Package ‘poseticDataAnalysis’

October 23, 2025

Type Package

Title Posetic Data Analysis

Version 0.1.0

Maintainer Alessandro Avellone <alessandro.avellone@unimib.it>

Description Build and manipulate partially ordered sets (posets), to perform some data analysis on them and to implement multi-criteria decision making procedures. Several efficient ways for generating linear extensions are implemented, together with functions for building mutual ranking probabilities, incomparability, dominance and separation scores (Fattore, M., De Capitani, L., Avellone, A., Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. ANNALS OF OPERATIONS RESEARCH <doi:10.1007/s10479-024-06352-3>).

License GPL (>= 2)

Imports methods

SystemRequirements C++20

RoxygenNote 7.3.2

NeedsCompilation yes

Encoding UTF-8

Repository CRAN

Collate '00package-exports.R' '00package-class.R' 'IsSymmetric.R' 'IsReflexive.R' 'IsAntisymmetric.R' 'IsTransitive.R' 'IsPreorder.R' 'IsPartialOrder.R' 'TransitiveClosure.R' 'ReflexiveClosure.R' 'POSet.R' 'LinearPOSet.R' 'ProductPOSet.R' 'BinaryVariablePOSet.R' 'IntersectionPOSet.R' 'LinearSumPOSet.R' 'DisjointSumPOSet.R' 'LiftingPOSet.R' 'CrownPOSet.R' 'FencePOSet.R' 'DualPOSet.R' 'LexicographicProductPOSet.R' 'POSetElements.R' 'DominanceMatrix.R' 'IsDominatedBy.R' 'Dominates.R' 'IsComparableWith.R' 'IsIncomparableWith.R' 'UpsetOf.R' 'IsUpset.R' 'DownsetOf.R' 'IsDownset.R' 'ComparabilitySetOf.R' 'IncomparabilitySetOf.R' 'POSetMinimals.R' 'POSetMaximals.R' 'IsMinimal.R' 'IsMaximal.R' 'CoverRelation.R' 'CoverMatrix.R' 'OrderRelation.R' 'IncomparabilityRelation.R' 'POSetMeet.R'

'POSetJoin.R' 'IsExtensionOf.R' 'LEGenerator.R'
'LEBubleyDyer.R' 'LEGet.R' 'ExactMRP.R'
'BubleyDyerMRPGenerator.R' 'BubleyDyerMRP.R'
'ExactEvaluation.R' 'BuildBubleyDyerEvaluationGenerator.R'
'BubleyDyerEvaluation.R' 'BLSDominance.R'
'BubleyDyerSeparation.R' 'BuildBubleyDyerSeparationGenerator.R'
'ExactSeparation.R' 'LexSeparation.R' 'LexMRP.R'
'FuzzyInBetweenness.R' 'FuzzyInBetweennessMinMax.R'
'FuzzyInBetweennessProbabilistic.R' 'FuzzySeparation.R'
'FuzzySeparationMinMax.R' 'FuzzySeparationProbabilistic.R'
'OptimalBidimensionalEmbedding.R'
'BidimentionalPosetRepresentation.R'

Author Alessandro Avellone [aut, cre],
Lucio De Capitani [aut],
Marco Fattore [aut]

Date/Publication 2025-10-23 13:10:03 UTC

Contents

`poseticDataAnalysis-package`
poseticDataAnalysis: Posetic Data Analysis

Description

Build and manipulate partially ordered sets (posets), to perform some data analysis on them and to implement multi-criteria decision making procedures. Several efficient ways for generating linear extensions are implemented, together with functions for building mutual ranking probabilities, incomparability, dominance and separation scores (Fattore, M., De Capitani, L., Avellone, A., Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. ANNALS OF OPERATIONS RESEARCH doi:10.1007/s10479024063523).

Author(s)

Maintainer: Alessandro Avellone <`alessandro.avellone@unimib.it`>

Authors:

- Lucio De Capitani <`lucio.decapitani1@unimib.it`>
- Marco Fattore <`marco.fattore@unimib.it`>

`BidimentionalPosetRepresentation`
Bidimensional representation of multidimensional ordinal binary data generated by a specific reversed pair of lexicographic linear extensions

---

Description

Starting from a dataset related to n statistical units, scored against k ordinal 0/1-indicators and partially ordered component-wise into a Boolean lattice $B_k = (\{0,1\}^k, \leq_{cmp})$, it finds the bidimensional data representation generated by a specific reversed pair of lexicographic linear extensions.

Usage

```
BidimentionalPosetRepresentation(profile, weights, variablesPriority)
```

Arguments

`profile`	Boolean matrix of dimension $m \times k$ of the unique $m \leq n$ different observed profiles. Each observed profile is row of `profile`. Each observed profile is repeated only once in the matrix `profile`.
`weights`	real vector of length m with the frequencies/weights of each observed profile. Element of position j in vector `weights` is the frequency/weight of the profile in row j of `profile`.
`variablesPriority`	integer vector of dimension k containing a permutation $i_1, ..., i_k$ of $1, ..., k$. This vector specifies the criterion to build the reversed pair of lexicographic linear extensions used to approximate B_k. The first linear extension is built by ordering profiles first according to their scores on V_{i_1}, then to the scores on V_{i_2} and so on, until V_{i_k}; the second linear extension is built by ordering profiles first according to their scores on V_{i_k}, then to the scores on $V_{i_{k-1}}$ and so on, until V_{i_1}.

Value

a list of 2 elements named `LossVAlue` and `Representation`.

`LossVAlue` real number indicating the value of the global error $L(D^{out}|D^{inp}, p)$ corresponding to the representation induced by the chosen `variablesPriority`.

`Representation` a data frame with m values (one value for each observed profile) of 5 variables named `profiles`, `x`, `y`, `weights` and `error`. `$profile` is an integer vector containing the base-10 representation of the k-dimensional Boolean vectors representing observed profiles. `$x` is an integer vector containing the x-coordinates of points representing observed profiles in the bidimensional representation. `$y` is an integer vector containing the y-coordinates of points representing observed profiles in the bidimensional representation. `$weights` is a real vector with the frequencies/weights of each observed profile. `$error` is a real vector with the values of the approximation errors $L(b|D^{inp}, p)$ associated to each observed profile in the bidimensional representation.

Examples

```
#SIMULATING OBSERVED BINARY DATA
#number of binary variables
k <- 6
#building observed profiles matrix
profiles <- sapply((0:(2^k-1)) ,function(x){ as.integer(intToBits(x))})
profiles <- t(profiles[1:k, ])
#building the vector of observation frequencies
weights <- sample.int(100, nrow(profiles), replace=TRUE)
#Chosing (at random) a variable priority
vp <- sample.int(k, k, replace=FALSE)
result <- BidimentionalPosetRepresentation(profiles, weights, vp)
```

`BinaryVariablePOSet` *Constructing a component-wise poset of binary vectors.*

Description

Constructs a component-wise poset, starting from a collection of binary variables.

Usage

```
BinaryVariablePOSet(variables)
```

Arguments

`variables`	A vector of character strings (the names of the input binary variables).

Details

Given k input binary variables, the function produces a poset $(V, \leq_{cmp})$, where V is the set of 2^k binary vectors built from the variables, and $\leq_{cmp}$ is the component-wise order.

Value

An object of S4 class `BinarVariablePOSet` (subclass of `POSet`).

Examples

```
vrbs <- c("var1", "var2", "var3")
binPoset <-  BinaryVariablePOSet(variables = vrbs)
```

`BinaryVariablePOSet-class` *An S4 class to represent a Binary Variable POSet.*

Description

An S4 class to represent a Binary Variable POSet.

Slots

`ptr` an external pointer to C++ data

`BLSDominance` *Computing the BLS dominance matrix of a poset.*

Description

Computes the dominance matrix of the input poset, based on the BLS formula of Brueggemann et al. (2003).

Usage

```
BLSDominance(poset)
```

Arguments

`poset`	Object of S4 class `POSet` (the input poset). Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

The BLS dominance matrix

References

Brueggemann R., Lerche D. B., Sørensen P. B. (2003). First attempts to relate structures of Hasse diagrams with mutual probabilities, in: Sørensen P.B., Brueggemann R., Lerche D.B., Voigt K.,Welzl G., Simon U., Abs M., Erfmann M., Carlsen L., Gyldenkærne S., Thomsen M., Fauser P., Mogensen B. B., Pudenz S., Kronvang B. Order Theory in Environmental Sciences Integrative approaches.The 5th workshop held at the National Environmental Research Institute (NERI), Roskilde, Denmark, November 2002. National Environmental Research Institute, Denmark - NERI Technical Report, No. 479.

Examples

```
el <- c("a", "b", "c", "d")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)
pos <- POSet(elements = el, dom = dom)

res <- BLSDominance(pos)
```

`BubleyDyerEvaluation` *Estimating function averages on linear extensions, by the Bubley-Dyer procedure.*

Description

`BubleyDyerEvaluation` computes the averages of the input functions (defined on linear orders) over a subset of linear extensions of the input poset, randomly generated by the Bubley-Dyer procedure.

Usage

```
BubleyDyerEvaluation(
  generator,
  n = NULL,
  error = NULL,
  output_every_sec = NULL
)
```

Arguments

`generator`	S4 object of class `BubleyDyerEvaluationGenerator` created by function `BuildBubleyDyerEvaluatior` implicitly containing the poset and the list of functions, whose averages are to be estimated.
`n`	Number of linear extensions to be generated. See the `Details`, for further information on this parameter.
`error`	A real number in $(0, 1)$, representing the "distance" from uniformity of the sampling distribution of the linear extensions. This parameter is used to determine the number of linear extensions to be sampled, in order to achieve the desired "distance". According to Bubley and Dyer (1999), if `error`$= \epsilon$ and E is the number of elements in the poset, then the number n_ϵ of sampled linear extensions is given by $n_\epsilon = E^4(\ln(E))^2 + E^3 \ln(E) \ln(\epsilon^{-1})$. If both arguments `n` and `error` are specified by the user, the number of linear extensions actually generated is `n`.
`output_every_sec`	Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `BubleyDyerEvaluation`, the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec`seconds.

