Package 'BiBitR'

June 30, 2017

Type Package

Title R Wrapper for Java Implementation of BiBit

Version 0.3.1

Date 2017-06-30

Author De Troyer Ewoud

Maintainer De Troyer Ewoud <ewoud.detroyer@uhasselt.be>

Description A simple R wrapper for the Java BiBit algorithm from ``A
biclustering algorithm for extracting bit-patterns from binary datasets''
from Domingo et al. (2011) <DOI:10.1093/bioinformatics/btr464>. An simple adaptation for the BiBit algorithm which allows noise in the biclusters is also introduced as well as a function to guide the algorithm towards given (sub)patterns. Further, a workflow to derive noisy biclusters from discoverd larger column patterns is included as well.

License GPL-3

Imports
stats,foreign,methods,utils,viridis,cluster,dendextend,lattice,grDevices,graphics,randomcoloR,biclust

RoxygenNote 5.0.1

SystemRequirements Java

NeedsCompilation no

Repository CRAN

Date/Publication 2017-06-30 17:38:22 UTC

R topics documented:

`bibit` *The BiBit Algorithm*

Description

A R-wrapper which directly calls the original Java code for the BiBit algorithm (`http://eps.upo.es/bigs/BiBit.html`) and transforms it to the output format of the `Biclust` R package.

Usage

```
bibit(matrix = NULL, minr = 2, minc = 2, arff_row_col = NULL,
  output_path = NULL)
```

Arguments

`matrix`	The binary input matrix.
`minr`	The minimum number of rows of the Biclusters.
`minc`	The minimum number of columns of the Biclusters.
`arff_row_col`	If you want to circumvent the internal R function to convert the matrix to `.arff` format, provide the pathname of this file here. Additionally, two `.csv` files should be provided containing 1 column of row and column names. These two files should not contain a header or quotes around the names, simply 1 column with the names. (*Example*: `arff_row_col=c("...\\data\\matrix.arff","...\\data\\rownames.csv","...\\dat`) *Note:* These files can be generated with the `make_arff_row_col` function. **Warning:** Should you use the `write.arff` function from the `foreign` package, remember to transpose the matrix first.
`output_path`	If as output, the original txt output of the Java code is desired, provide the outputh path here (without extension). In this case the `bibit` function will skip the transformation to a Biclust class object and simply return `NULL`. (*Example*: `output_path="...\\out\\bibitresult"`) (*Description Output*: The following information about every bicluster generated will be printed in the output file: number of rows, number of columns, name of rows and name of columns.

Details

This function uses the original Java code directly (with the intended input and output). Because the Java code was not refactored, the `rJava` package could not be used. The `bibit` function does the following:

1. Convert R matrix to a `.arff` output file.
2. Use the `.arff` file as input for the Java code which is called by `system()`.
3. The outputted `.txt` file from the Java BiBit algorithm is read in and transformed to a `Biclust` object.

Because of this, there is a chance of *overhead* when applying the algorithm on large datasets. Make sure your machine has enough RAM available when applying to big data.

Value

A Biclust S4 Class object.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit(data,minr=5,minc=5)
result
MaxBC(result)

## End(Not run)
```

`bibit2` *The BiBit Algorithm with Noise Allowance*

Description

Same function as `bibit` with an additional new noise parameter which allows 0's in the discovered biclusters (See Details for more info).

Usage

```
bibit2(matrix = NULL, minr = 2, minc = 2, noise = 0,
  arff_row_col = NULL, output_path = NULL, extend_columns = "none",
  extend_mincol = 1, extend_limitcol = 1, extend_noise = noise,
  extend_contained = FALSE)
```

Arguments

`matrix` The binary input matrix.

`minr` The minimum number of rows of the Biclusters.

`minc` The minimum number of columns of the Biclusters.

`noise` Noise parameter which determines the amount of zero's allowed in the bicluster (i.e. in the extra added rows to the starting row pair).

- `noise=0`: No noise allowed. This gives the same result as using the `bibit` function. (default)
- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a (extra) row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of `5`, the number of allowed 0's would be `1`.
- `noise>=1`: The `noise` parameter will be the number of allowed 0's in a (extra) row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

`arff_row_col` If you want to circumvent the internal R function to convert the matrix to `.arff` format, provide the pathname of this file here. Additionally, two `.csv` files should be provided containing 1 column of row and column names. These two files should not contain a header or quotes around the names, simply 1 column with the names.
(*Example*: `arff_row_col=c("...\\data\\matrix.arff","...\\data\\rownames.csv","...\\dat`
Note: These files can be generated with the `make_arff_row_col` function.
Warning: Should you use the `write.arff` function from the `foreign` package, remember to transpose the matrix first.

`output_path` If as output, the original txt output of the Java code is desired, provide the outputh path here (without extension). In this case the `bibit` function will skip the transformation to a Biclust class object and simply return `NULL`.
(*Example*: `output_path="...\\out\\bibitresult"`)
(*Description Output*: The following information about every bicluster generated will be printed in the output file: number of rows, number of columns, name of rows and name of columns.

`extend_columns` *Column Extension Parameter*
Can be one of the following: `"none"`, `"naive"`, `"recursive"` which will apply either a naive or recursive column extension procedure. (See Details Section for more information.)
Based on the extension, additional biclusters will be created in the Biclust object which can be seen in the column and row names of the `RowxNumber` and `NumberxCol` slots (`"_Ext"` suffix).

The `info` slot will also contain some additional information. Inside this slot, `BC.Extended` contains info on which original biclusters were extended, how many columns were added, and in how many extra extended biclusters this resulted.

Warning: Using a percentage-based `extend_noise` (or `noise` by default) in combination with the recursive procedure will result in a large amount of biclusters and increase the computation time a lot. Depending on the data when using recursive in combination with a noise percentage, it is advised to keep it reasonable small (e.g. 10%). Another remedy is to sufficiently increase the `extend_limitcol` either as a percentage or integer to limit the candidates of columns.

`extend_mincol` *Column Extension Parameter*
A minimum number of columns that a bicluster should be able to be extended with before saving the result. (Default=1)

`extend_limitcol`
Column Extension Parameter
The number (`extend_limitcol>=1`) or percentage (`0<extend_limitcol<1`) of 1's that a column (subsetted on the BC rows) should at least contain for it to be a candidate to be added to the bicluster as an extension. (Default=1) (Increase this parameter if the recursive extension takes too long. Limiting the pool of candidates will decrease computation time, but restrict the results more.)

`extend_noise` *Column Extension Parameter*
The maximum allowed noise (in each row) when extending the columns of the bicluster. Can take the same as the `noise` parameter. By default this is the same value as `noise`.

`extend_contained`
Column Extension Parameter
Logical value if extended results should be checked if they contain each other (and deleted if this is the case). Default = `FALSE`. This can be a lengthy procedure for a large amount of biclusters (>1000).

Value

A Biclust S4 Class object.

Details - General

`bibit2` follows the same steps as described in the Details section of `bibit`.
Following the general steps of the BiBit algorithm, the allowance for noise in the biclusters is inserted in the original algorithm as such:

1. Binary data is encoded in bit words.
2. Take a pair of rows as your starting point.
3. Find the maximal overlap of 1's between these two rows and save this as a pattern/motif. You now have a bicluster of 2 rows and N columns in which N is the number of 1's in the motif.

4. Check all remaining rows if they match this motif, *however* allow a specific amount of 0's in this matching as defined by the `noise` parameter. Those rows that match completely or those within the allowed noise range are added to bicluster.
5. Go back to *Step 2* and repeat for all possible row pairs.

Note: Biclusters are only saved if they satisfy the `minr` and `minc` parameter settings and if the bicluster is not already contained completely within another bicluster.

What you will end up with are biclusters not only consisting out of 1's, but biclusters in which 2 rows (the starting pair) are all 1's and in which the other rows could contain 0's (= noise).

Note: Because of the extra checks involved in the noise allowance, using noise might increase the computation time a little bit.

Details - Column Extension

An optional procedure which can be applied *after* applying the BiBit algorithm (with noise) is called *Column Extension*. The procedure will add extra columns to a BiBit bicluster, keeping into account the allowed `extend_noise` level in each row. The primary goal is to, after applying BiBit with noise, to also try and add some noise to the 2 initial ‘perfect‘ rows. Other parameters like `extend_mincol` and `extend_limitcol` can also further restrict which extensions should be discovered.
This procedure can be done either *naively* (fast) or *recursively* (more slow and thorough) with the `extend_columns` parameter.

`"naive"` Subsetting on the bicluster rows, the column candidates are ordered based on the most 1's in a column. Afterwards, in this order, each column is sequentially checked and added when the resulted BC is still within row noise levels.
This has 2 major consequences:
- If 2 columns are identical, the first in the dataset is added, while the second isn't (depending on the noise level allowed per row).
- If 2 non-identical columns are viable to be added (correct row noise), the column with the most 1's is added. Afterwards the second column might not be viable anymore.

Note that using this method will always result in a maximum of 1 extended bicluster per original bicluster.

`"recursive"` Conditioning the group of candidates for the allowed row noise level, each possible/allowed combination of adding columns to the bicluster is checked. Only the resulted biclusters with the highest number of extra columns are saved. Of course this could result in multiple extensions for 1 bicluster if there are multiple ‘maximum added columns‘ results.

Note: These procedures are followed by a fast check if the extensions resulted in any duplicate biclusters. If so, these are deleted from the final result.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

result1 <- bibit2(data,minr=5,minc=5,noise=0.2)
result1
MaxBC(result1,top=1)

result2 <- bibit2(data,minr=5,minc=5,noise=3)
result2
MaxBC(result2,top=2)

## End(Not run)
```

