

Introduction to iBBiG

Aedin Culhane, Daniel Gusenleitner

October 30, 2017

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.

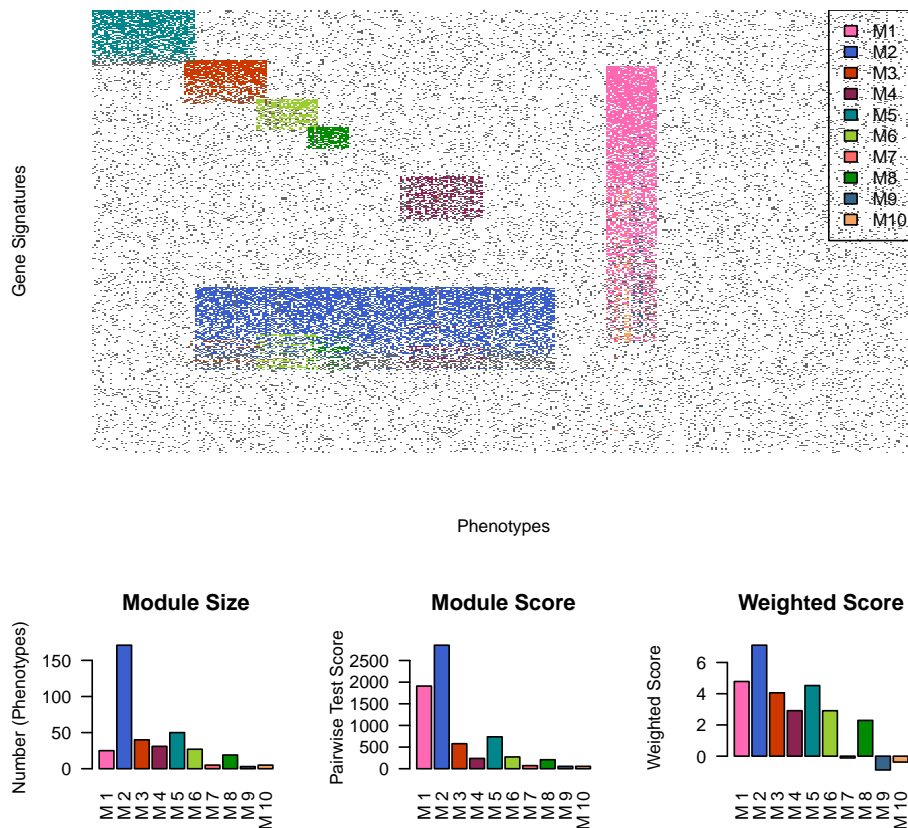
```

> 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 1 1.000
GS_2 2 0.977
GS_3 5 1.000
GS_4 3 1.000
GS_5 6 0.900
GS_6 8 0.950
GS_7 4 0.775
```

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

	GS_1	GS_2	GS_3	GS_4	GS_5
M 1	1.00000000	0.000000000	0.000000000	0.000000000	0.000000000
M 2	0.00000000	0.977142857	0.000000000	0.19209040	0.16860465
M 3	0.00000000	0.194444444	0.05882353	1.000000000	0.07692308
M 4	0.00000000	0.177142857	0.000000000	0.000000000	0.000000000
M 5	0.00000000	0.000000000	1.000000000	0.05882353	0.000000000
M 6	0.00000000	0.154285714	0.000000000	0.06349206	0.900000000
M 7	0.20000000	0.000000000	0.000000000	0.000000000	0.000000000
M 8	0.00000000	0.108571429	0.000000000	0.000000000	0.08888889
M 9	0.12000000	0.000000000	0.000000000	0.000000000	0.000000000
M 10	0.1538462	0.005586592	0.000000000	0.000000000	0.000000000

	GS_6	GS_7
M 1	0.00000000	0.000000000
M 2	0.1169591	0.22674419
M 3	0.00000000	0.000000000
M 4	0.00000000	0.77500000
M 5	0.00000000	0.000000000
M 6	0.1190476	0.000000000
M 7	0.00000000	0.000000000
M 8	0.95000000	0.000000000
M 9	0.00000000	0.000000000
M 10	0.00000000	0.02272727