Details

The function `BubleyDyerEvaluation` allows the user to update previously computed averages, so as to improve estimation accuracy. The `generator` internally stores the averages computed at each call of `BubleyDyerEvaluation`. At the subsequent call (with the same `generator` argument), the previously computed averages are updated, based on the newly sampled linear extensions. In this case, the number of additional linear extensions is determined either directly, by parameter `n`, or indirectly, by specifying parameter `error`, which sets the desired "distance" from uniformity of the sampling distribution of linear extensions, in the Bubley-Dyer procedure. In the latter case, the number of additional linear extensions is computed as $n_\epsilon - n_a$, where n_ϵ is the number of linear extensions necessary to achive the desired "distance" and n_a is the total number of linear extensions generated in the previous calls of `BubleyDyerEvaluation`. If $n_\epsilon - n_a \leq 0$, no further linear extensions are generated and a warning message is displayed.

In case new function averages are desired, run `BubleyDyerEvaluation` with a `generator` argument newly generated by function `BuildBubleyDyerEvaluationGenerator`.

Value

List of the estimated averages, along with the number of linear extensions used to compute them.

References

Bubley, R., Dyer, M. (1999). Faster random generation of linear extensions. Discrete Mathematics, 201, 81-88. https://doi.org/10.1016/S0012-365X(98)00333-1

Examples

```
el1 <- c("a", "b", "c", "d")
el2 <- c("x", "y")
el3 <- c("h", "k")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = el1, dom = dom)

pos2 <- LinearPOSet(elements = el2)

pos3 <- LinearPOSet(elements = el3)

pos <- ProductPOSet(pos1, pos2, pos3)

# median_distr computes the frequency distribution of median profile

elements <- POSetElements(pos)

median_distr <- function(le) {
  n <- length(elements)
  if (n %% 2 != 0) {
    res <- (elements == le[(n + 1) / 2])
  } else {
    res <- (elements == le[n / 2])
  }
  res <- as.matrix(res)
```

```
    rownames(res) <- elements
    colnames(res) <- "median_distr"
    return (as.matrix(res))
  }

  # computation with parameter n
  BDgen <- BuildBubleyDyerEvaluationGenerator(poset = pos, seed = NULL, median_distr)
  res <- BubleyDyerEvaluation(BDgen, n=40000, output_every_sec=1)
  #refinement results with parameter n
  res <- BubleyDyerEvaluation(BDgen, n=10000, output_every_sec=1)
  #refinement results with parameter error
  res <- BubleyDyerEvaluation(BDgen, error=0.2, output_every_sec=1)
  #Attempt to further refine results with parameter `error=0.2` does not modify previous result
  res <- BubleyDyerEvaluation(BDgen, error=0.2, output_every_sec=1)

  # computation with parameter error
  BDgen <- BuildBubleyDyerEvaluationGenerator(poset = pos, seed = NULL, median_distr)
  res <- BubleyDyerEvaluation(BDgen, error=0.2, output_every_sec=1)
```

BubleyDyerEvaluationGenerator-class

An S4 class to represent function evaluation based on the Bubley-Dyer procedure.

Description

An S4 class to represent function evaluation based on the Bubley-Dyer procedure.

Slots

`ptr` an external pointer to a C++ data

BubleyDyerGenerator-class

An S4 class to represent the linear extension generator based on the Bubley-Dyer procedure.

Description

An S4 class to represent the linear extension generator based on the Bubley-Dyer procedure.

Slots

`ptr` an external pointer to C++ data

BubleyDyerMRP *Approximating MRP matrix computation, using the Bubley-Dyer procedure.*

Description

Computes an approximated MRP matrix, starting from a set of linear extensions sampled according to the Bubley-Dyer procedure.

Usage

```
BubleyDyerMRP(generator, n = NULL, error = NULL, output_every_sec = NULL)
```

Arguments

`generator`	The approximated MRP matrix generator created by function `BubleyDyerMRPGenerator()`.
`n`	Number of linear extensions generated to compute the approximated MRP matrix. See `Details` for further information on this argument.
`error`	A real number in $(0, 1)$, representing the "distance" from uniformity of the sampling distribution of the linear extensions. This parameter is used to determine the number of linear extensions to be sampled, in order to achieve the desired "distance". According to Bubley and Dyer (1999), if `error`$= \epsilon$ and E is the number of elements in the poset, then the number n_ϵ of sampled linear extensions is given by $n_\epsilon = E^4(\ln(E))^2 + E^3 \ln(E) \ln(\epsilon^{-1})$. If both arguments `n` and `error` are specified by the user, the number of linear extensions actually generated is `n`.
`output_every_sec`	Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `BubleyDyerMRP`, a message reporting the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec` seconds.

Details

The function `BubleyDyerMRP` allows the user to update a previously computed approximated MRP matrix, so as to improve estimation accuracy. Specifically, the argument `generator` internally stores the approximated MRP matrix computed at each execution of `BubleyDyerMRP`. At the subsequent call (with the same `generator` argument), the previously computed MRP are updated, based on the newly sampled linear extensions. In this case, the number of additional linear extensions is determined either directly, by parameter `n`, or indirectly, by specifying parameter `error`, which sets the desired "distance" from uniformity of the sampling distribution of linear extensions, in the Bubley-Dyer procedure. In the latter case, the number of additional linear extensions is computed as $n_\epsilon - n_a$, where n_ϵ is the number of linear extensions necessary to achive the desired "distance" and n_a is the total number of linear extensions generated in the previous calls of `BubleyDyerEvaluation`. If $n_\epsilon - n_a \leq 0$, no further linear extensions are generated and a warning message is displayed.

In case a newly computed approximated MRP matrix is desired, run `BubleyDyerMRP` with a `generator` argument newly generated by function `BubleyDyerMRPGenerator()`.

Value

A list of two elements: 1) the matrix of approximated MRP and 2) the number of linear extensions generated to compute it.

References

Bubley, R., Dyer, M. (1999). Faster random generation of linear extensions. Discrete Mathematics, 201, 81-88. https://doi.org/10.1016/S0012-365X(98)00333-1

Examples

```
el1 <- c("a", "b", "c")
el2 <- c("x", "y", "z")
el3 <- c("h", "k")
dom <- matrix(c(
  "a", "b",
  "c", "b"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = el1, dom = dom)

pos2 <- LinearPOSet(elements = el2)

pos3 <- LinearPOSet(elements = el3)

pos <- ProductPOSet(pos1, pos2, pos3)

BDgen <- BubleyDyerMRPGenerator(pos)
#MRP computation with parameter n
res <- BubleyDyerMRP(BDgen, n=700000, output_every_sec=1)
#MRP refinement with parameter n
res <- BubleyDyerMRP(BDgen, n=100000, output_every_sec=1)
#MRP refinement with parameter error
res <- BubleyDyerMRP(BDgen, error=0.05, output_every_sec=1)
#Attempt to further refine MRP with parameter `error=0.05` does not modify previous result
res <- BubleyDyerMRP(BDgen, error=0.05, output_every_sec=1)

#new MRP computation with parameter error
BDgen <- BubleyDyerMRPGenerator(pos)
res <- BubleyDyerMRP(BDgen, error=0.05, output_every_sec=1)
```

BubleyDyerMRPGenerator

Generator of an approximated MRP matrix.

Description

Creates an object of S4 class `BubleyDyerMRPGenerator` for the computation of an approximated MRP matrix, starting from a set of random linear extensions, sampled according to the Bubley-Dyer procedure. Actually, this function does not compute the MRP matrix, but just the object that will compute it, by using function `BubleyDyerMRP`.

Usage

```
BubleyDyerMRPGenerator(poset = NULL, seed = NULL)
```

Arguments

`poset`	Object of S4 class `POSet` representing the poset whose MRP are computed. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`seed`	Positive integer to initialize the random linear extension generation.

Value

An object of S4 class `BubleyDyerMRPGenerator`.

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)

BDgen <- BubleyDyerMRPGenerator(pos)
```

BubleyDyerMRPGenerator-class

An S4 class to represent the MRP generator based on the Bubley-Dyer procedure.

Description

An S4 class to represent the MRP generator based on the Bubley-Dyer procedure.

Slots

`ptr` an external pointer to C++ data

BubleyDyerSeparation

Approximated separation matrices computation, using the Bubley-Dyer procedure (see Bubley and Dyer, 1999).

Description

Computes approximated separation matrices, starting from a set of linear extensions sampled according to the Bubley-Dyer procedure.

Usage

```
BubleyDyerSeparation(
  generator,
  n = NULL,
  error = NULL,
  output_every_sec = NULL
)
```

Arguments

`generator`	The approximated separation matrices generator created by function `BuildBubleyDyerSeparationGener`
`n`	number of linear extensions generated to compute the approximated MRP matrix. See documentation for function `BubleyDyerMRP()` for further information on this argument.
`error`	A real number in $(0, 1)$ representing the "distance" from uniformity of the sampling distribution of the linear extensions. This parameter is used to determine the number of linear extensions to be sampled. If both arguments `n` and `error` are specified by the user, the number of linear extensions actually generated is `n`. See documentation for function `BubleyDyerMRP` for further information on this argument.

`output_every_sec`
: Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `BubleyDyerSeparation`, a message reporting the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec` seconds.

Details

See the documentation of `BuildBubleyDyerSeparationGenerator()` for details on how the different types of separations are defined and computed.

Value

A list containing: 1) the required type of approximated separation matrices, according to the parameter `type` used to build the `generator` (see`BuildBubleyDyerSeparationGenerator()`); 2) the number of generated linear extensions.

References

Bubley, R., Dyer, M. (1999). Faster random generation of linear extensions. Discrete Mathematics, 201, 81-88. https://doi.org/10.1016/S0012-365X(98)00333-1

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)

BDgen <- BuildBubleyDyerSeparationGenerator(pos, seed = NULL,
                  type="symmetric", "asymmetricUpper", "vertical")

SEP_matrices <- BubleyDyerSeparation(BDgen, n=10000, output_every_sec=5)
```

`BubleyDyerSeparationGenerator-class`
An S4 class to represent function separation based on the Bubley-Dyer procedure.