---

bibit3 — *The BiBit Algorithm with Noise Allowance guided by Provided Patterns.*

Description

Same function as `bibit2` but only aims to discover biclusters containing the (sub) pattern of provided patterns or their combinations.

Usage

```
bibit3(matrix = NULL, minr = 1, minc = 2, noise = 0,
  pattern_matrix = NULL, subpattern = TRUE, pattern_combinations = FALSE,
  arff_row_col = NULL, extend_columns = "none", extend_mincol = 1,
  extend_limitcol = 1, extend_noise = noise, extend_contained = FALSE)
```

Arguments

`matrix`	The binary input matrix.
`minr`	The minimum number of rows of the Biclusters. (Note that in contrast to `bibit` and `bibit2`, this can be be set to 1 since we are looking for additional rows to the provided pattern.)
`minc`	The minimum number of columns of the Biclusters.
`noise`	Noise parameter which determines the amount of zero's allowed in the bicluster (i.e. in the extra added rows to the starting row pair).

- `noise=0`: No noise allowed. This gives the same result as using the `bibit` function. (default)

- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a (extra) row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of 5, the number of allowed 0's would be 1.
- `noise>=1`: The `noise` parameter will be the number of allowed 0's in a (extra) row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

`pattern_matrix` Matrix (Number of Patterns x Number of Data Columns) containing the patterns of interest.

`subpattern` Boolean value if sub patterns are of interest as well (default=TRUE).

`pattern_combinations`
Boolean value if the pairwise combinations of patterns (the intersecting 1's) should also used as starting points (default=FALSE).

`arff_row_col` Same argument as in `bibit` and `bibit2`. However you can only provide 1 pattern by using this option. For `bibit3` to work, the pattern has to be added 2 times on top of the matrix (= identical first 2 rows).

`extend_columns` *Column Extension Parameter*
Can be one of the following: `"none"`, `"naive"`, `"recursive"` which will apply either a naive or recursive column extension procedure. (See Details Section for more information.)
Based on the extension, additional biclusters will be created in the Biclust object which can be seen in the column and row names of the `RowxNumber` and `NumberxCol` slots (`"_Ext"` suffix).
The `info` slot will also contain some additional information. Inside this slot, `BC.Extended` contains info on which original biclusters were extended, how many columns were added, and in how many extra extended biclusters this resulted.

Warning: Using a percentage-based `extend_noise` (or `noise` by default) in combination with the recursive procedure will result in a large amount of biclusters and increase the computation time a lot. Depending on the data when using recursive in combination with a noise percentage, it is advised to keep it reasonable small (e.g. 10%). Another remedy is to sufficiently increase the `extend_limitcol` either as a percentage or integer to limit the candidates of columns.

`extend_mincol` *Column Extension Parameter*
A minimum number of columns that a bicluster should be able to be extended with before saving the result. (Default=1)

`extend_limitcol`
Column Extension Parameter
The number (`extend_limitcol>=1`) or percentage (`0<extend_limitcol<1`) of 1's that a column (subsetted on the BC rows) should at least contain for it to be a candidate to be added to the bicluster as an extension. (Default=1) (Increase this parameter if the recursive extension takes too long. Limiting the pool of candidates will decrease computation time, but restrict the results more.)

`extend_noise` *Column Extension Parameter*
The maximum allowed noise (in each row) when extending the columns of the bicluster. Can take the same as the `noise` parameter. By default this is the same value as `noise`.

`extend_contained`
Column Extension Parameter
Logical value if extended results should be checked if they contain each other (and deleted if this is the case). Default = `FALSE`. This can be a lengthy procedure for a large amount of biclusters (>1000).

Details

The goal of the `bibit3` function is to provide one or multiple patterns in order to only find those biclusters exhibiting those patterns. Multiple patterns can be given in matrix format, `pattern_matrix`, and their pairwise combinations can automatically be added to this matrix by setting `pattern_combinations=TRUE`. All discovered biclusters are still subject to the provided `noise` level.

Three types of Biclusters can be discovered:

Full Pattern: Bicluster which overlaps completely (within allowed noise levels) with the provided pattern. The column size of this bicluster is always equal to the number of 1's in the pattern.

Sub Pattern: Biclusters which overlap with a part of the provided pattern within allowed noise levels. Will only be given if `subpattern=TRUE` (default). Setting this option to `FALSE` decreases computation time.

Extended: Using the resulting biclusters from the full and sub patterns, other columns will be attempted to be added to the biclusters while keeping the noise as low as possible (the number of rows in the BC stays constant). This can be done either with `extend_columns` equal to `"naive"` or `"recursive"`. More info on the difference can be found in the Details Section of `bibit2`.
Naturally the articially added pattern rows will not be taken into account with the noise levels as they are 0 in each other column.
The question which is attempted to be answered here is *'Do the rows, which overlap partly or fully with the given pattern, have other similarities outside the given pattern¿*

How?
The BiBit algorithm is applied to a data matrix that contains 2 identical artificial rows at the top which contain the given pattern. The default algorithm is then slightly altered to only start from this articial row pair (=Full Pattern) or from 1 artificial row and 1 other row (=Sub Pattern).

