00000000	0.00000000	0.00000000
M 3	0.07575758	0.00000000	0.00000000
M 4	0.00000000	0.00000000	0.00000000
M 5	0.00000000	0.00000000	0.73170732
M 6	0.90322581	0.1136364	0.00000000
M 7	0.00000000	0.00000000	0.00000000
M 8	0.10869565	0.9523810	0.00000000
M 9	0.00000000	0.00000000	0.02380952
M 10	0.00000000	0.00000000	0.00000000

```
> JIdist(res,binMat, margin="col")
```

	n	JI
GS_1	2	0.960
GS_2	1	0.966
GS_3	4	1.000
GS_4	3	0.976
GS_5	6	0.903
GS_6	8	0.952
GS_7	5	0.732

```
> JIdist(res,binMat, margin="row")
```

	n	JI
GS_1	2	0.740
GS_2	1	0.800
GS_3	4	0.880
GS_4	3	0.791
GS_5	6	0.605
GS_6	8	0.655
GS_7	5	0.588

```
> JIdist(res,binMat, margin="both")
```

	n	JI
GS_1	2	1.700
GS_2	1	1.766
GS_3	4	1.880
GS_4	3	1.766

```
GS_5 6 1.508
GS_6 8 1.608
GS_7 5 1.320
```

To view the code of the function `JIdist`

```
> showMethods(JIdist)
> getMethod(iBBiG::JIdist, signature(clustObj = "iBBiG", GS = "iBBiG"))
> getMethod("JIdist", signature(clustObj="iBBiG", GS="iBBiG"))
```

To extract performance statistics between two `iBBiG` results, use `analyzeClust`, which will take a single `iBBiG` result object or a list of objects and compare these to a gold standard (another `iBBiG` or `biclust` object). Again results can be based on matches to the best row, column or both.

```
> analyzeClust(res, binMat)
```

```
[1] "list"
      Run n      JI nRow nCol col-accuracy col-sensitivity
GS_1  1 2 0.960  185  24      0.998      0.960
GS_2  1 1 0.966   60 171      0.985      0.971
GS_3  1 4 1.000   44  50      1.000      1.000
GS_4  1 3 0.976   37  41      0.998      1.000
GS_5  1 6 0.903   39  29      0.993      0.933
GS_6  1 8 0.952   28  21      0.998      1.000
GS_7  1 5 0.732   41  31      0.973      0.750

      col-specificity col-PPV col-NPV row-accuracy
GS_1          1.000  1.000  0.997      0.838
GS_2          0.996  0.994  0.978      0.963
GS_3          1.000  1.000  1.000      0.985
GS_4          0.997  0.976  1.000      0.978
GS_5          0.997  0.966  0.995      0.958
GS_6          0.997  0.952  1.000      0.975
GS_7          0.997  0.968  0.973      0.948

      row-sensitivity row-specificity row-PPV row-NPV
GS_1          0.740          1.000  1.000  0.698
GS_2          0.800          1.000  1.000  0.956
GS_3          0.880          1.000  1.000  0.983
GS_4          0.850          0.992  0.919  0.983
GS_5          0.867          0.965  0.667  0.989
GS_6          0.950          0.976  0.679  0.997
GS_7          0.750          0.969  0.732  0.972
```

```
> analyzeClust(res, binMat, margin="col")
```

```
[1] "list"
      Run n      JI nRow nCol col-accuracy col-sensitivity
GS_1  1 2 0.960  185  24      0.998      0.960
GS_2  1 1 0.966   60 171      0.985      0.971
GS_3  1 4 1.000   44  50      1.000      1.000
GS_4  1 3 0.976   37  41      0.998      1.000
GS_5  1 6 0.903   39  29      0.993      0.933
GS_6  1 8 0.952   28  21      0.998      1.000
GS_7  1 5 0.732   41  31      0.973      0.750
```