Description

An S4 class to represent function separation based on the Bubley-Dyer procedure.

Slots

`ptr` an external pointer to a C++ data

`types` list of separation to be computed

BuildBubleyDyerEvaluationGenerator

Generator for the approximated computation of the mean value of functions over linear extensions.

Description

`BuildBubleyDyerEvaluationGenerator` creates an object of S4 class `BuildBubleyDyerEvaluationGenerator`, for the estimation of the mean values of the input functions, over linear extensions sampled according to the Bubley-Dyer procedure. Actually, this function does not perform the computation of mean values, but just generates the object that will compute them by using function `BubleyDyerEvaluation`.

Usage

```
BuildBubleyDyerEvaluationGenerator(poset, seed, f1, ...)
```

Arguments

`poset`	An object of S4 class `POSet` representing the poset from which linear extensions are generated. Object `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`seed`	Positive integer to initialize random linear extension generation. Set `seed=NULL` for random initialization.
`f1`	The function whose mean value is to be computed. `f1` must be an R-function having as a single parameter a linear extension of `poset` and returning a numerical matrix.
`...`	Further functions whose mean values are to be computed.

Value

An object of S4-class `BuildBubleyDyerEvaluationGenerator`.

Examples

```
el1 <- c("a", "b", "c", "d")
el2 <- c("x", "y")
el3 <- c("h", "k")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = el1, dom = dom)

pos2 <- LinearPOSet(elements = el2)

pos3 <- LinearPOSet(elements = el3)

pos <- ProductPOSet(pos1, pos2, pos3)

# median_distr computes the frequency distribution of median profile

elements <- POSetElements(pos)

median_distr <- function(le) {
  n <- length(elements)
  if (n %% 2 != 0) {
    res <- (elements == le[(n + 1) / 2])
  } else {
    res <- (elements == le[n / 2])
  }
  res <- as.matrix(res)
  rownames(res) <- elements
  colnames(res) <- "median_distr"
  return (as.matrix(res))
}

BDgen <- BuildBubleyDyerEvaluationGenerator(poset = pos, seed = NULL, median_distr)
```

BuildBubleyDyerSeparationGenerator

Generator of an approximated separation matrix.

Description

Creates an object of S4 class BubleyDyerSeparationGenerator for the computation of approximated separation matrices, starting from a set of random linear extensions, sampled according to the Bubley-Dyer procedure (see Bubley and Dyer, 1999) Actually, this function does not compute the separation matrices, but just the object that will compute them, by using function BubleyDyerSeparation.

Usage

```
BuildBubleyDyerSeparationGenerator(poset, seed, type, ...)
```

Arguments

poset	Object of S4 class POSet representing the poset whose separation matrices are to be computed. Argument poset must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...) .

`seed`	Positive integer to initialize the random linear extension generation.
`type`	type of separation to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".
`...`	additional types of separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Details

The symmetric separation associated to elements a and b in the input poset is the average absolute difference between the positions of a and b observed in the sampled linear extensions (whose elements are arranged in ascending order):

$Sep_{ab} = \frac{1}{n}\sum_{i^1}^{n} |Pos_{l_i}(a) - Pos_{l_i}(b)|$,

where n is the numbers of sampled linear extensions; l_i represents a sampled linear extension and $Pos_{l_i}(\cdot)$ stands for the position of element $\cdot$ into the sequence of poset elements arranged in increasing order according to l_i.

Asymmetric lower and upper separations are defined as: $Sep_{a<b} = \frac{1}{n}\sum_{i^1}^{n}(Pos_{l_i}(b) - Pos_{l_i}(a)) \nVdash (a <_{l_i} b)$, $Sep_{b<a} = \frac{1}{n}\sum_{i^1}^{n}(Pos_{l_i}(a) - Pos_{l_i}(b)) \nVdash (b <_{l_i} a)$, where $a \leq_{l_i} b$ means that a is lower or equal to b in the linear order defined by linear extension l_i and $\nVdash$ is the indicator function. Note that $Sep_{ab} = Sep_{a<b} + Sep_{a<b}$.

Vertical and horizontal separations ($vSep$ and $hSep$, respectively) are defined as

$vSep_{ab} = |Sep_{a<b} - Sep_{b<a}|$ and #' $hSep_{ab} = Sep_{ab} - vSep_{ab}|$.

For a detailed explanation on why $vSep$ and $hSep$ can be interpreted as vertical and horizontal components of the separation between two poset elements, see Fattore et. al (2024).

Value

An object of S4 class `BubleyDyerSeparationGenerator`.

References

Bubley, R., Dyer, M. (1999). Faster random generation of linear extensions. Discrete Mathematics, 201, 81-88. https://doi.org/10.1016/S0012-365X(98)00333-1

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)
```

```
BDgen <- BuildBubleyDyerSeparationGenerator(pos, seed = NULL,
              type="symmetric", "asymmetricUpper", "vertical")
```

`ComparabilitySetOf` *Extracting the comparability set of a poset element.*

Description

Extracts the elements comparable with the input element, in the poset.

Usage

```
ComparabilitySetOf(poset, element)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element`	A character string (the name of the input poset element).

Value

A vector of character strings (the names of the poset elements comparable to the input element).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

cmp <- ComparabilitySetOf(pos, "a")
```

CoverMatrix *Computing the cover matrix of a poset.*

Description

Computes the cover matrix of the input poset.

Usage

```
CoverMatrix(poset)
```

Arguments

`poset`	An object of S4 class POSet. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A square boolean matrix C ($C[i, j] = TRUE$ if and only if the j-th element of the input poset covers the i-th).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

C <- CoverMatrix(pos)
```

CoverRelation *Computing the cover relation of a poset.*

Description

Computes the cover relation of the input poset.

Usage

```
CoverRelation(poset)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A two-column matrix M of character strings (element $M[i, 2]$ covers element $M[i, 1]$).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

M <- CoverRelation(pos)
```

`CrownPOSet` *Building crowns.*

Description

Builds a crown from two unordered collections of elements, with the same size.

Usage

```
CrownPOSet(elements_1, elements_2)
```

Arguments

`elements_1`	A list of character strings.
`elements_2`	A list of character strings.

Details

Let $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ be two disjoint collections of n elements. The "crown" over them is the poset $P = (V, \lhd)$ having $a_1, \ldots, a_n, b_1, \ldots, b_n$ as ground set and where $(a_i||a_j)$, $(b_i||b_j)$, $(a_i||b_i)$ and $a_i \lhd b_j$, for each $i \neq j$ ($||$ stands for "incomparable to").

Value

A crown, an object of S4 class `POSet`.

Examples

```
elems1<-c("a1", "a2", "a3", "a4", "a5")
elems2<-c("b1", "b2", "b3", "b4", "b5")
crown<-CrownPOSet(elems1, elems2)
```

DisjointSumPOSet *Disjoint sum of posets.*

Description

Computes the disjoint sum of the input posets.

Usage

```
DisjointSumPOSet(poset1, poset2, ...)
```

Arguments

poset1	An object of S4 class POSet. Argument poset1 must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...).
poset2	An object of S4 class POSet. Argument poset2 must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...).
...	Optional additional objects of S4 class POSet. Optional arguments must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...).

Details

Let $P_1 = (V_1, \leq_1), \dots, P_k = (V_k, \leq_k)$ be k posets on disjoint ground sets. Their disjoint sum is the poset $P = (V, \lhd)$ having as ground set the union of the input ground sets, with $a \leq b$ if and only if $a, b \in V_i$ and $a \leq_i b$ for some i.

Value

The disjoint sum poset, an object of S4 class POSet.

Examples

```
elems1 <- c("a", "b", "c", "d")
elems2 <- c("e", "f", "g", "h")

dom1 <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

dom2 <- matrix(c(
  "e", "f",
  "g", "h",
  "h", "f"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems1, dom = dom1)

pos2 <- POSet(elements = elems2, dom = dom2)

dsj.sum <- DisjointSumPOSet(pos1, pos2)
```

`DominanceMatrix` *Computing the dominance matrix.*

Description

Computes the dominance matrix of the input poset.

Usage

```
DominanceMatrix(poset)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

An $n \times n$ boolean matrix Z, where n is the number of poset elements, with $Z[i, j] = TRUE$, if and only if the j-th poset element weakly dominates ($\leq$) the i-th element, in the input order relation.

Examples

```
elems <- c("a", "b", "c", "d")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

Z <- DominanceMatrix(pos)
```

`Dominates` *Checking whether one element dominates another.*

Description

Given two elements a and b of V, checks whether $a \leq b$ in poset $(V, \leq)$.

Usage

```
Dominates(poset, element1, element2)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element1`	A character string (the name of a poset element).
`element2`	A character string (the name of a poset element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- Dominates(pos, "a", "d")
```

`DownsetOf` *Computing downsets.*

Description

Computes the downset of a set of elements of the input poset.

Usage

```
DownsetOf(poset, elements)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`elements`	A vector of character strings (the names of the input elements).