Note 1 - Large Data:
The `arff_row_col` can still be provided in case of large data matrices, but the `.arff` file should already contain the pattern of interest in the first two rows. Consequently not more than 1 pattern at a time can be investigated with a single call of `bibit3`.

Note 2 - Viewing Results:
A `print` and `summary` method has been implemented for the output object of `bibit3`. It gives an overview of the amount of discovered biclusters and their dimensions
Additionally, the `bibit3_patternBC` function can extract a Bicluster and add the artificial pattern rows to investigate the results.

Value

A S3 list object, `"bibit3"` in which each element (apart from the last one) corresponds with a provided pattern or combination thereof.
Each element is a list containing:

- `Number`**:** Number of Initially found BC's by applying BiBit with the provided pattern.
- `Number_Extended`**:** Number of additional discovered BC's by extending the columns.
- `FullPattern`**:** Biclust S4 Class Object containing the Bicluster with the Full Pattern.
- `SubPattern`**:** Biclust S4 Class Object containing the Biclusters showing parts of the pattern.
- `Extended`**:** Biclust S4 Class Object containing the additional Biclusters after extending the biclusters (column wise) of the full and sub patterns
- `info`**:** Contains `Time_Min` element which includes the elapsed time of parts and the full analysis.

The last element in the list is a matrix containing all the investigated patterns.

Author(s)

Ewoud De Troyer

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

Examples

```
## Not run:
set.seed(1)
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
colsel <- sample(1:ncol(data),ncol(data))
data <- data[sample(1:nrow(data),nrow(data)),colsel]

pattern_matrix <- matrix(0,nrow=3,ncol=100)
pattern_matrix[1,1:7] <- 1
pattern_matrix[2,11:15] <- 1
pattern_matrix[3,13:20] <- 1

pattern_matrix <- pattern_matrix[,colsel]


out <- bibit3(matrix=data,minr=2,minc=2,noise=0.1,pattern_matrix=pattern_matrix,
              subpattern=TRUE,extend_columns=TRUE,pattern_combinations=TRUE)
out  # OR print(out) OR summary(out)


bibit3_patternBC(result=out,matrix=data,pattern=c(1),type=c("full","sub","ext"),BC=c(1,2))

## End(Not run)
```

bibit3_patternBC *Extract BC from* `bibit3` *result and add pattern*

Description

Function which will print the BC matrix and add 2 duplicate articial pattern rows on top. The function allows you to see the BC and the pattern the BC was guided towards to.

Usage

```
bibit3_patternBC(result, matrix, pattern = c(1), type = c("full", "sub",
  "ext"), BC = c(1))
```

Arguments

`result`	Result produced by `bibit3`
`matrix`	The binary input matrix.
`pattern`	Vector containing either the number or name of which patterns the BC results should be extracted.
`type`	Vector for which BC results should be printed. • Full Pattern (`"full"`) • Sub Pattern (`"sub"`) • Extended (`"ext"`)
`BC`	Vector of BC indices which should be printed, conditioned on `pattern` and `type`.

Value

Prints queried biclusters.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
set.seed(1)
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
colsel <- sample(1:ncol(data),ncol(data))
data <- data[sample(1:nrow(data),nrow(data)),colsel]

pattern_matrix <- matrix(0,nrow=3,ncol=100)
pattern_matrix[1,1:7] <- 1
```

```
pattern_matrix[2,11:15] <- 1
pattern_matrix[3,13:20] <- 1

pattern_matrix <- pattern_matrix[,colsel]


out <- bibit3(matrix=data,minr=2,minc=2,noise=0.1,pattern_matrix=pattern_matrix,
              subpattern=TRUE,extend_columns=TRUE,pattern_combinations=TRUE)
out  # OR print(out) OR summary(out)


bibit3_patternBC(result=out,matrix=data,pattern=c(1),type=c("full","sub","ext"),BC=c(1,2))

## End(Not run)
```

`BiBitR` *A biclustering algorithm for extracting bit-patterns from binary datasets*

Description

BiBitR is a simple R wrapper which directly calls the original Java code for applying the BiBit algorithm. The original Java code can be found at `http://eps.upo.es/bigs/BiBit.html` by Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz.

The BiBitR package also includes the following functions and/or workflows:

- A slightly adapted version of the original BiBit algorithm which now allows allows noise when adding rows to the bicluster (`bibit2`).
- A function which accepts a pattern and, using the BiBit algorithm, will find biclusters fully or partly fitting the given pattern (`bibit3`).
- A workflow which can discover larger patterns (and their biclusters) using BiBit and classic hierarchical clustering approaches (`BiBitWorkflow`).

References

Domingo S. Rodriguez-Baena, Antonia J. Perez-Pulido and Jesus S. Aguilar-Ruiz (2011), "A biclustering algorithm for extracting bit-patterns from binary datasets", *Bioinformatics*

`BiBitWorkflow` *BiBit Workflow*

Description

Workflow to discover larger (noisy) patterns in big data using BiBit

Usage