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

	n	JI
GS_1	1	1.000
GS_2	2	0.977
GS_3	5	1.000
GS_4	3	1.000
GS_5	6	0.900
GS_6	8	0.950
GS_7	4	0.775

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

	n	JI
GS_1	1	0.764
GS_2	2	0.800
GS_3	5	0.860
GS_4	3	0.767
GS_5	6	0.548
GS_6	8	0.645
GS_7	4	0.646

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

	n	JI
GS_1	1	1.764
GS_2	2	1.777
GS_3	5	1.860
GS_4	3	1.767

```
GS_5 6 1.448
GS_6 8 1.595
GS_7 4 1.421
```

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 1 1.000  191   25         1.000           1.000
GS_2   1 2 0.977   60  171         0.990           0.977
GS_3   1 5 1.000   43   50         1.000           1.000
GS_4   1 3 1.000   36   40         1.000           1.000
GS_5   1 6 0.900   35   27         0.992           0.900
GS_6   1 8 0.950   31   19         0.998           0.950
GS_7   1 4 0.775   39   31         0.978           0.775
      col-specificity col-PPV col-NPV row-accuracy
GS_1                1      1   1.000       0.852
GS_2                1      1   0.983       0.962
GS_3                1      1   1.000       0.982
GS_4                1      1   1.000       0.975
GS_5                1      1   0.992       0.952
GS_6                1      1   0.997       0.972
GS_7                1      1   0.976       0.958
      row-sensitivity row-specificity row-PPV row-NPV
GS_1                0.764           1.000   1.000   0.718
GS_2                0.800           1.000   1.000   0.956
GS_3                0.860           1.000   1.000   0.980
GS_4                0.825           0.992   0.917   0.981
GS_5                0.767           0.968   0.657   0.981
GS_6                1.000           0.971   0.645   1.000
GS_7                0.775           0.978   0.795   0.975
```

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

```
[1] "list"
      Run n      JI nRow nCol col-accuracy col-sensitivity
GS_1   1 1 1.000  191   25         1.000           1.000
GS_2   1 2 0.977   60  171         0.990           0.977
GS_3   1 5 1.000   43   50         1.000           1.000
GS_4   1 3 1.000   36   40         1.000           1.000
GS_5   1 6 0.900   35   27         0.992           0.900
GS_6   1 8 0.950   31   19         0.998           0.950
GS_7   1 4 0.775   39   31         0.978           0.775
```

	col-specificity	col-PPV	col-NPV	row-accuracy
GS_1	1	1	1.000	0.852
GS_2	1	1	0.983	0.962
GS_3	1	1	1.000	0.982
GS_4	1	1	1.000	0.975
GS_5	1	1	0.992	0.952
GS_6	1	1	0.997	0.972
GS_7	1	1	0.976	0.958

	row-sensitivity	row-specificity	row-PPV	row-NPV
GS_1	0.764		1.000	1.000
GS_2	0.800		1.000	1.000
GS_3	0.860		1.000	1.000
GS_4	0.825		0.992	0.917
GS_5	0.767		0.968	0.657
GS_6	1.000		0.971	0.645
GS_7	0.775		0.978	0.795

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	0	29.12712	0	0	0	0	0
sig_2	0	0	0	0	33.04621	0	0	0	0	0

```
> Clusterscores(res)
```


	M 1	M 2	M 3	M 4	M 5
	1909.84107	2853.54342	578.31967	237.83470	736.70702
	M 6	M 7	M 8	M 9	M 10
	272.25168	71.20281	208.02998	55.17235	54.84358

```
> 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	1909.841	2853.543	578.3197
Number of Rows:	191.000	60.000	36.0000
Number of Columns:	25.000	171.000	40.0000

```
> 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	237.8347	2853.543	1909.841
Number of Rows:	39.0000	60.000	191.000
Number of Columns:	31.0000	171.000	25.000

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

An object of class iBBiG

There was one cluster found with Score 1909.841 and
191 Rows and 25 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 [1:191, 1:25] 1 1 1 1 1 1 1 1 1 1 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:191] "sig_51" "sig_52" "sig_53" "sig_54" ...
.. ..$ : chr [1:25] "cov_251" "cov_252" "cov_253" "cov_254" ...
$ Bicluster2: 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_52" "cov_53" "cov_54" ...
$ Bicluster3: num [1:36, 1:40] 1 1 0 1 1 1 1 1 1 1 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:36] "sig_46" "sig_47" "sig_48" "sig_49" ...
.. ..$ : chr [1:40] "cov_46" "cov_47" "cov_48" "cov_49" ...
```

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

	cov_251	cov_252	cov_253	cov_254
sig_51	1	1	1	1
sig_52	1	1	1	1
sig_53	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 3.4.2 (2017-09-28), x86_64-apple-darwin15.6.0
- Locale: C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
- Running under: OS X El Capitan 10.11.6
- Matrix products: default

- BLAS:
`/Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib`
- LAPACK:
`/Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, grid, methods, stats, utils
- Other packages: MASS 7.3-47, biclust 1.2.0, colorspace 1.3-2, iBBiG 1.22.0, lattice 0.20-35
- Loaded via a namespace (and not attached): ade4 1.7-8, compiler 3.4.2, flexclust 1.3-4, modeltools 0.2-21, parallel 3.4.2, stats4 3.4.2, tools 3.4.2, xtable 1.8-2

References