Value

A vector of character strings (the names of the elements of the downset).

Examples

```
elems<- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "a", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

dwn <- DownsetOf(pos, c("b","d"))
```

`DualPOSet` *Dual of a poset.*

Description

Computes the dual of the input poset.

Usage

```
DualPOSet(poset)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).

Details

Let $P = (V, \leq)$ be a poset. Then its dual $P_d = (V, \leq_d)$ is defined by $a \leq_d b$ if and only if $b \leq a$ in P. In other words, the dual of P is obtained by reversing its dominances.

Value

The dual of the input poset, an object of S4 class `POSet`.

Examples

```
elems <- c("a", "b", "c", "d")

doms <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems, dom = doms)

dual <- DualPOSet(pos1)
```

`ExactEvaluation` *Computing function mean values on linear extensions*

Description

`ExactEvaluation` computes the mean values of the input functions (defined on linear orders) over the set of linear extensions of the input poset. The linear extensions are generated according to the algorithm given in Habib M, Medina R, Nourine L and Steiner G (2001).

Usage

```
ExactEvaluation(poset, output_every_sec = NULL, f1, ...)
```

Arguments

`poset` An object of S4 class `POSet` representing the poset from which linear extensions are generated. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

`output_every_sec`
integer specifying a time interval (in seconds). By specifying this argument, during the execution of `BubleyDyerEvaluation`, the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec`seconds.

`f1`	the function whose average value has to be computed. `f1` can be an R-function having as a single argument a linear extension of `poset` and returning a numerical matrix.
`...`	Further functions whose averages are to be computed.

Value

A list of the computed averages, along with the number of linear extensions generated to compute them.

References

Habib M, Medina R, Nourine L and Steiner G (2001). Efficient algorithms on distributive lattices. Discrete Applied Mathematics, 110, 169-187. https://doi.org/10.1016/S0166-218X(00)00258-4.

Examples

```
el1 <- c("a", "b", "c", "d")
el2 <- c("x", "y")
el3 <- c("h", "k")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = el1, dom = dom)

pos2 <- LinearPOSet(elements = el2)

pos3 <- LinearPOSet(elements = el3)

pos <- ProductPOSet(pos1, pos2, pos3)

# median_distr computes the frequency distribution of median profile

elements <- POSetElements(pos)

median_distr <- function(le) {
  n <- length(elements)
  if (n %% 2 != 0) {
    res <- (elements == le[(n + 1) / 2])
  } else {
    res <- (elements == le[n / 2])
  }
  res <- as.matrix(res)
  rownames(res) <- elements
  colnames(res) <- "median_distr"
  return (as.matrix(res))
}

res  <- ExactEvaluation(pos, output_every_sec=1, median_distr)
```

ExactMRP *Computing Mutual Ranking Probabilities (MRP).*

Description

Computes the MRP matrix of a poset. The MRP associated to $a \leq b$, with a and b two elements of the input poset, is the share of linear extensions of it where b dominates a. The linear extensions are computed according to the algorithm given in Habib M, Medina R, Nourine L and Steiner G (2001).

Usage

```
ExactMRP(poset, output_every_sec = NULL)
```

Arguments

`poset`	An object of S4 class `POSet` representing the poset whose MRP are computed. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`output_every_sec`	Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `ExactMRP` a message reporting the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec` seconds.

Value

A list of two elements: 1) the MRP matrix and 2) the number of linear extensions generated to compute it.

References

Habib M, Medina R, Nourine L and Steiner G (2001). Efficient algorithms on distributive lattices. Discrete Applied Mathematics, 110, 169-187. https://doi.org/10.1016/S0166-218X(00)00258-4.

Examples

```
el1 <- c("a", "b", "c", "d")
el2 <- c("x", "y")
el3 <- c("h", "k")
dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = el1, dom = dom)

pos2 <- LinearPOSet(elements = el2)

pos3 <- LinearPOSet(elements = el3)

pos <- ProductPOSet(pos1, pos2, pos3)

MRP <- ExactMRP(pos, output_every_sec=1)
```

`ExactMRPGenerator-class` *An S4 class to represent the exact MRP generator.*

Description

An S4 class to represent the exact MRP generator.

Slots

`ptr` an external pointer to C++ data

`ExactSeparation` *Exact separation matrices computation.*

Description

Computes exact separation matrices by evaluating the average separation over all the linear extensions of the input poset. The linear extensions are generated according to the algorithm given in Habib M, Medina R, Nourine L and Steiner G (2001).

Usage

```
ExactSeparation(poset, output_every_sec = NULL, type, ...)
```

Arguments

`poset`	Object of S4 class `POSet` representing the poset whose separation matrix is computed. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

`output_every_sec`
: Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `BubleyDyerSeparation`, a message reporting the number of linear extensions progressively generated is printed on the R-Console, every `output_every_sec` seconds.

`type`
: type of separation to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

`...`
: additional types of Separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Details

The symmetric separation associated to two elements a and b of the input poset, is the average absolute difference between the positions of a and b observed over all linear extensions (whose elements are arranged in ascending order):

$Sep_{ab} = \frac{1}{n}\sum_{i^1}^{n} |Pos_{l_i}(a) - Pos_{l_i}(b)|$,

where n is the numbers of linear extensions of the input poset; l_i represents a single linear extension and $Pos_{l_i}(\cdot)$ stands for the position of element $\cdot$ into the sequence of poset elements arranged in increasing order according to l_i.

Asymmetric lower and upper separations are defined as: $Sep_{a<b} = \frac{1}{n}\sum_{i^1}^{n}(Pos_{l_i}(b) - Pos_{l_i}(a))\mathbb{1}(a <_{l_i} b)$, $Sep_{b<a} = \frac{1}{n}\sum_{i^1}^{n}(Pos_{l_i}(a) - Pos_{l_i}(b))\mathbb{1}(b <_{l_i} a)$, where $a \leq_{l_i} b$ means that a is lower or equal to b in the linear order defined by linear extension l_i and $\mathbb{1}$ is the indicator function. Note that $Sep_{ab} = Sep_{a<b} + Sep_{a<b}$.

Vertical and horizontal separations ($vSep$ and $hSep$, respectively) are defined as

$vSep_{ab} = |Sep_{a<b} - Sep_{b<a}|$ and #' $hSep_{ab} = Sep_{ab} - vSep_{ab}|$.

For a detailed explanation on why $vSep$ and $hSep$ can be interpreted as vertical and horizontal components of the separation between poset elements, see Fattore et. al (2024).

Value

A list containing: 1) the required type of approximated separation matrices, according to the parameter `type` used to build the `generator` (see function `BuildBubleyDyerSeparationGenerator`); 2) the number of generated linear extensions.

References

Habib M, Medina R, Nourine L and Steiner G (2001). Efficient algorithms on distributive lattices. Discrete Applied Mathematics, 110, 169-187. https://doi.org/10.1016/S0166-218X(00)00258-4.

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
    "a", "b",
    "c", "b",
    "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)

SEP_matrices <- ExactSeparation(pos, output_every_sec=5, "symmetric","asymmetricUpper", "vertical")
```

`FencePOSet` *Building fences.*

Description

Builds a fence, from an unordered collection of elements.

Usage

```
FencePOSet(elements, orientation = "upFirst")
```

Arguments

`elements`	A list of character strings (the names of the fence elements).
`orientation`	Either "upFirst" (the first element dominates the second) or "downFirst" (the first element is dominated by the second).

Value

A fence, an object of S4 class `POSet`.

Examples

```
elems <- c("a", "b", "c", "d", "e")
fence <- FencePOSet(elems, orientation="upFirst")
```

`FromPOSet-class` *An S4 class to represent a virtual class for POSet extention.*

Description

An S4 class to represent a virtual class for POSet extention.

Slots

`ptr` an external pointer to C++ data

FuzzyInBetweenness *Fuzzy in-betweenness array computation*

Description

Starting from a poset dominance matrix, computes in-betweenness arrays by using a user supplied t-norm and t-conorm.

Usage

```
FuzzyInBetweenness(dom, norm, conorm, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degree between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of the poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`norm`	R-function defining the t-norm
`conorm`	R-function defining the t-conorm
`type`	type of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper". For details on the definition of symmetric and asymmetric in-betweenness see Fattore et al. (2024).
`...`	additional types of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper".

Value

a list of three-dimensional arrays, one array for each type of in-betweenness selected by parameter `type`. The array element of position $[i, j, k]$ represents $finb_{p_i,p_j,p_k}$ for symmetric in-betweenness, $finb_{p_i<p_j<p_k}$ for asymmetricLower in-betweenness, and $finb_{p_k<p_j<p_i}$ for asymmetricUpper in-betweenness.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)

BLS <- BLSDominance(pos)

tnorm <- function(x,y){x*y}

tconorm <- function(x,y){x+y-x*y}

FinB <- FuzzyInBetweenness(BLS, norm=tnorm, conorm=tconorm, type="symmetric", "asymmetricLower")
```

FuzzyInBetweennessMinMax

Fuzzy in-betweenness array computation with minimum t-norm and maximum t-conorm

---

Description

Starting from a poset dominance matrix, computes in-betweenness arrays by using minimum t-norm and maximum t-conorm.