```
BiBitWorkflow(matrix, minr = 2, minc = 2, similarity_type = "col",
  func = "agnes", link = "average", par.method = 0.625,
  cut_type = "gap", cut_pm = "Tibs2001SEmax", gap_B = 500,
  gap_maxK = 50, noise = 0.1, noise_select = 0, plots = c(3:5),
  BCresult = NULL, simmatresult = NULL, treeresult = NULL,
  plot.type = "device", filename = "BiBitWorkflow", verbose = TRUE)
```

Arguments

`matrix`	The binary input matrix.
`minr`	The minimum number of rows of the Biclusters.
`minc`	The minimum number of columns of the Biclusters.
`similarity_type`	Which dimension to use for the Jaccard Index in Step 2. This is either columns (`"col"`, default) or both (`"both"`).
`func`	Which clustering function to use in Step 3. Either `"agnes"` (= default) or `"hclust"`.
`link`	Which clustering link to use in Step 3. The available links (depending on `func`) are: • `hclust`: `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"` • `agnes`: `"average"` (default), `"single"`, `"complete"`, `"ward"`, `"weighted"`, `"gaverage"` or `"flexible"` (More details in `hclust` and `agnes`)
`par.method`	Additional parameters used for flexible link (See `agnes`). Default is `c(0.625)`
`cut_type`	Which method should be used to decide the number of clusters in the tree in Step 4? • `"gap"`: Use the Gap Statistic (default). • `"number"`: Select a set number of clusters. • `"height"`: Cut the tree at specific dissimilarity height.
`cut_pm`	Cut Parameter (depends on `cut_type`) for Step 4 • Gap Statistic (`cut_type="gap"`): How to compute optimal number of clusters? Choose one of the following: `"Tibs2001SEmax"` (default), `"globalmax"`, `"firstmax"`, `"firstSEmax"` or `"globalSEmax"`. • Number (`cut_type="number"`): Integer for number of clusters. • Height (`cut_type="height"`): Numeric dissimilarity value where the tree should be cut (`[0,1]`).
`gap_B`	Number of bootstrap samples (default=500) for Gap Statistic (`clusGap`).
`gap_maxK`	Number of clusters to consider (default=50) for Gap Statistic (`clusGap`).
`noise`	The allowed noise level when growing the rows on the merged patterns in Step 6. (default=`0.1`, namely allow 10% noise.) • `noise=0`: No noise allowed.

- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of `5`, the number of allowed 0's would be `1`.
- `noise>=1`: The `noise` parameter will be the number of allowed 0's in a row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

`noise_select` Should the allowed noise level be automatically selected for each pattern? (Using ad hoc method to find the elbow/kink in the Noise Scree plots)

- `noise_select=0`: Do *NOT* automatically select the noise levels. Use the the noise level given in the `noise` parameter (default).
- `noise_select=1`: Using the Noise Scree plot (with 'Added Rows' on the y-axis), find the noise level where the current number of added rows at this noise level is larger than the mean of 'added rows' at the lower noise levels. After locating this noise level, lower the noise level by 1. This is your automatically selected elbow/kink and therefore your noise level.
- `noise_select=2`: Applies the same steps as for `noise_select=1`, but instead of decreasing the noise level by only 1, keep decreasing the noise level until the number of added rows isn't decreasing anymore either.

`plots` Vector for which plots to draw:

1. Image plot of the similarity matrix computed in Step 2.
2. Same as `plots=1`, but the rows and columns are reordered with the hierarchical tree.
3. Dendrogram of the tree, its clusters colored after the chosen cut has been applied.
4. Noise Scree plots for all the Saved Patterns. Two plots will be plotted, both with Noise on the x-axis. The first one will have the number of Added Number of Rows on that noise level on the y-axis, while the second will have the Total Number of Rows (i.e. cumulative of the first). If the title of one of the subplots is red, then this means that the Bicluster grown from this pattern, using the chosen noise level, was eventually deleted due to being a duplicate or non-maximal.
5. Image plot of the Jaccard Index similarity matrix between the final biclusters after Step 6.

`BCresult` Import a BiBit Biclust result for Step 1 (e.g. extract from an older BiBitWorkflow object `$info$BiclustInitial`). This can be useful if you want to cut the tree differently/make different plots, but don't want to do the BiBit calculation again.

`simmatresult` Import a (custom) Similarity Matrix (e.g. extract from older BiBitWorkflow object `$info$BiclustSimInitial`). Note that Step 1 (BiBit) will still be executed if `BCresult` is not provided.

`treeresult` Import a (custom) tree (`hclust` object) based on the BiBit/Similarity (e.g. extract from older BiBitWorkflow object `$info$Tree`).

`plot.type` Output Type

- `"device"`: All plots are outputted to new R graphics devices (default).
- `"file"`: All plots are saved in external files. Plots 1 and 2 are saved in separate `.png` files while all other plots are joint together in a single `.pdf` file.
- `"other"`: All plots are outputted to the current graphics device, but will overwrite each other. Use this if you want to include one or more plots in a sweave/knitr file or if you want to export a single plot by your own chosen format.

`filename`	Base filename (with/without directory) for the plots if `plot.type="file"` (default=`"BiBitWorkflow"`).
`verbose`	Logical value if progress of workflow should be printed.

Details

Looking for Noisy Biclusters in large data using BiBit (`bibit2`) often results in many (overlapping) biclusters. In order decrease the number of biclusters and find larger meaningful patterns which make up noisy biclusters, the following workflow can be applied. Note that this workflow is primarily used for data where there are many more rows (e.g. patients) than columns (e.g. symptoms). For example the workflow would discover larger meaningful symptom patterns which, conditioned on the allowed noise/zeros, subsets of the patients share.

1. Apply BiBit with *no noise* (Preferably with high enough `minr` and `minc`).
2. Compute Similarity Matrix (Jaccard Index) of all biclusters. By default this measure is only based on column similarity. This implies that the rows of the BC's are not of interest in this step. The goal then would be to discover highly overlapping column patterns and, in the next steps, merge them together.
3. Apply Agglomerative Hierarchical Clustering on Similarity Matrix (default = average link)
4. Cut the dendrogram of the clustering result and merge the biclusters based on this. (default = number of clusters is determined by the Tibs2001SEmax Gap Statistic)
5. Extract Column Memberships of the Merged Biclusters. These are saved as the new column *Patterns*.
6. Starting from these patterns, *(noisy) rows* are grown which match the pattern, creating a single final bicluster for each pattern. At the end duplicate/non-maximal BC's are deleted.

Using the described workflow (and column similarity in Step 2), the final result will contain biclusters which focus on larger column patterns.

Value

A BiBitWorkflow S3 List Object with 3 slots:

- `Biclust`: Biclust Class Object of Final Biclustering Result (after Step 6).
- `BiclustSim`: Jaccard Index Similarity Matrix of Final Biclustering Result (after Step 6).
- `info`: List Object containing:
  - `BiclustInitial`: Biclust Class Object of Initial Biclustering Result (after Step 1).
  - `BiclustSimInitial`: Jaccard Index Similarity Matrix of Initial Biclustering Result (after Step 1).

- `Tree`: Hierarchical Tree of `BiclustSimInitial` as `hclust` object.
- `Number`: Vector containing the initial number of biclusters (`InitialNumber`), the number of saved patterns after cutting the tree (`PatternNumber`) and the final number of biclusters (`FinalNumber`).
- `GapStat`: Vector containing all different optimal cluster numbers based on the Gap Statistic.
- `BC.Merge`: A list (length of merged saved patterns) containing which biclusters were merged together after cutting the tree.
- `MergedColPatterns`: A list (length of merged saved patterns) containing the indices of which columns make up that pattern.
- `MergedNoiseThresholds`: A vector containing the selected noise levels for the merged saved patterns.
- `Coverage`: A list containing: 1. a vector of the total number (and percentage) of unique rows the final biclusters cover. 2. a table showing how many rows are used more than a single time in the final biclusters.
- `Call`: A match.call of the original function call.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
## Simulate Data ##
# DATA: 10000x50
# BC1: 200x10
# BC2: 100x10
# BC1 and BC2 overlap 5 columns

# BC3: 200x10
# BC4: 100x10
# BC3 and bC4 overlap 2 columns

# Background 1 percentage: 0.15
# BC Signal Percentage: 0.9

set.seed(273)
mat <- matrix(sample(c(0,1),10000*50,replace=TRUE,prob=c(1-0.15,0.15)),
              nrow=10000,ncol=50)
mat[1:200,1:10] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                          nrow=200,ncol=10)
mat[300:399,6:15] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                            nrow=100,ncol=10)
mat[400:599,21:30] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=200,ncol=10)
mat[700:799,29:38] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=100,ncol=10)
mat <- mat[sample(1:10000,10000,replace=FALSE),sample(1:50,50,replace=FALSE)]
```

```
# Computing gap statistic for initial 1381 BC takes approx. 15 min.
# Gap Statistic chooses 4 clusters.
out <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2)
summary(out$Biclust)

# Reduce computation by selecting number of clusters manually.
# Note: The "ClusterRowCoverage" function can be used to provided extra info
#       on the number of cluster choice.
#       How?
#       - More clusters result in smaller column patterns and more matching rows.
#       - Less clusters result in larger column patterns and less matching rows.
# Step 1: Initial Workflow Run
out2 <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2,cut_type="number",cut_pm=10)
# Step 2: Use ClusterRowCoverage
temp <- ClusterRowCoverage(result=out2,matrix=mat,noise=0.2,plots=2)
# Step 3: Use BiBitWorkflow again (using previously computed parts) with new cut parameter
out3 <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2,cut_type="number",cut_pm=4,
                      BCresult = out2$info$BiclustInitial,
                      simmatresult = out2$info$BiclustSimInitial)
summary(out3$Biclust)

## End(Not run)
```

bibit_columnextension *Column Extension Procedure*

Description

Function which accepts result from `bibit`, `bibit2` or `bibit3` and will (re-)apply the column extension procedure. This means if the result already contained extended biclusters that these will be deleted.

