Package ‘runibic’

December 6, 2025

Type Package

Title runibic: row-based biclustering algorithm for analysis of gene expression data in R

Version 1.33.0

Author Patryk Orzechowski, Artur Pańszczyk

Maintainer Patryk Orzechowski <`patryk.orzechowski@gmail.com`>

Description This package implements UbiBic algorithm in R. This biclustering algorithm for analysis of gene expression data was introduced by Zhenjia Wang et al. in 2016. It is currently considered the most promising biclustering method for identification of meaningful structures in complex and noisy data.

Depends R (>= 3.4.0), biclust, SummarizedExperiment

Imports Rcpp (>= 0.12.12), testthat, methods

Suggests knitr, rmarkdown, GEOquery, affy, airway, QUBIC

VignetteBuilder knitr

LinkingTo Rcpp

SystemRequirements C++11, GNU make

biocViews Microarray, Clustering, GeneExpression, Sequencing, Coverage

Encoding UTF-8

License MIT + file LICENSE

LazyData true

NeedsCompilation yes

URL `http://github.com/athril/runibic`

BugReports `http://github.com/athril/runibic/issues`

RoxygenNote 6.0.1

git_url https://git.bioconductor.org/packages/runibic

git_branch devel

git_last_commit 3cbdc87

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-12-05

Contents

`backtrackLCS` *Retrieving a Longest Common Subsequence between two integer vectors.*

Description

This function retrieves the Longest Common Subsequence (LCS) between two integer vectors by backtracking the matrix obtained with dynamic programming.

Usage

```
backtrackLCS(x, y)
```

Arguments

`x`	an integer vector
`y`	an integer vector

Value

an integer vector containing the the Longest Common Subsequence (LCS) between vectors x and y (i.e. the values that appear in both x and y in the same order)

See Also

`runibic` `pairwiseLCS` `calculateLCS`

Examples

```
A <- c(1, 2, 3, 4, 5)
B <- c(1, 2, 4)
backtrackLCS(A, B)
```

`BCUnibic-class` *Class BCUnibic*

Description

An S4 class to represent `BCUnibic-class` UniBic biclustering algorithm for numeric input. The class is intended to use with

See Also

`runibic`

`BCUnibicD-class` *Class BCUnibicD*

Description

An S4 class `BCUnibicD-class` defines UniBic biclustering algorithm for discrete input.

`calculateLCS` *Calculate all Longest Common Subsequences between a matrix.*

Description

This function computes unique pairwise Longest Common Subsequences between each row of input matrix. The function outputs a list sorted by Longest Common Subsequences (LCS) length. The output list contains the length of calculated LCS, indices, of the first and second rows between which LCS was calculated. The function uses two different sorting methods. The default one uses Fibonacci Heap used in original implementation of Unibic, the second one uses standard sorting algorithm from C++ STL.

Usage

```
calculateLCS(discreteInput, useFibHeap = TRUE)
```

Arguments

`discreteInput`	is a input discrete matrix
`useFibHeap`	boolean value for choosing which sorting method should be used in sorting of output

Value

a list with sorted values based on calculation of the length of LCS between all pairs of rows

See Also

`runibic` `backtrackLCS` `pairwiseLCS`

Examples

```
A <- matrix(c(4, 3, 1, 2, 5, 8, 6, 7), nrow=2, byrow=TRUE)
calculateLCS(A, TRUE)
```

`cluster` *Calculate biclusters from sorted list of LCS scores and row indices*

Description

This function search for biclusters in the input matrix. The calculations are based on the integer matrix with indexes indicating positions of j-th smallest element in each row and the results from calculations of Longest Common Subsequence between all rows in the input matrix. The paramteres of this function can be obtained from other functions provided by this package.