Usage

```
FuzzyInBetweennessMinMax(dom, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degree between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`type`	type of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper". For details on the definition of symmetric and asymmetric in-betweenness see Fattore et al. (2024).
`...`	additional types of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper".

Value

a list of three-dimensional arrays, one array for each type of in-betweenness selected by parameter `type`. The array element of position $[i, j, k]$ represents $finb_{p_i,p_j,p_k}$ for symmetric in-betweenness, $finb_{p_i<p_j<p_k}$ for asymmetricLower in-betweenness, and $finb_{p_k<p_j<p_i}$ for asymmetricUpper in-betweenness.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)

BLS <- BLSDominance(pos)

FinB <- FuzzyInBetweennessMinMax(BLS, type="symmetric", "asymmetricLower")
```

FuzzyInBetweennessProbabilistic

Fuzzy in-betweenness array computation with Product t-norm and Probabilistic-sum t-conorm

---

Description

Starting from a poset dominance matrix, computes in-betweenness arrays by using Product t-norm and Probabilistic-sum t-conorm

Usage

```
FuzzyInBetweennessProbabilistic(dom, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degree between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of the poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`type`	type of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper". For details on the definition of symmetric and asymmetric in-betweenness see Fattore et al. (2024).
`...`	additional types of in-betweenness to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper".

Value

a list of three-dimensional arrays, one array for each type of in-betweenness selected by parameter `type`. The array element of position $[i, j, k]$ represents $finb_{p_i,p_j,p_k}$ for symmetric in-betweenness, $finb_{p_i<p_j<p_k}$ for asymmetricLower in-betweenness, and $finb_{p_k<p_j<p_i}$ for asymmetricUpper in-betweenness.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)

BLS <- BLSDominance(pos)

FinB <- FuzzyInBetweennessProbabilistic(BLS, type="symmetric", "asymmetricLower")
```

`FuzzySeparation` *Fuzzy separation matrix computation*

Description

Starting from a poset dominance matrix, computes fuzzy separation matrices by using the t-norm and t-conorm supplied by the user.

Usage

```
FuzzySeparation(dom, norm, conorm, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degree between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of the poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`norm`	R-function defining the t-norm

`conorm`	R-function defining the t-conorm
`type`	type of fuzzy Separation to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal". For details on the definition of symmetric, asymmetric, vertical and horizontal separations see Fattore et al. (2024).
`...`	additional types of fuzzy separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Value

list of required fuzzy separation matrices.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)

BLS <- BLSDominance(pos)

tnorm <- function(x,y){x*y}

tconorm <- function(x,y){x+y-x*y}

FSep <- FuzzySeparation(BLS, norm=tnorm, conorm=tconorm,
            type="symmetric", "asymmetricLower", "vertical")
```

`FuzzySeparationMinMax` *Fuzzy Separation computation with minimum t-norm and maximum t-conorm*

Description

Starting from a poset dominance matrix, computes fuzzy Separation matrices by using minimum t-norm and maximum t-conorm.

Usage

```
FuzzySeparationMinMax(dom, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degree between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of the poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`type`	type of fuzzy separation to be computed. Possible Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal". For details on the definition of symmetric, asymmetric, vertical and horizontal separations see Fattore et al. (2024).
`...`	additional types of fuzzy separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Value

list of required fuzzy separation matrices.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)

BLS <- BLSDominance(pos)

FSep <- FuzzySeparationMinMax(BLS, type="symmetric", "asymmetricLower", "vertical")
```

FuzzySeparationProbabilistic

Fuzzy Separation matrix computation with Product t-norm and Probabilistic-sum t-conorm

Description

Starting from a poset dominance matrix, computes fuzzy Separation matrices by using Product t-norm and Probabilistic-sum t-conorm

Usage

```
FuzzySeparationProbabilistic(dom, type, ...)
```

Arguments

`dom`	square matrix representing the dominance degrees between pairs of poset elements. Columns and rows names of `dom` are interpreted as the labels of the poset elements. `dom` can be computed by using functions such as `BLSDominance`, `BubleyByesMRP` and `ExactMRP`.
`type`	type of fuzzy separation to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal". For details on the definition of symmetric, asymmetric, vertical and horizontal separations see Fattore et al. (2024).
`...`	additional types of fuzzy separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Value

list of required fuzzy separation matrices.

References

Fattore, M., De Capitani, L., Avellone, A., and Suardi, A. (2024). A fuzzy posetic toolbox for multi-criteria evaluation on ordinal data systems. Annals of Operations Research, https://doi.org/10.1007/s10479-024-06352-3.

Examples

```
el <- c("a", "b", "c", "d")

dom_list <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom_list)
```

```
BLS <- BLSDominance(pos)

FSep <- FuzzySeparationProbabilistic(BLS, type="symmetric", "asymmetricLower", "vertical")
```

IncomparabilityRelation

Computing the incomparability relation of a poset.

Description

Computes the incomparability relation of the input poset.

Usage

```
IncomparabilityRelation(poset)
```

Arguments

`poset`	An object of S4 class POSet. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A two-column matrix M (element $M[i, 2]$ is incomparable with element $M[i, 1]$).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

M <- IncomparabilityRelation(pos)
```

`IncomparabilitySetOf` *Extracting the incomparability set of a poset element.*

Description

Extracts the elements incomparable with the input element, in the poset.

Usage

```
IncomparabilitySetOf(poset, element)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element`	A character string (the names of a single poset element).

Value

A vector of character strings (the names of the poset elements incomparable with the input element).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)
incmp <- IncomparabilitySetOf(pos, "a")
```

`IntersectionPOSet` *Computing the intersection of a collection of posets.*

Description

Computes the poset $(V, \leq_\cap) = (V, \leq_1) \cap \cdots \cap (V, \leq_k)$.

Usage

```
IntersectionPOSet(poset1, poset2, ...)
```

Arguments

`poset1`	An object of S4 class `POSet`. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`poset2`	An object of S4 class `POSet`. Argument `poset2` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`...`	Optional additional objects of S4 class `POSet`. Optional arguments must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Details

Let $P_1 = (V, \leq_1), \cdots, P_k = (V, \leq_k)$ be k posets on the same set V. The intersection poset $P_\cap = P_1 \cap \cdots \cap P_k$ is the poset $(V, \leq_\cap)$ where $a \leq_\cap b$ if and only if $a \leq_i b$ for all $i = 1 \cdots k$.

Value

The intersection poset, an object of S4 class `POSet`.

Examples

```
elems <- c("a", "b", "c", "d")

dom1 <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

dom2 <- matrix(c(
  "a", "b",
  "b", "c",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems, dom = dom1)

pos2 <- POSet(elements = elems, dom = dom2)

pos_int <- IntersectionPOSet(pos1, pos2)
```

`IsAntisymmetric` *Checking binary relation antisymmetry.*

Description

Checks whether the input binary relation is antisymmetric.

Usage

```
IsAntisymmetric(rel)
```

Arguments

`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation,

Value

A boolean value.

Examples

```
rel <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d",
  "a", "a"
), ncol = 2, byrow = TRUE)

chk <- IsAntisymmetric(rel)
```

`IsComparableWith` *Checking comparability between two elements of a poset.*

Description

Checks whether two elements a and b of V are comparable in the input poset $(V, \leq)$.

Usage

```
IsComparableWith(poset, element1, element2)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element1`	A character string (the name of a poset element).
`element2`	A character string (the name of a poset element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsComparableWith(pos, "a", "d")
```

`IsDominatedBy` *Checking whether one element is dominated by another.*

Description

Given two elements a and b of V, checks whether $a \leq b$ in poset $(V, \leq)$.

Usage

```
IsDominatedBy(poset, element1, element2)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element1`	A character string (the name of a poset element).
`element2`	A character string (the name of a poset element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsDominatedBy(pos, "a", "d")
```

`IsDownset` *Checking for downsets.*

Description

Checks whether the input elements form a downset, in the input poset.

Usage

```
IsDownset(poset, elements)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`elements`	A vector of character strings (the names of the input) elements).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsDownset(pos, c("a", "b", "c"))
```

IsExtensionOf — *Checking poset extensions.*

Description

Checks whether poset1 is an extension of poset2.

Usage

```
IsExtensionOf(poset1, poset2)
```

Arguments

`poset1`	An object of S4 class 'POSet'. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`poset2`	An object of S4 class 'POSet'. Argument `poset2` must be created by using any function contained in the package aimed at building object of S4cl ass `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom1 <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

dom2 <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d",
  "a", "c"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems, dom = dom1)
pos2 <- POSet(elements = elems, dom = dom2)

chk <- IsExtensionOf(pos1, pos2)
```

IsIncomparableWith *Checking incomparability between two elements of a poset.*

Description

Checks whether two elements a and b of V are incomparable, in the input poset $(V, \leq)$.

Usage

```
IsIncomparableWith(poset, element1, element2)
```

Arguments

poset	An object of S4 class POSet. Argument poset must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...) .
element1	A character string (the name of a poset element).
element2	A character string (the name of a poset element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsIncomparableWith(pos, "a", "d")
```

IsMaximal *Checking maximality.*

Description

Checks whether the input element is maximal in the input poset.

Usage

```
IsMaximal(poset, element)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element`	A character string (the name of the input element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk<-IsMaximal(pos, "b")
```

`IsMinimal` *Checking minimality.*

Description

Checks whether the input element is minimal in the input poset.

Usage

```
IsMinimal(poset, element)
```

Arguments

`poset`	An object of S4-class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`element`	A character string (the name of the input element).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsMinimal(pos, "a")
```

`IsPartialOrder` *Checking for partial ordering.*

Description

Checks whether the input binary relation is a partial order.

Usage

```
IsPartialOrder(set, rel)
```

Arguments

`set`	A list of character strings (the names of the elements of the set, on which the binary relation is defined).
`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A boolean value.