	col-specificity	col-PPV	col-NPV	row-accuracy
GS_1	1.000	1.000	0.997	0.838
GS_2	0.996	0.994	0.978	0.963
GS_3	1.000	1.000	1.000	0.985
GS_4	0.997	0.976	1.000	0.978
GS_5	0.997	0.966	0.995	0.958
GS_6	0.997	0.952	1.000	0.975
GS_7	0.997	0.968	0.973	0.948

	row-sensitivity	row-specificity	row-PPV	row-NPV
GS_1	0.740	1.000	1.000	0.698
GS_2	0.800	1.000	1.000	0.956
GS_3	0.880	1.000	1.000	0.983
GS_4	0.850	0.992	0.919	0.983
GS_5	0.867	0.965	0.667	0.989
GS_6	0.950	0.976	0.679	0.997
GS_7	0.750	0.969	0.732	0.972

Again to view the code of the function, you could:

```
> showMethods(analyzeClust)
> getMethod("analyzeClust", signature(clustObj="iBBiG", GS="iBBiG"))
```

The structure of *iBBiG* differs from *BiClust* in that it contains *ClusterScores*. *ClusterScores* are the scores for each module. *RowScoreNumber* are the scores for each row in the cluster. *Seeddata* is a copy of *binMat*.

```
> str(binMat)
```

```
Formal class 'iBBiG' [package "iBBiG"] with 8 slots
 ..@ Seeddata      : num [1:400, 1:400] 1 1 1 1 1 1 0 0 1 1 ...
 .. ..- attr(*, "dimnames")=List of 2
 .. .. ..$ : chr [1:400] "sig_1" "sig_2" "sig_3" "sig_4" ...
 .. .. ..$ : chr [1:400] "cov_1" "cov_2" "cov_3" "cov_4" ...
 ..@ RowScorexNumber: num[0 , 0 ]
 ..@ Clusterscores  : num(0)
 ..@ Parameters     :List of 3
 .. ..$ designMatrix: num [1:7, 1:6] 251 51 1 46 81 106 151 51 251 1 ...
 .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. ..$ : chr [1:7] "M1" "M2" "M3" "M4" ...
 .. .. .. ..$ : chr [1:6] "startC" "startR" "endC" "endR" ...
 .. ..$ nRow       : num 400
 .. ..$ nCol       : num 400
 ..@ RowxNumber    : logi [1:400, 1:7] FALSE FALSE FALSE FALSE FALSE FALSE ...
 ..@ NumberxCol   : logi [1:7, 1:400] FALSE FALSE TRUE FALSE FALSE FALSE ...
 ..@ Number       : int 7
 ..@ info         : list()
```

```
> RowScorexNumber(res)[1:2,]
```

	M 1	M 2	M 3	M 4	M 5	M 6	M 7	M 8	M 9	M 10
sig_1	0	0	0	29.12712	0	0	0	0	0	0
sig_2	0	0	0	33.04621	0	0	0	0	0	0

```
> Clusterscores(res)
```



```

      M 1      M 2      M 3      M 4      M 5
2835.88432 1824.20629 595.07269 737.47840 239.07043
      M 6      M 7      M 8      M 9      M 10
295.58934 87.92357 211.45922 62.54084 60.71773

```

```
> Seeddata(res)[1:2,1:2]
```

```

      cov_1 cov_2
sig_1     1     1
sig_2     1     0

```

There are also the slots for info and Parameters which can contain additional user-entered information about the analysis. We can subset or reorder the results like so:

```
> res[1:3]
```

An object of class iBBiG

Number of Clusters found: 3

First 3 Cluster scores and sizes:

```

      M 1      M 2      M 3
Cluster Score 2835.884 1824.206 595.0727
Number of Rows: 60.000 185.000 37.0000
Number of Columns: 171.000 24.000 41.0000