Usage

```
cluster(discreteInput, discreteInputValues, scores, geneOne, geneTwo, rowNumber,
  colNumber)
```

Arguments

`discreteInput`	an integer matrix with indices of sorted columns
`discreteInputValues`	an integer matrix with discrete values
`scores`	a numeric vector with LCS length
`geneOne`	a numeric vector with first row indexes from pairwise LCS calculation
`geneTwo`	a numeric vector with second row indexes from pairwise LCS calculation
`rowNumber`	a int with number of rows in the input matrix
`colNumber`	a int with number of columns in the input matrix

Value

a list with information of found biclusters

See Also

`runibic` `calculateLCS` `unisort`

Examples

```
A <- matrix( c(4,3,1,2,5,8,6,7,9,10,11,12),nrow=4,byrow=TRUE)
iA <- unisort(A)
lcsResults <- calculateLCS(A)
cluster(iA, A, lcsResults$lcslen, lcsResults$a, lcsResults$b, nrow(A), ncol(A))
```

`pairwiseLCS` *Calculate a matrix of Longest Common Subsequence (LCS) between a pair of numeric vectors*

Description

This function calculates the matrix with Longest Common Subsequence (LCS) between two numeric vectors. From given matrix we can locate the size of the Longest Common Subsequence in the last column in the last row.

Usage

```
pairwiseLCS(x, y)
```

Arguments

`x`	an integer vector
`y`	an integer vector

Value

a matrix computed using dynamic programming that stores the Longest Common Subsequence (LCS) between two vectors A and B.

See Also

`runibic` `calculateLCS` `backtrackLCS`

Examples

```
A <- c(1, 2, 3, 4, 5)
B <- c(1, 2, 4)
pairwiseLCS(A, B)
```

`runibic` *runibic: parallel row-based biclustering algorithm for analysis of gene expression data in R*

Description

`runibic` is a package that contains much faster parallel version of one of the most accurate biclustering algorithms, UniBic. The original method was reimplemented from C to C++11, OpenMP was added for parallelization.

If you use this package, please cite it as: Patryk Orzechowski, Artur Pańszczyk, Xiuzhen Huang, Jason H Moore; "runibic: a Bioconductor package for parallel row-based biclustering of gene expression data"; Bioinformatics, 2018, bty512, doi: https://doi.org/10.1093/bioinformatics/bty512

Each of the following functions `BCUnibic`, `BCUnibicD`, `runibic` perform biclustering using UniBic biclustering algorithm. The major difference between the functions is that `BCUnibicD` require a discretized matrix, whilst `BCUnibic` (or `runibic`) could be applied to numeric one.

Usage

```
BCUnibic(x = NULL, t = 0.95, q = 0, f = 1, nbic = 100, div = 0,
  useLegacy = FALSE)

BCUnibicD(x = NULL, t = 0.95, q = 0, f = 1, nbic = 100, div = 0,
  useLegacy = FALSE)

runibic(x = NULL, t = 0.95, q = 0, f = 1, nbic = 100, div = 0, useLegacy=FALSE)
```

Arguments

`x`	numeric or integer matrix (depends on the function)
`t`	consistency level of the block (0.5-1.0].
`q`	a double value for quantile discretization
`f`	filtering overlapping blocks (default 1 do not remove any blocks)
`nbic`	maximum number of biclusters in output
`div`	number of ranks for up(down)-regulated genes: default: 0==ncol(x)
`useLegacy`	boolean value for using legacy parameter settings

Details

For a given input matrix we first perform discretization and create index matrix using `runiDiscretize` function. The discretization is performed taking into account quantiles of the data. The resulting index matrix allows to detect order-preserving trends between each pair of the rows irrespective to the order of columns. After the ranking, the matrix is split by rows into subgroups based on the significance of the future biclusters. In each of the chunks, we calculate pairwise calculations of Longest Common Subsequence LCS between all pairs of the rows. LCS calculations are performed using dynamic programming and determine the longest order-preserving trend between each pair of the rows. After partitioning the matrix strict order-preserving biclusters are determined and later expanded to approximate-trend biclusters within `cluster` function.

This package provides 3 main functions: `runibic` and `BCUnibic` perform UniBic biclustering algorithm on numeric data, whilst `BCUnibicD` could be applied to integer ones. The latter two methods are compatible with `Biclust` class.