Examples

```
set<-c("a", "b", "c", "d")

rel <- matrix(c(
  "a", "b",
  "c", "b",
  "d", "a",
  "c", "a",
  "d", "b",
  "a", "a",
  "b", "b",
  "c", "c",
  "d", "d"
), ncol = 2, byrow = TRUE)

chk <- IsPartialOrder(set, rel)
```

`IsPreorder` *Checking for pre-ordering (or quasi-ordering).*

Description

Checks whether the input relation is a pre-order (aka, a quasi-order), i.e. if it is reflexive and transitive.

Usage

```
IsPreorder(set, rel)
```

Arguments

`set`	A list of character strings (the names of the elements of the set, on which the binary relation is defined).
`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A boolean value.

Examples

```
set<-c("a", "b", "c", "d")

rel <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "a",
  "d", "a",
  "c", "a",
  "d", "b",
  "a", "a",
  "b", "b",
  "c", "c",
  "d", "d"
), ncol = 2, byrow = TRUE)

chk<-IsPreorder(set, rel)
```

IsReflexive *Checking binary relation reflexivity.*

Description

Checks whether the input binary relation is reflexive.

Usage

```
IsReflexive(set, rel)
```

Arguments

set	A list of strings (the names of the elements of the set, on which the binary relation is defined).
rel	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A boolean value.

Examples

```
set<-c("a", "b", "c", "d", "e")

rel <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d",
  "a", "a",
  "b", "b",
  "c", "c"
), ncol = 2, byrow = TRUE)

chk <- IsReflexive(set, rel)
```

`IsSymmetric` *Checking binary relation symmetry.*

Description

Checks whether the input binary relation is symmetric.

Usage

```
IsSymmetric(rel)
```

Arguments

`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A boolean value.

Examples

```
rel <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d",
  "b", "a",
  "b", "c",
  "d", "b"
), ncol = 2, byrow = TRUE)

chk<-isSymmetric(rel)
```

`IsTransitive` *Checking binary relation transitivity.*

Description

Checks whether the input relation is transitive.

Usage

```
IsTransitive(rel)
```

Arguments

`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A boolean value.

Examples

```
rel <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d",
  "a", "d",
  "c", "d"
), ncol = 2, byrow = TRUE)

chk<-IsTransitive(rel)
```

`IsUpset` *Checking upsets.*

Description

Checks whether the input elements form an upset, in the input poset.

Usage

```
IsUpset(poset, elements)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`elements`	A vector of character strings (the names of the input elements).

Value

A boolean value.

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

chk <- IsUpset(pos, c("a", "b", "c"))
```

`LEBubleyDyer` *Generator of linear extensions through the Bubley-Dyer procedure.*

Description

Creates an object of S4 class `LEBubleyDyer`, needed to sample the linear extensions of a given poset according to the Bubley-Dyer procedure. Actually, this function does not sample the linear extensions, but just generates the object that will sample them by using function `LEGet`.

Usage

```
LEBubleyDyer(poset, seed = NULL)
```

Arguments

`poset`	An object of S4 class `POSet` representing the poset whose linear extensions are generated. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`seed`	Positive integer to initialize the random linear extension generation.

Value

An object of class `LEBubleyDyer`.

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)

LEgenBD <- LEBubleyDyer(pos)
```

LEGenerator *Generator of all the linear extensions of a poset.*

Description

Creates an object of S4 class `LEGenerator` to generate all of the linear extensions of a given poset. Actually, this function does not generate the linear extensions, but just the object that will generate them by using function `LEGet`.

Usage

```
LEGenerator(poset)
```

Arguments

`poset`	An object of S4 class `POSet` representing the poset whose linear extensions are generated. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

An S4 class object `LEGenerator`.

Examples

```
el <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = el, dom = dom)

LEgen <- LEGenerator(pos)
```

`LEGenerator-class` *An S4 class to represent the exact linear extension generator.*

Description

An S4 class to represent the exact linear extension generator.

Slots

`ptr` an external pointer to C++ data

`LEGet` *Generates linear extensions of a given poset, by using a linear extension generator*

Description

Generates the linear extensions of a poset, by using a linear extension generator built by functions `LEGenerator` and `LEBubleyDyer`.

Usage

```
LEGet(
  generator,
  from_start = TRUE,
  n = NULL,
  error = NULL,
  output_every_sec = NULL
)
```

Arguments

`generator`	The linear extension generator built by function `LEGenerator()` or `LEBubleyDyer()`. If `LEGenerator` is used, `n` linear extensions of the poset are generated, according to a deterministic generation order, consistent with the algorithm given in Habib M, Medina R, Nourine L and Steiner G (2001). If `LEBubleyDyer` is used, the proper number of linear extensions is randomly sampled by using the Bubley and Dyer (1999) procedure, based on parameter `n` and `error`.
`from_start`	Logical value indicating whether the linear extensions generator should be reset or not. If `from_start=FALSE`, linear extensions generation starts from the last linear extension generated in the previous `LEGet` call. If `from_start=TRUE`, previous `LEGet` calls do not impact on the new linear extensions generation. In more details, when `generator` is created via `LEGenerator()`, linear extensions are built after a deterministic rule that fixes the generation order. In this case, if `from_start=TRUE` the generation starts from the first linear extension

in the generation order; if `from_start=FALSE`, the first linear extension generated is the successor, in the prefixed generation order, to the last one generated by the previous `LEGet` call. This allows the user to generate the set of all linear extensions of a poset in subsequent `LEGet` calls, so as to keep control on computational times and memory space.

When `generator` is created via `LEBubleyDyer()`, linear extensions are generated according to the Bubley-Dyer sampling procedure that, starting from a randomly generated linear extension, produces a sequence of linear orders in which the k-th one is obtained by a proper random modification of the $(k-1)$-th one. In this case, if `from_start=TRUE`, the first linear extension is chosen at random; if `from_start=FALSE`, it is obtained by randomly modifying the last one, generated in the previous `LEGet` call.

`n` number of linear extensions to be generated.

`error` A real number in the interval $(0, 1)$ representing the "distance" from uniformity in the sampling distribution of linear extensions. This parameter is used only when `generator` is of class `BubleyDyerGenerator`. It determines the number of linear extensions to be genrated, in order to achieve the desired "distance" from uniformity, in the sampling distribution of linear extensions. According to Bubley and Dyer (1999), if `error=`ϵ and E is the number of elements in the poset, then the number n_ϵ of linear extensions to be sampled is given by

$n_\epsilon = E^4(\ln(E))^2 + E^3 \ln(E) \ln(\epsilon^{-1})$.

If both arguments `n` and `error` are specified by the user, the number of linear extensions actually generated is `n`.

`output_every_sec`

integer specifying a time interval (in seconds). By specifying this argument, during the execution of `LEGet`, the number of linear extensions actually generated is printed on the R-Console, every `output_every_secseconds`.

Value

A matrix whose columns reports the generated linear extensions

References

Bubley, R., Dyer, M. (1999). Faster random generation of linear extensions. Discrete Mathematics, 201, 81-88. https://doi.org/10.1016/S0012-365X(98)00333-1

Habib M, Medina R, Nourine L and Steiner G (2001). Efficient algorithms on distributive lattices. Discrete Applied Mathematics, 110, 169-187. https://doi.org/10.1016/S0166-218X(00)00258-4.

Examples

```
el <- c("a", "b", "c", "d", "e")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "c", "d"
), ncol = 2, byrow = TRUE)
```

```
pos <- POSet(elements = el, dom = dom)

LEgen <- LEGenerator(pos)
LEmatrix <- LEGet(LEgen)

LEgen <- LEGenerator(pos)
LEmatrix_first <- LEGet(LEgen, n=10)
LEmatrix_second <- LEGet(LEgen, from_start=FALSE)
LEmatrix <- cbind(LEmatrix_first,LEmatrix_second)

#Randomly generate n=30 linear extensions from the poset
LEgen <- LEBubleyDyer(pos)
LEmatrix <- LEGet(LEgen, n=30)

#Randomly generate n=60 linear extensions from the poset in two steps
LEgen <- LEBubleyDyer(pos)
LEmatrix_first <- LEGet(LEgen, n=30)
LEmatrix_second <- LEGet(LEgen, n=30, from_start=FALSE)
LEmatrix <- cbind(LEmatrix_first,LEmatrix_second)

#Generates linear extensions from the poset with precision=1
LEgen <- LEBubleyDyer(pos)
LEmatrix <- LEGet(LEgen, error=0.002)
```

LexicographicProductPOSet

Computing lexicographic product orders.

Description

Computes the lexicographic product order of the input partial orders.

Usage

```
LexicographicProductPOSet(poset1, poset2, ...)
```

Arguments

`poset1`	An object of S4 class `POSet`. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`poset2`	An object of S4 class `POSet`. Argument `poset2` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`...`	Optional additional objects of S4 class `POSet`. Optional arguments must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).

Details

Let $P_1 = (V_1, \leq_1), \cdots, P_k = (V_k, \leq_k)$ be k posets. The lexicographic product poset $P_{lxprd} = (V, \leq_{lxprd})$ has ground set the Cartesian product of the input ground sets, with $(a_1, \ldots, a_k) \leq_{lxprd} (b_1, \ldots, b_k)$ if and only $a_1 \leq_1 b_1$, or there exists j such that $a_i = b_i$ for $i < j$ and $a_j \leq_j b_j$.

Value

The lexicographic product poset, an object of S4 class `POSet`.

Examples

```
elems1 <- c("a", "b", "c", "d", "e")
elems2 <- c("f", "g", "h")

dom1 <- matrix(c(
  "a", "b",
  "c", "b",
  "d", "b"
), ncol = 2, byrow = TRUE)

dom2 <- matrix(c(
  "g", "f",
  "h", "f"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems1, dom = dom1)

pos2 <- POSet(elements = elems2, dom = dom2)