Usage

```
bibit_columnextension(result, matrix, arff_row_col = NULL, BC = NULL,
  extend_columns = "naive", extend_mincol = 1, extend_limitcol = 1,
  extend_noise = 1, extend_contained = FALSE)
```

Arguments

`result`	Result from `bibit`, `bibit2` or `bibit3`.
`matrix`	The binary input matrix.
`arff_row_col`	The same file directories (with the same limitations) as given in `bibit`, `bibit2` or `bibit3`.
`BC`	A numeric/integer vector of BC's which should be extended. Different behaviour for the 3 types of input results:

`bibit` `BC` directly takes the corresponding biclusters from the result and extends them. (e.g. `BC=c(1,10)` is then remapped to `c("BC1","BC1_Ext1","BC2","BC2_Ext1") in the n`

`bibit2` `BC` corresponds with the original non-extended biclusters from the `bibit2` result. These original biclusters are selected and extended. (e.g. `BC=c(1,10)` selects biclusters `c("BC1","BC10")` which are then remapped to `c("BC1","BC1_Ext1","BC2","BC2`

`bibit3` `BC` corresponds with the biclusters when combining the FULLPATTERN and SUBPATTERN result together. For example choosing `BC=1` would only select the 1 FULLPATTERN bicluster for each pattern and try to extend it. (e.g. `BC=c(1,10)` selects biclusters 1 and 10 from the combined fullpattern and subpattern result (meaning the full pattern BC and the 9th subpattern BC) which are then remapped to `c("BC1","BC1_Ext1","BC2","BC2_Ext1") in the new output)`

`extend_columns` *Column Extension Parameter*
Can be one of the following: `"naive"` or `"recursive"` which will apply either a naive or recursive column extension procedure. (See Details Section for more information.)
Based on the extension, additional biclusters will be created in the Biclust object which can be seen in the column and row names of the `RowxNumber` and `NumberxCol` slots (`"_Ext"` suffix).
The `info` slot will also contain some additional information. Inside this slot, `BC.Extended` contains info on which original biclusters were extended, how many columns were added, and in how many extra extended biclusters this resulted.

Warning: Using a percentage-based `extend_noise` in combination with the recursive procedure will result in a large amount of biclusters and increase the computation time a lot. Depending on the data when using recursive in combination with a noise percentage, it is advised to keep it reasonable small (e.g. 10%). Another remedy is to sufficiently increase the `extend_limitcol` either as a percentage or integer to limit the candidates of columns.

`extend_mincol` *Column Extension Parameter*
A minimum number of columns that a bicluster should be able to be extended with before saving the result. (Default=1)

`extend_limitcol`
Column Extension Parameter
The number (`extend_limitcol>=1`) or percentage (`0<extend_limitcol<1`) of 1's that a column (subsetted on the BC rows) should at least contain for it to be a candidate to be added to the bicluster as an extension. (Default=1) (Increase this parameter if the recursive extension takes too long. Limiting the pool of candidates will decrease computation time, but restrict the results more.)

`extend_noise` *Column Extension Parameter*
The maximum allowed noise (in each row) when extending the columns of the bicluster. Can take the same as the `noise` parameter.

`extend_contained`
Column Extension Parameter
Logical value if extended results should be checked if they contain each other (and deleted if this is the case). Default = `FALSE`. This can be a lengthy procedure for a large amount of biclusters (>1000).

Value

A Biclust S4 Class object or bibit3 S3 list Class object

Details - Column Extension

An optional procedure which can be applied *after* applying the BiBit algorithm (with noise) is called *Column Extension*. The procedure will add extra columns to a BiBit bicluster, keeping into account the allowed `extend_noise` level in each row. The primary goal is to, after applying BiBit with noise, to also try and add some noise to the 2 initial 'perfect' rows. Other parameters like `extend_mincol` and `extend_limitcol` can also further restrict which extensions should be discovered.
This procedure can be done either *naively* (fast) or *recursively* (more slow and thorough) with the `extend_columns` parameter.

`"naive"` Subsetting on the bicluster rows, the column candidates are ordered based on the most 1's in a column. Afterwards, in this order, each column is sequentially checked and added when the resulted BC is still within row noise levels.
This has 2 major consequences:

- If 2 columns are identical, the first in the dataset is added, while the second isn't (depending on the noise level allowed per row).
- If 2 non-identical columns are viable to be added (correct row noise), the column with the most 1's is added. Afterwards the second column might not be viable anymore.

Note that using this method will always result in a maximum of 1 extended bicluster per original bicluster.

`"recursive"` Conditioning the group of candidates for the allowed row noise level, each possible/allowed combination of adding columns to the bicluster is checked. Only the resulted biclusters with the highest number of extra columns are saved. Of course this could result in multiple extensions for 1 bicluster if there are multiple 'maximum added columns' results.

Note: These procedures are followed by a fast check if the extensions resulted in any duplicate biclusters. If so, these are deleted from the final result.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:

set.seed(1)
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

result <- bibit2(data,minr=5,minc=5,noise=0.1,extend_columns = "recursive",
              extend_mincol=1,extend_limitcol=1)
result
```

```
result2 <- bibit_columnextension(result=out,matrix=data,arff_row_col=NULL,BC=c(1,10),
                                 extend_columns="recursive",extend_mincol=1,
                                 extend_limitcol=1,extend_noise=2,extend_contained=FALSE)
result2

## End(Not run)
```

`ClusterRowCoverage` *Row Coverage Plots*

Description

Plotting function to be used with the `BiBitWorkflow` output. It plots the number of clusters (of the hierarchical tree) versus the number/percentage of row coverage and number of final biclusters (see Details for more information).

Usage