```

```
> res[c(4,2,1)]
```

An object of class iBBiG

Number of Clusters found: 3

First 3 Cluster scores and sizes:

```

      M 4      M 2      M 1
Cluster Score 737.4784 1824.206 2835.884
Number of Rows: 44.0000 185.000 60.000
Number of Columns: 50.0000 24.000 171.000

```

```
> res[1, drop=FALSE]
```

An object of class iBBiG

There was one cluster found with Score 2835.884 and
60 Rows and 171 columns

3 Using biclust functions

An object from *iBBiG* extends the class *biclust* and can therefore use methods available to a *biclust* object. For example, there are several plot functions in *BiClust*

```

> class(res)
> par(mfrow=c(2,1))
> drawHeatmap2(res@Seeddata, res, number=4)

```

```
> biclustmember(res, res@Seeddata)
> biclustbarchart(res@Seeddata, Bicres=res)
> plotclust(res, res@Seeddata)
```

Statistical measures of biclustering performance including the Chia and Karuturi Function, Coherence measures and F Statistics are available within the *biclust* R packages.

There are function to process data, binarize or discretize data. For example, given gene expression data we can binarize or discretize the data matrix as follows and this can be input into iBBiG

```
> data(BicatYeast)
> BicatYeast[1:5,1:5]
> binarize(BicatYeast[1:5,1:5], threshold=0.2)
> discretize(BicatYeast[1:5,1:5])
```

The sub-matrices of each cluster can be extracted from the original matrix, using the function `bicluster`

```
> Modules<-bicluster(res@Seeddata, res, 1:3)
> str(Modules)
```

List of 3

```
$ Bicluster1: num [1:60, 1:171] 0 1 1 1 1 1 0 0 1 1 ...
.- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:60] "sig_251" "sig_252" "sig_253" "sig_254" ...
.. ..$ : chr [1:171] "cov_51" "cov_53" "cov_54" "cov_55" ...
$ Bicluster2: num [1:185, 1:24] 1 1 1 1 1 1 1 1 1 1 ...
.- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:185] "sig_51" "sig_52" "sig_53" "sig_54" ...
.. ..$ : chr [1:24] "cov_251" "cov_252" "cov_253" "cov_254" ...
$ Bicluster3: num [1:37, 1:41] 1 1 0 1 1 1 1 1 1 1 ...
.- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:37] "sig_46" "sig_47" "sig_48" "sig_49" ...
.. ..$ : chr [1:41] "cov_46" "cov_47" "cov_48" "cov_49" ...
```

```
> Modules[[1]][1:3,1:4]
```

	cov_51	cov_53	cov_54	cov_55
sig_251	0	1	0	1
sig_252	1	0	1	1
sig_253	1	1	1	1

To write results to a file use the following:

```
> writeBiclusterResults("Modules.txt", res, bicName="Output from iBBiG with default pa
>
```

4 Session Info

- R version 4.0.0 (2020-04-24), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C

- Running under: Ubuntu 18.04.4 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.11-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.11-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, grid, methods, stats, utils
- Other packages: MASS 7.3-51.6, biclust 2.0.2, colorspace 1.4-1, iBBiG 1.32.0, lattice 0.20-41
- Loaded via a namespace (and not attached): R6 2.4.1, Rcpp 1.0.4.6, additivityTests 1.1-4, ade4 1.7-15, assertthat 0.2.1, class 7.3-17, compiler 4.0.0, crayon 1.3.4, dplyr 0.8.5, ellipsis 0.3.0, flexclust 1.4-0, ggplot2 3.3.0, glue 1.4.0, gtable 0.3.0, lifecycle 0.2.0, magrittr 1.5, modeltools 0.2-23, munsell 0.5.0, parallel 4.0.0, pillar 1.4.3, pkgconfig 2.0.3, purrr 0.3.4, rlang 0.4.5, scales 1.1.0, stats4 4.0.0, tibble 3.0.1, tidyr 1.0.2, tidyselect 1.0.0, tools 4.0.0, vctrs 0.2.4, xtable 1.8-4

References