#Lexicographic product of pos1 and pos2
lex.prod <- LexicographicProductPOSet(pos1, pos2)
```

`LexicographicProductPOSet-class`
An S4 class to represent a Lexicographic Product POSet.

Description

An S4 class to represent a Lexicographic Product POSet.

Slots

`ptr` an external pointer to C++ data

LexMRP — *MRP matrix computation over the set of lexicographic linear extensions.*

Description

Considering the component-wise poset built stating from k ordinal variables, computes MRP matrix by analyzing all poset lexicographic linear extensions.

Usage

```
LexMRP(nvar, deg)
```

Arguments

`nvar`	positive integer specifying the number k of ordinal variables.
`deg`	parameter specifying the number of degrees of each variable. If all k variables have the same number m of degrees, it can be:

1. the positive integer m. In this case variable degree labels are supposed to be the integers $0 < 1 < ... < m$ and columns and rows of the computed MRP matrix are named accordingly to this;
2. a character vector of length m specifying the variable degree labels (in this case columns and rows of the computed MRP matrix are named accordingly to `deg`).

If the k variables have different number $(m_1, ..., m_k)$ of degrees, it can be:

1. a length-k positive integers vector specifying the values of $m_1, ..., m_k$. In this case variable degree labels for the j-th variable are supposed to be the integers $0 < 1 < ... < m_j$ and columns and rows of the computed MRP matrix are named accordingly to this;
2. a list of k character vectors. The j-th list element is a character vector of length m_j specifying the degree labels for the j-th variable (in this case columns and rows of the computed MRP matrix are named accordingly to `deg`).

Value

the MRP matrix computed over the set of lexicographic linear extensions.

Examples

```
#variables with common number of degrees
# default labels for variable degrees
nvar <- 3
deg  <- 4
lMRP <- LexMRP(nvar=nvar, deg=deg)
```

```
#user supplied variable degree labels
nvar <- 3
deg  <- c("a","b","c","d")
lMRP <- LexMRP(nvar=nvar, deg=deg)


#variables with different numbers of degrees
# default labels for variable degrees
nvar <- 3
deg  <- c(4,2,3)
lMRP <- LexMRP(nvar=nvar, deg=deg)

#user supplied variable degree labels
nvar <- 3
deg  <- list(c("a","b","c","d"),c("0","1"),c("x","y","z"))
lMRP <- LexMRP(nvar=nvar, deg=deg)
```

`LexSeparation` *Separation matrices computation over the set of lexicographic linear extensions.*

Description

Considering the component-wise poset built stating from k ordinal variables, computes separation matrices by analyzing all poset lexicographic linear extensions.

Usage

```
LexSeparation(nvar, deg, type, ...)
```

Arguments

`nvar`	positive integer specifying the number k of ordinal variables.
`deg`	parameter specifying the number of degrees of each variable. If all k variables have the same number m of degrees, it can be: 1. the positive integer m. In this case variable degree labels are supposed to be the integers $0 < 1 < ... < m$ and columns and rows of computed separation matrices are named accordingly to this; 2. a character vector of length m specifying the variable degree labels (in this case columns and rows of computed separation matrices are named accordingly to deg). If the k variables have different number $(m_1, ..., m_k)$ of degrees, it can be: 1. a length-k positive integers vector specifying the values of $m_1, ..., m_k$. In this case variable degree labels for the j-th variable are supposed to be the integers $0 < 1 < ... < m_j$ and columns and rows of computed separation matrices are named accordingly to this;

	2. a list of k character vectors. The j-th list element is a character vector of length m_j specifying the degree labels for the j-th variable (in this case columns and rows of computed separation matrices are named accordingly to deg).
`type`	type of separation matrix to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".
`...`	additional types of separations to be computed. Possible choices are: "symmetric", "asymmetricLower", "asymmetricUpper", "vertical", "horizontal".

Value

list of required separation matrices.

Examples

```
#variables with common number of degrees
# default labels for variable degrees
nvar <- 3
deg  <- 4
LexSep <- LexSeparation(nvar=nvar, deg=deg, type= "symmetric", "asymmetricLower")

#user supplied variable degree labels
nvar <- 3
deg  <- c("a","b","c","d")
LexSep <- LexSeparation(nvar=nvar, deg=deg, type= "symmetric", "asymmetricLower")


#variables with different numbers of degrees
# default labels for variable degrees
nvar <- 3
deg  <- c(4,2,3)
LexSep <- LexSeparation(nvar=nvar, deg=deg, type= "symmetric", "asymmetricLower")

#user supplied variable degree labels
nvar <- 3
deg  <- list(c("a","b","c","d"),c("0","1"),c("x","y","z"))
LexSep <- LexSeparation(nvar=nvar, deg=deg, type= "symmetric", "asymmetricLower")
```

`LiftingPOSet` *Lifting posets.*

Description

Lifts the input poset, i.e. adds a (possibly new) bottom element to it.

Usage

```
LiftingPOSet(poset, element)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`element`	A character string (the name of the added bottom).

Value

The lifted poset, an object of S4 class `POSet`.

Examples

```
elems <- c("a", "b", "c", "d")

doms <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = doms)

#Lifting
lifted.pos <- LiftingPOSet(pos, "bot")
```

`LinearPOSet` *Constructing a Linearly Ordered Set.*

Description

Constructs a linearly (or completely, or totally) ordered set (V,$\leq_{lin}$), starting from set V,

Usage

```
LinearPOSet(elements)
```

Arguments

`elements`	A character string vector containing the labels of the elements of V in ascending order according to $\leq_{lin}$, i.e. such that `elements[h]`$\leq_{lin}$`elements[k]`if and only if h $\leq$ k

Value

An object of S4 class `LinearPOSet` (subclass of `POSet`)

Examples

```
elems <- c("a", "b", "c", "d")
linpos <- LinearPOSet(elements = elems)
```

`LinearPOSet-class` *An S4 class to represent a Linear POSet.*

Description

An S4 class to represent a Linear POSet.

Slots

`ptr` an external pointer to C++ data

`LinearSumPOSet` *Linear sum of posets.*

Description

Computes the linear sum of the input posets.

Usage

```
LinearSumPOSet(poset1, poset2, ...)
```

Arguments

`poset1`	An object of S4 class `POSet`. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`poset2`	An object of S4 class `POSet`. Argument `poset2` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`...`	Optional additional objects of S4 class `POSet`. Optional arguments must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).

Details

Let $P_1 = (V_1, \leq_1), \dots, P_k = (V_k, \leq_k)$ be k posets on disjoint ground sets. Their linear sum is the poset $P = (V, \lhd)$ having as ground set the union of the input ground sets, with $a \leq b$ if and only if $a \leq_i b$ for some i, or $a \in V_i$ and $b \in V_j$, with $i < j$. In other words, the linear sum is obtained by stacking the input posets from bottom, and making all of the minimal elements of P_i covering all of the maximal elements of P_{i-1} ($i > 1$).

Value

The linear sum poset, an object of S4 class POSet.

Examples

```
elems1 <- c("a", "b", "c", "d")
elems2 <- c("e", "f", "g", "h")

dom1 <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

dom2 <- matrix(c(
  "e", "f",
  "g", "h",
  "h", "f"
), ncol = 2, byrow = TRUE)

pos1 <- POSet(elements = elems1, dom = dom1)

pos2 <- POSet(elements = elems2, dom = dom2)

#Linear sum of pos1 and pos2
lin.sum <- LinearSumPOSet(pos1, pos2)
```

OptimalBidimensionalEmbedding

Dimensionality reduction of multidimensional ordinal binary data

---

Description

Starting from a dataset with n statistical units, scored against k ordinal 0/1-indicators and partially ordered component-wise into a Boolean lattice $B_k = (\{0,1\}^k, \leq_{cmp})$, it finds the bidimensional data representation that optimally preserves the input order relation. The algorithm finding the best bidimensional representation is optimized by using a parallel C++ implementation.

Usage

```
OptimalBidimensionalEmbedding(
  profile,
  weights,
  output_every_sec = NULL,
  thread_share = 1
)
```

Arguments

`profile`	Boolean matrix of dimension $m \times k$ of the unique $m \leq n$ different observed profiles. Each observed profile is a row of `profile`. Each observed profile is repeated only once in the matrix `profile`.
`weights`	real vector of length m with the frequencies/weights of each observed profiles. Element of position j in vector `weights` is the frequency/weight of the profile in row j of `profile`.
`output_every_sec`	Integer specifying a time interval (in seconds). By specifying this argument, during the execution of `OptimalBidimensionalEmbedding`, a message reporting the number of reversed pairs of lexicographic linear extensions analyzed is printed on the R-Console, every `output_every_sec` seconds. Note that the number of reversed pairs of lexicographic linear extensions to be analyzed is $k!/2$.
`thread_share`	real number in the interval $(0, 1])$ specifying the share of CPU threads to be involved in the algorithm execution.

Value

a list of 5 elements named `allLoss`, `variablesPriority`, `bestLossVAlue`, `bestVariablePriority`, and `bestRepresentation`.

`allLoss` is a vector of dimension $k!/2$ reporting the value of the loss function $L(D^{out}|D^{inp}, p)$ corresponding to the representation induced by each reversed pairs of lexicographic linear extensions. This loss function measures the global errors made in approximating the order structure of the input Boolean Lattice B_k with its bidimensional representations.