```
ClusterRowCoverage(result, matrix, maxCluster = 20, noise = 0.1,
  noise_select = 0, plots = c(1:3), verbose = TRUE,
  plot.type = "device", filename = "RowCoverage")
```

Arguments

`result`	A BiBitWorkflow Object.
`matrix`	Accompanying binary data matrix which was used to obtain `result`.
`maxCluster`	Maximum number of clusters to cut the tree at (default=20).
`noise`	The allowed noise level when growing the rows on the merged patterns after cutting the tree. (default=`0.1`, namely allow 10% noise.) • `noise=0`: No noise allowed. • `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of `5`, the number of allowed 0's would be `1`. • `noise>=1`: The `noise` parameter will be the number of allowed 0's in a row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.
`noise_select`	Should the allowed noise level be automatically selected for each pattern? (Using ad hoc method to find the elbow/kink in the Noise Scree plots) • `noise_select=0`: Do *NOT* automatically select the noise levels. Use the the noise level given in the `noise` parameter (default) • `noise_select=1`: Using the Noise Scree plot (with 'Added Rows' on the y-axis), find the noise level where the current number of added rows at this noise level is larger than the mean of 'added rows' at the lower noise levels. After locating this noise level, lower the noise level by 1. This is your automatically selected elbow/kink and therefore your noise level.

- `noise_select=2`: Applies the same steps as for `noise_select=1`, but instead of decreasing the noise level by only 1, keep decreasing the noise level until the number of added rows isn't decreasing anymore either.

`plots` Vector for which plots to draw:

1. Number of Clusters versus Row Coverage Percentage
2. Number of Clusters versus Number of Row Coverage
3. Number of Clusters versus Final Number of Biclusters

`verbose` Logical value if the progress bar of merging/growing the biclusters should be shown. (default=`TRUE`)

`plot.type` Output Type

- `"device"`: All plots are outputted to new R graphics devices (default).
- `"file"`: All plots are saved in external files. Plots are joint together in a single `.pdf` file.
- `"other"`: All plots are outputted to the current graphics device, but will overwrite each other. Use this if you want to include one or more plots in a sweave/knitr file or if you want to export a single plot by your own chosen format.

`filename` Base filename (with/without directory) for the plots if `plot.type="file"` (default=`"RowCoverage"`).

Details

The graph of number of chosen tree clusters versus the final row coverage can help you to make a decision on how many clusters to choose in the hierarchical tree. The more clusters you choose, the smaller (albeit more similar) the patterns are and the more rows will fit your patterns (i.e. more row coverage).

Value

A data frame containing the number of clusters and the corresponding number of row coverage, percentage of row coverage and the number of final biclusters.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
## Prepare some data ##
set.seed(254)
mat <- matrix(sample(c(0,1),5000*50,replace=TRUE,prob=c(1-0.15,0.15)),
              nrow=5000,ncol=50)
mat[1:200,1:10] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                          nrow=200,ncol=10)
mat[300:399,6:15] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                            nrow=100,ncol=10)
mat[400:599,21:30] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=200,ncol=10)
mat[700:799,29:38] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=100,ncol=10)
mat <- mat[sample(1:5000,5000,replace=FALSE),sample(1:50,50,replace=FALSE)]

## Apply BiBitWorkflow ##
out <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2,cut_type="number",cut_pm=10)
# Make ClusterRowCoverage Plots
ClusterRowCoverage(result=out,matrix=mat,maxCluster=20,noise=0.2)

## End(Not run)
```

`ColInfo` *Column Info of Biclusters*

Description

Function that returns which column labels are part of the pattern derived from the biclusters. Additionally, a biclustmember plot and a general barplot of the column labels (retrieved from the biclusters) can be drawn.

Usage

```
ColInfo(result, matrix, plots = c(1, 2), plot.type = "device",
  filename = "ColInfo")
```

Arguments

`result`	A Biclust Object.
`matrix`	Accompanying data matrix which was used to obtain `result`.
`plots`	Which plots to draw: 1. Barplot of number of appearances of column labels in bicluster results. 2. Biclustmember plot of BC results (see `biclustmember`).
`plot.type`	Output Type • `"device"`: All plots are outputted to new R graphics devices (default). • `"file"`: All plots are saved in external files. Plots are joint together in a single `.pdf` file. • `"other"`: All plots are outputted to the current graphics device, but will overwrite each other. Use this if you want to include one or more plots in a sweave/knitr file or if you want to export a single plot by your own chosen format.
`filename`	Base filename (with/without directory) for the plots if `plot.type="file"` (default=`"RowCoverage"`).

Value

A list object (length equal to number of Biclusters) in which vectors of column labels are saved.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit(data,minr=5,minc=5)
ColInfo(result=result,matrix=data)

## End(Not run)
```

`ColNoiseBC` *Barplots of Column Noise for Biclusters*

Description

Draws barplots of column noise of chosen biclusters. This plot can be helpful in determining which column label is often zero in noisy biclusters.

Usage

```
ColNoiseBC(result, matrix, BC = 1:result@Number, plot.type = "device",
  filename = "ColNoise")
```

Arguments

`result`	A Biclust Object.
`matrix`	Accompanying binary data matrix which was used to obtain `result`.
`BC`	Numeric vector to select of which BC's a column noise bar plot should be drawn.
`plot.type`	Output Type • `"device"`: All plots are outputted to new R graphics devices (default). • `"file"`: All plots are saved in external files. Plots are joint together in a single `.pdf` file. • `"other"`: All plots are outputted to the current graphics device, but will overwrite each other. Use this if you want to include one or more plots in a sweave/knitr file or if you want to export a single plot by your own chosen format.
`filename`	Base filename (with/without directory) for the plots if `plot.type="file"` (default=`"RowCoverage"`).

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit2(data,minr=5,minc=5,noise=1)
ColNoiseBC(result=result,matrix=data,BC=1:3)

## End(Not run)
```

`CompareResultJI` *Compare Biclustering Results using Jaccard Index*

Description

Creates a heatmap and returns a similarity matrix of the Jaccard Index (Row, Column or both dimensions) in order to compare 2 different biclustering results or compare the biclusters of a single result.

Usage

```
CompareResultJI(BCresult1, BCresult2 = NULL, type = "both", plot = TRUE)
```

Arguments

`BCresult1`	A S4 Biclust object. If only this input Biclust object is given, the biclusters of this single result will be compared.
`BCresult2`	A second S4 Biclust object to which `BCresult1` should be compared. (default=`NULL`)
`type`	Of which dimension should the Jaccard Index be computed? Can be `"row"`, `"col"` or `"both"` (default).
`plot`	Logical value if plot should be outputted (default=`TRUE`).

Details

The Jaccard Index between two biclusters is calculated as following:

$$JI(BC1,BC2) = \frac{(m_1 + m_2 - m_{12})}{m_{12}}$$

in which

- `type="row"` or `type="col"`
  - $m_1 =$ Number of rows/columns of BC1
  - $m_2 =$ Number of rows/columns of BC2
  - $m_{12} =$ Number of rows/columns of union of row/column membership of BC1 and BC2
- `type="both"`
  - $m_1 =$ Size of BC1 (rows times columns)
  - $m_2 =$ Size of BC2 (rows times columns)
  - $m_{12} = m_1 + m_2 -$ size of overlapping BC of BC1 and BC2

Value

A list containing

- `SimMat`: The JI Similarity Matrix between the compared biclusters.
- `MaxSim`: A list containing the maximum values on each row (`BCResult1`) and each column (`BCResult2`).