Value

`Biclust` object with detected biclusters

Functions

- `BCUnibic`: `BCUnibic` performs biclustering using UniBic on numeric matrix. It is intended to use as a method called from `biclust`.
- `BCUnibicD`: perform biclustering using UniBic on integer matrix. It is intended to use as a method called from `biclust`.
- `runibic`: perform biclustering using UniBic on numeric matrix.

Author(s)

Patryk Orzechowski `patryk.orzechowski@gmail.com`, Artur Pańszczyk `panszczyk.artur@gmail.com`

References

Wang, Zhenjia, et al. "UniBic: Sequential row-based biclustering algorithm for analysis of gene expression data." Scientific reports 6 (2016): 23466.

Patryk Orzechowski, Artur Pańszczyk, Xiuzhen Huang, Jason H. Moore: "runibic: a Bioconductor package for parallel row-based biclustering of gene expression data", bioRxiv (2017): 210682, doi: https://doi.org/10.1101/210682

See Also

`runiDiscretize` `set_runibic_params` `BCUnibic-class` `BCUnibicD-class` `unisort`

Examples

```
A <- matrix(replicate(100, rnorm(100)), nrow=100, byrow=TRUE)
runibic(A)
BCUnibic(A)
BCUnibic(A, t = 0.95, q = 0, f = 1, nbic = 100, div = 0)
B <- runiDiscretize(A)
runibic(B)
BCUnibicD(B, t = 0.95, q = 0, f = 1, nbic = 100, div = 0)
biclust::biclust(A, method=BCUnibic(), t = 0.95, q = 0, f = 1, nbic = 100, div = 0)
biclust::biclust(B, method=BCUnibicD(), t = 0.95, q = 0, f = 1, nbic = 100, div = 0)
```

`runiDiscretize` *Discretize an input matrix*

Description

This function discretizes the input matrix. `runiDiscretize` uses paramaters: 'div' and 'q', which are set by set_runibic_params function. The funtion returns a discrete matrix with given number of ranks based on the parameter div. In contrast to biclust::discretize the function takes into consideration the quantile parameter 'q'. When 'q' parameter is higher or equal 0.5 a simple discretization is used with equal sizes of the levels using the quantiles. If 'q' parameter is lower than 0.5 we use up(down)-regulated discretization divided into three parts.

Usage

```
runiDiscretize(x)
```

Arguments

`x`	a numeric matrix

Value

a discretized matrix containing integers only

See Also

`set_runibic_params` `calculateLCS` `discretize`

Examples

```
A <- replicate(10, rnorm(20))
runiDiscretize(A)
```

`set_runibic_params` *Set the parameters for runibic algorithm*

Description

runibic function for setting parameters

Usage

```
set_runibic_params(t = 0.85, q = 0, f = 1, nbic = 100L, div = 0L,
  useLegacy = FALSE)
```

Arguments

`t`	consistency level of the block (0.5-1.0]
`q`	a double value for quantile discretization
`f`	filtering overlapping blocks, default 1(do not remove any blocks)
`nbic`	maximum number of biclusters in output
`div`	number of ranks as which we treat the up(down)-regulated value: default: 0==ncol(x)
`useLegacy`	boolean value for legacy parameters management

Value

NULL (an empty value)

See Also

`runibic`

Examples

```
set_runibic_params(0.85, 0, 1, 100, 0, FALSE)
```

`unisort` *Computing the indexes of j-th smallest values of each row*

Description

This function sorts separately each row of a integer matrix and returns a matrix in which the value in i-th row and j-th column represent the index of the j-th smallest value of the i-th row.

Usage

```
unisort(x)
```

Arguments

`x`	a integer matrix

Value

a integer matrix with indexes indicating positions of j-th smallest element in each row

See Also

`runibic` `calculateLCS` `runiDiscretize`

Examples

```
A <- matrix(c(4, 3, 1, 2, 5, 8, 6, 7), nrow=2, byrow=TRUE)
unisort(A)
```

Index