`variablesPriority` is a matrix with $k!/2$ rows and k columns. Each row is an integer vector of dimension k containing a permutation $i_1, ..., i_k$ of $1, ..., k$. This vector specifies the criterion to build the reversed pair of lexicographic linear extensions used to approximate B_k. The first linear extension is built by ordering profiles first according to their scores on V_{i_1}, then to the scores on V_{i_2} and so on, until V_{i_k}; the second linear extension is built by ordering profiles first according to their scores on V_{i_k}, then to the scores on $V_{i_{k-1}}$ and so on, until V_{i_1}. The j-th row of `variablesPriority` identifies the reversed pair of lexicographic linear extensions inducing the bidimensional representation associated to the j-th global loss in `allLoss`.

`bestLossVAlue` real number indicating the minimum value of the global error $L(D^{out}|D^{inp}, p)$ among the $k!/2$ global errors associated to the different pairs of reversed lexicographic linear extensions.

`bestVariablePriority` integer vector of dimension k containing the permutation of $1, ..., k$ inducing the best bidimensional representation, i.e. the bidimensional representation with associated global error `bestLossVAlue`.

`bestRepresentation` a data frame with m values (one value for each observed profile) of 5 variables named `profiles`, `x`, `y`, `weights` and `error`. `$profile` is an integer vector containing the base-10 representation of the k-dimensional Boolean vectors representing observed profiles. `$x` is an integer vector containing the x-coordinates of points representing observed profiles in the optimal bidimensional representation. `$y` is an integer vector containing the y-coordinates of points representing observed profiles in the optimal bidimensional representation. `$weights` is a real vector with the frequencies/weights of each observed profile. `$error` is a real vector with the values of the approximation errors $L(b|D^{inp}, p)$ associated to each observed profile in the optimal bidimensional representation.

Examples

```
#SIMULATING OBSERVED BINARY DATA
#number of binary variables
k <- 6
#building observed profiles matrix
profiles <- sapply((0:(2^k-1)) ,function(x){ as.integer(intToBits(x))})
profiles <- t(profiles[1:k, ])
#building the vector of observation frequencies
weights <- sample.int(100, nrow(profiles), replace=TRUE)
#FINDING THE OPTIMAL BIDIMENSIONAL REPRESENTATION
result <- OptimalBidimensionalEmbedding(profiles, weights)
```

OrderRelation *Extracting the order relation of a poset.*

Description

Extracts the order relation from the input poset.

Usage

```
OrderRelation(poset)
```

Arguments

poset	An object of S4 class POSet. Argument poset must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...) .

Value

A two-column matrix M of character strings (element $M[i, 2]$ dominates element $M[i, 1]$).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)
```

```
M <- OrderRelation(pos)
```

POSet *Constructing a Partially Ordered Set.*

Description

Constructs an object of class `POSet`, representing a partially ordered set (poset) $P = (V, \leq)$.

Usage

```
POSet(elements, dom = matrix(ncol = 2, nrow = 0))
```

Arguments

`elements`	A vector of character strings (the labels of the elements of the ground set V).
`dom`	Two-columns matrix of element labels, representing the dominances in the order relation $\leq$. The generic `k-th` row of `dom` contains a pair of elements of V, with `dom[k, 1]` $\leq$ `dom[k, 2]`.

Value

An object $(V, \leq)$ of S4 class `POSet`, where V is the ground set and $\leq$ is the partial order relation on it

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)
```

`POSet-class` *An S4 class to represent a POSet.*

Description

An S4 class to represent a POSet.

Slots

`ptr` an external pointer to C++ data

`POSetElements` *Getting poset elements.*

Description

Gets the elements of the ground set V of the input poset $(V, \leq)$.

Usage

```
POSetElements(poset)
```

Arguments

`poset`	An object of S4 class `POSet` Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A vector of labels (the names of the elements of the ground set V).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

gset <- POSetElements(pos)
```

`POSetJoin` *Computing join (least upper bound).*

Description

The function computes the join (if existing) of a set of elements, in the input poset.

Usage

```
POSetJoin(poset, elements)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`elements`	A list of character strings (the names of some poset elements).

Value

A character string (the name of the join).

Examples

```
elems <- c("a", "b", "c", "d")

doms <- matrix(c(
  "a", "b",
  "c", "b",
  "a", "d",
  "a", "a",
  "b", "b",
  "c", "c",
  "d", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = doms)

lub<-POSetJoin(pos, c("a", "c"))
```

POSetMaximals *Computing the maximal elements of a poset.*

Description

Computes the maximal elements of the input poset, i.e. those elements being strictly dominated by no other elements.

Usage

```
POSetMaximals(poset)
```

Arguments

poset	An object of S4 class POSet. Argument poset must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...) .

Value

A vector of character strings (the names of the maximal elements).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b"
  ,
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

mxs <- POSetMaximals(pos)
```

POSetMeet *Computing meet (greatest lower bound).*

Description

The function computes the meet (if existing) of a set of elements, in the input poset.

Usage

```
POSetMeet(poset, elements)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...).
`elements`	A list of character strings (the names of some poset elements).

Value

A character string (the name of the meet).

Examples

```
elems <- c("a", "b", "c", "d", "e")

doms <- matrix(c(
  "a", "b",
  "c", "b",
  "c", "e",
  "b", "d",
  "a", "d",
  "c", "d",
  "a", "a",
  "b", "b",
  "c", "c",
  "d", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = doms)

glb<-POSetMeet(pos, c("b", "e"))
```

`POSetMinimals` *Computing the minimal elements of a poset.*

Description

Computes the minimal elements of the input poset, i.e. those elements strictly dominating no other elements.

Usage

```
POSetMinimals(poset)
```

Arguments

`poset`	An object of S4 class `POSet`. Argument `poset` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Value

A vector of character strings (the names of the minimal elements).

Examples

```
elems <- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)

mnms <-POSetMinimals(pos)
```

`ProductPOSet` *Constructing the product of posets.*

Description

Constructs the product poset $(V, \leq_{prd})$, starting from a collection of posets.

Usage

```
ProductPOSet(poset1, poset2, ...)
```

Arguments

`poset1`	An object of S4 class `POSet`. Argument `poset1` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`poset2`	An object of S4 class `POSet`. Argument `poset2` must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .
`...`	Optional additional objects of S4 class `POSet`. Optional arguments must be created by using any function contained in the package aimed at building object of S4 class `POSet` (e.g. `POSet()`, `LinearPOSet()`, `ProductPOSet()`, ...) .

Details

Let $P_1 = (V_1, \leq_1), ..., P_k = (V_k, \leq_k)$ be a collection of posets. The product poset $P = P_1 \times ... \times P_k$ is the poset $(V, \leq_{prd})$ where $V = V_1 \times ... \times V_k$ and given $(a_1, ..., a_k) \in V$ and $(b_1, ..., b_k) \in V$, $(a_1, ..., a_k) \leq_{prd} (b_1, ..., b_k)$ if and only if $a_i \leq_i b_i$ for all $i = 1, ..., k$.

Value

The product poset, an object of S4 class `ProductPOSet` (subclass of `POSet`).

Examples

```
elems1 <- c("a", "b", "c", "d")
elems2 <- c("x", "y", "z")
elems3 <- c("q", "r")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

p1 <- POSet(elements = elems1, dom = dom)
p2 <- LinearPOSet(elements = elems2)
p3 <- LinearPOSet(elements = elems3)

prd12 <- ProductPOSet(p1, p2)

prd123 <- ProductPOSet(p1, p2, p3)
```

`ProductPOSet-class` *An S4 class to represent a Product POSet.*

Description

An S4 class to represent a Product POSet.

Slots

`ptr` an external pointer to C++ data

`ReflexiveClosure` *Computing reflexive closure.*

Description

Computes the reflexive closure of the input binary relation.

Usage

```
ReflexiveClosure(set, rel)
```

Arguments

`set`	A list of character strings (the names of the elements of the set, on which the binary relation is defined).
`rel`	A two-columns character matrix, each row comprising an element (pair) of the relation.

Value

A reflexive binary relation, as a two-columns character matrix (each row comprises an element (pair) of the transitivity closed relation).

Examples

```
set<-c("a", "b", "c", "d", "e")

rel <- matrix(c(
  "a", "b",
  "c", "b",
  "d", "a",
  "c", "a",
  "a", "a",
  "d", "d"
), ncol = 2, byrow = TRUE)

r.clo<-ReflexiveClosure(set, rel)
```

`TransitiveClosure` *Computing transitive closure.*

Description

Computes the transitive closure of the input binary relation.

Usage

```
TransitiveClosure(rel)
```

Arguments

`rel`	A two-columns character matrix, each row comprising an element (pair) of the binary relation.

Value

A transitive binary relation, as a two-columns character matrix (each row comprises an element (pair) of the transitivity closed relation).

Examples

```
rel <- matrix(c(
  "a", "b",
  "c", "b",
  "d", "a",
  "c", "a",
  "a", "a",
  "b", "b",
  "c", "c",
  "d", "d"
), ncol = 2, byrow = TRUE)

t.clo<-TransitiveClosure(rel)
```

UpsetOf *Computing upsets.*

Description

Computes the upset of a set of elements of the input poset.

Usage

```
UpsetOf(poset, elements)
```

Arguments

poset	An object of S4 class POSet. Argument poset must be created by using any function contained in the package aimed at building object of S4 class POSet (e.g. POSet(), LinearPOSet(), ProductPOSet(), ...) .
elements	a vector of character strings (the names of the input elements).

Value

A vector of character strings (the names of the poset elements in the upset).

Examples

```
elems<- c("a", "b", "c", "d")

dom <- matrix(c(
  "a", "b",
  "c", "b",
  "b", "d"
), ncol = 2, byrow = TRUE)

pos <- POSet(elements = elems, dom = dom)
```

```
up <- UpsetOf(pos, c("a","c"))
```

Index