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

# Result 1
result1 <- bibit(data,minr=5,minc=5)
result1

# Result 2
result2 <- bibit(data,minr=2,minc=2)
result2

## Compare all BC's of Result 1 ##
Sim1 <- CompareResultJI(BCresult1=result1,type="both")
Sim1$SimMat

## Compare BC's of Result 1 and 2 ##
Sim12 <- CompareResultJI(BCresult1=result1,BCresult2=result2,type="both",plot=FALSE)
str(Sim12)

## End(Not run)
```

make_arff_row_col *Transform R matrix object to BiBit input files.*

Description

Transform the R matrix object to 1 .arff for the data and 2 .csv files for the row and column names. These are the 3 files required for the original BiBit Java algorithm The path of these 3 files can then be used in the arff_row_col parameter of the bibit function.

Usage

```
make_arff_row_col(matrix, name = "data", path = "")
```

Arguments

matrix	The binary input matrix.
name	Basename for the 3 input files.
path	Directory path where to write the 3 input files to.

Value

3 input files for BiBit:

- One .arff file containing the data.
- One .csv file for the row names. The file contains 1 column of names without quotation.
- One .csv file for the column names. The file contains 1 column of names without quotation.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]

make_arff_row_col(matrix=data,name="data",path="")

result <- bibit(data,minr=5,minc=5,
               arff_row_col=c("data_arff.arff","data_rownames.csv","data_colnames.csv"))

## End(Not run)
```

MaxBC *Finding Maximum Size Biclusters*

Description

Simple function which scans a `Biclust` result and returns which biclusters have maximum row, column or size (row*column).

Usage

```
MaxBC(result, top = 1)
```

Arguments

`result`	A `Biclust` result. (e.g. The return object from `bibit` or `bibit2`)
`top`	The number of top row/col/size dimension which are searched for. (e.g. default `top=1` gives only the maximum)

Value

A list containing:

- `$row`: A matrix containing in the columns the Biclusters which had maximum rows, and in the rows the Row Dimension, Column Dimension and Size.
- `$column`: A matrix containing in the columns the Biclusters which had maximum columns, and in the rows the Row Dimension, Column Dimension and Size.
- `$size`: A matrix containing in the columns the Biclusters which had maximum size, and in the rows the Row Dimension, Column Dimension and Size.

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
data <- matrix(sample(c(0,1),100*100,replace=TRUE,prob=c(0.9,0.1)),nrow=100,ncol=100)
data[1:10,1:10] <- 1 # BC1
data[11:20,11:20] <- 1 # BC2
data[21:30,21:30] <- 1 # BC3
data <- data[sample(1:nrow(data),nrow(data)),sample(1:ncol(data),ncol(data))]
result <- bibit(data,minr=2,minc=2)

MaxBC(result)


## End(Not run)
```

NoiseScree *Noise Scree Plots*

Description

Extract patterns from either a Biclust or BiBitWorkflow object (see Details) and plot the Noise Scree plot (same as plot 4 in `BiBitWorkflow`). Additionally, if `FisherResult` is available (from `RowTest_Fisher`), this info will be added to the plot.

Usage

```
NoiseScree(result, matrix, type = c("Added", "Total"), pattern = NULL,
  noise_select = 0, alpha = 0.05)
```

Arguments

`result`	A Biclust or BiBitWorkflow Object.
`matrix`	Accompanying binary data matrix which was used to obtain `result`.
`type`	Either `"Added"` or `"Total"`. Should the noise level be plotted against the number of added rows (at that noise level) or the total number of rows (up to that noise level)?
`pattern`	Numeric vector for which patterns the noise scree plot should be drawn (default = all patterns).
`noise_select`	Should an automatic noise selection be applied and drawn (blue vertical line) on the plot? (Using ad hoc method to find the elbow/kink in the Noise Scree plots) • `noise_select=0`: No noise selection is applied and no line is drawn (default). • `noise_select=1`: Using the Noise Scree plot (with 'Added Rows' on the y-axis), find the noise level where the current number of added rows at this noise level is larger than the mean of 'added rows' at the lower noise levels. After locating this noise level, lower the noise level by 1. This is your automatically selected elbow/kink and therefore your noise level. • `noise_select=2`: Applies the same steps as for `noise_select=1`, but instead of decreasing the noise level by only 1, keep decreasing the noise level until the number of added rows isn't decreasing anymore either.
`alpha`	If info from the Fisher Exact test is available, which significance level should be used to in the plot (Noise versus Significant Fisher Exact Test rows). (default=0.05)

Details

Biclust S4 Object Using the column patterns of the Biclust result, the noise level is plotted versus the number of `"Total"` or `"Added"` rows.

BiBitWorkflow S3 Object The merged column patterns (after cutting the hierarchical tree) are extracted from the BiBitWorkflow object, namely the `$info$MergedColPatterns` slot. These patterns are used to plot the noise level versus the number of `"Total"` or `"Added"` rows.

If information on the Fisher Exact Test is available, then this info will added to the plot (noise level versus significant rows).

Value

`NULL`

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
## Prepare some data ##
set.seed(254)
mat <- matrix(sample(c(0,1),5000*50,replace=TRUE,prob=c(1-0.15,0.15)),
              nrow=5000,ncol=50)
mat[1:200,1:10] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                          nrow=200,ncol=10)
mat[300:399,6:15] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                            nrow=100,ncol=10)
mat[400:599,21:30] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=200,ncol=10)
mat[700:799,29:38] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=100,ncol=10)
mat <- mat[sample(1:5000,5000,replace=FALSE),sample(1:50,50,replace=FALSE)]

## Apply BiBitWorkflow ##
out <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2,cut_type="number",cut_pm=4)
# Make Noise Scree Plot - Default
NoiseScree(result=out,matrix=mat,type="Added")
NoiseScree(result=out,matrix=mat,type="Total")
# Make Noise Scree Plot - Use Automatic Noies Selection
NoiseScree(result=out,matrix=mat,type="Added",noise_select=2)
NoiseScree(result=out,matrix=mat,type="Total",noise_select=2)

## Apply RowTest_Fisher on BiBitWorkflow Object ##
out2 <- RowTest_Fisher(result=out,matrix=mat)
# Fisher output is added to "NoiseScree" plot
NoiseScree(result=out2,matrix=mat,type="Added")
NoiseScree(result=out2,matrix=mat,type="Total")

## End(Not run)
```

`RowTest_Fisher` *Apply Fisher Exact Test on Bicluster Rows*

Description

Accepts a Biclust or BiBitWorkflow result and applies the Fisher Exact Test for each row (see Details).

Usage

```
RowTest_Fisher(result, matrix, p.adjust = "BH", alpha = 0.05,
  pattern = NULL)
```

Arguments

`result`	A Biclust or BiBitWorkflow Object.
`matrix`	Accompanying binary data matrix which was used to obtain `result`.
`p.adjust`	Which method to use when adjusting p-values, see `p.adjust` (default=`"BH"`).
`alpha`	Significance level (adjusted p-values) when constructing the `FisherInfo` object (default=0.05).
`pattern`	Numeric vector for which patterns/biclusters the Fisher Exact Test needs to be computed (default = all patterns/biclusters).

Details

Extracts the patterns from either a `Biclust` or `BiBitWorkflow` object (see below). Afterwards for each pattern all rows will be tested using the Fisher Exact Test. This test compares the part of the row inside the pattern (of the bicluster) with the part of the row outside the pattern. The Fisher Exact Test gives you some information on if the row is uniquely active for this pattern.

Depending on the `result` input, different patterns will be extract and different info will be returned:

Biclust S4 Object Using the column patterns of the Biclust result, all rows are tested using the Fisher Exact Test. Afterwards the following 2 objects are added to the `info` slot of the Biclust object:

- `FisherResult`: A list object (one element for each pattern) of data frames (Number of Rows $\times$ 6) which contain the names of the rows (`Names`), the noise level of the row inside the pattern (`Noise`), the signal percentage inside the pattern (`InsidePerc1`), the signal percentage outside the pattern (`OutsidePerc1`), the p-value of the Fisher Exact Test (`Fisher_pvalue`) and the adjusted p-value of the Fisher Exact Test (`Fisher_pvalue_adj`).
- `FisherInfo`: Info object which contains a comparison of the current row membership for each pattern with a 'new' row membership based on the significant rows (from the Fisher Exact Test) for each pattern. It is a list object (one element for each pattern) of lists (6 elements). These list objects per pattern contain the number of new, removed and identical rows (`NewRows`, `RemovedRows`, `SameRows`) when comparing the significant rows with the original row membership (as well as their indices (`NewRows_index`, `RemovedRows_index`)). The `MaxNoise` element contains the maximum noise of all Fisher significant rows.

BiBitWorkflow S3 Object The merged column patterns (after cutting the hierarchical tree) are extracted from the BiBitWorkflow object, namely the `$info$MergedColPatterns` slot. Afterwards the following object is added to the `$info` slot of the BiBitWorkflow object:

- `FisherResult`: Same as above

Value

Depending on `result`, a `FisherResult` and/or `FisherInfo` object will be added to the `result` and returned (see Details).

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
## Prepare some data ##
set.seed(254)
mat <- matrix(sample(c(0,1),5000*50,replace=TRUE,prob=c(1-0.15,0.15)),
              nrow=5000,ncol=50)
mat[1:200,1:10] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                          nrow=200,ncol=10)
mat[300:399,6:15] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                            nrow=100,ncol=10)
mat[400:599,21:30] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=200,ncol=10)
mat[700:799,29:38] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=100,ncol=10)
mat <- mat[sample(1:5000,5000,replace=FALSE),sample(1:50,50,replace=FALSE)]

## Apply BiBitWorkflow ##
out <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.2,cut_type="number",cut_pm=4)

## Apply RowTest_Fisher on Biclust Object -> returns Biclust Object ##
out_new <- RowTest_Fisher(result=out$Biclust,matrix=mat)
# FisherResult output in info slot
str(out_new@info$FisherResult)
# FisherInfo output in info slot (comparison with original BC's)
str(out_new@info$FisherInfo)


## Apply RowTest_Fisher on BiBitWorkflow Object -> returns BiBitWorkflow Object ##
out_new2 <- RowTest_Fisher(result=out,matrix=mat)
# FisherResult output in BiBitWorkflow info element
str(out_new2$info$FisherResult)
# Fisher output is added to "NoiseScree" plot
NoiseScree(result=out_new2,matrix=mat,type="Added")

## End(Not run)
```

summary,Biclust-method
Summary Method for Biclust Class

Description

Summary Method for Biclust Class

Usage

```
## S4 method for signature 'Biclust'
summary(object)
```

Arguments

object	Biclust S4 Object

UpdateBiclust_RowNoise
Update a Biclust or BiBitWorkflow Object with a new Noise Level

Description

Apply a new noise level on a Biclust object result or BiBitWorkflow result. See Details on how both objects are affected.

Usage

```
UpdateBiclust_RowNoise(result, matrix, noise = 0.1, noise_select = 0,
  removeBC = FALSE)
```

Arguments

result — A Biclust or BiBitWorkflow Object.

matrix — Accompanying binary data matrix which was used to obtain `result`.

noise — The new noise level which should be used in the rows of the biclusters. (default=`0.1`, namely allow 10% noise.).

- `noise=0`: No noise allowed.
- `0<noise<1`: The `noise` parameter will be a noise percentage. The number of allowed 0's in a row in the bicluster will depend on the column size of the bicluster. More specifically `zeros_allowed = ceiling(noise * columnsize)`. For example for `noise=0.10` and a bicluster column size of `5`, the number of allowed 0's would be `1`.

- `noise>=1`: The `noise` parameter will be the number of allowed 0's in a row in the bicluster independent from the column size of the bicluster. In this noise option, the noise parameter should be an integer.

`noise_select` Should the allowed noise level be automatically selected for each pattern? (Using ad hoc method to find the elbow/kink in the Noise Scree plots)

- `noise_select=0`: Do *NOT* automatically select the noise levels. Use the the noise level given in the `noise` parameter (default)
- `noise_select=1`: Using the Noise Scree plot (with 'Added Rows' on the y-axis), find the noise level where the current number of added rows at this noise level is larger than the mean of 'added rows' at the lower noise levels. After locating this noise level, lower the noise level by 1. This is your automatically selected elbow/kink and therefore your noise level.
- `noise_select=2`: Applies the same steps as for `noise_select=1`, but instead of decreasing the noise level by only 1, keep decreasing the noise level until the number of added rows isn't decreasing anymore either.

`removeBC` *(Only applicable when result is a Biclust object)* Logical value if after applying a new noise level, duplicate and non-maximal BC's should be deleted.

Details

Biclust S4 Object Using the column patterns of the Biclust result, new grows are grown using the inputted `noise` level. The `removeBC` parameter decides if duplicate and non-maximal BC's should be deleted. Afterwards a new `Biclust` S4 object is returned with the new biclusters.

BiBitWorkflow S3 Object The merged column patterns (after cutting the hierarchical tree) are extracted from the BiBitWorkflow object, namely the `$info$MergedColPatterns` slot. Afterwards, using the new `noise` level, new rows are grown and the returned object is an updated `BiBitWorkflow` object. (e.g. The final Biclust slot, MergedNoiseThresholds, coverage,etc. are updated)

Value

A `Biclust` or `BiBitWorkflow` Object (See Details)

Author(s)

Ewoud De Troyer

Examples

```
## Not run:
## Prepare some data ##
set.seed(254)
mat <- matrix(sample(c(0,1),5000*50,replace=TRUE,prob=c(1-0.15,0.15)),
              nrow=5000,ncol=50)
mat[1:200,1:10] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                          nrow=200,ncol=10)
mat[300:399,6:15] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                            nrow=100,ncol=10)
mat[400:599,21:30] <- matrix(sample(c(0,1),200*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=200,ncol=10)
mat[700:799,29:38] <- matrix(sample(c(0,1),100*10,replace=TRUE,prob=c(1-0.9,0.9)),
                             nrow=100,ncol=10)
mat <- mat[sample(1:5000,5000,replace=FALSE),sample(1:50,50,replace=FALSE)]

## Apply BiBitWorkflow ##
out <- BiBitWorkflow(matrix=mat,minr=50,minc=5,noise=0.1,cut_type="number",cut_pm=4)
summary(out$Biclust)

## Update Rows with new noise level on Biclust Obect -> returns Biclust Object ##
out_new <- UpdateBiclust_RowNoise(result=out$Biclust,matrix=mat,noise=0.3)
summary(out_new)
out_new@info$Noise.Threshold # New Noise Levels

## Update Rows with new noise level on BiBitWorkflow Obect -> returns BiBitWorkflow Object ##
out_new2 <- UpdateBiclust_RowNoise(result=out,matrix=mat,noise=0.2)
summary(out_new2$Biclust)
out_new2$info$MergedNoiseThresholds # New Noise Levels

## End(Not run)
```

Index