Package 'comato'

March 2, 2018

Type Package

Title Analysis of Concept Maps and Concept Landscapes

Version 1.1

Date 2018-03-02

Author Andreas Muehling

Maintainer Andreas Muehling <andreas.muehling@tum.de>

Description Provides methods for the import/export and automated analysis of concept maps and concept landscapes (sets of concept maps).

License GPL-3

Imports igraph, Matrix, lattice, gdata, XML, cluster, clusterSim, graphics, stats

Encoding UTF-8

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-03-02 16:36:47 UTC

R topics documented:

`analyze.graph.measures`

Analyze graph measures of a concept map

Description

`analyze.graph.measures` analyzes several graph measures. For actual implementations see `analyze.graph.measures.co` or `analyze.graph.measures.igraph`.

Usage

```
analyze.graph.measures(x)
```

Arguments

x	A conceptmap.

Value

A list of several graph measures.

`analyze.graph.measures.conceptmap`
Analyzing graph measures of a concept map

Description

`analyze.graph.measures` analyzes several basic graph measures of a given graph in form of a conceptmap object. All measures are derived by the appropriate functions of igraph.

Usage

```
## S3 method for class 'conceptmap'
analyze.graph.measures(x)
```

Arguments

x	A conceptmap object.

Value

A list with named components that contain the betweenness measure, the edge.connectivity, the diameter, the degree distribution and the communities using the Fastgreedy algorithm.

Examples

```
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(15, 0.7, type="gnp"), "name", value=1:15)
analyze.graph.measures(conceptmap(g1))
```

analyze.graph.measures.igraph *Analyzing graph measures of an igraph object*

Description

analyze.graph.measures.igraph is a convenience method that can be called directly on the result of landscape. It works just like analyze.graph.measures.conceptmap

Usage

```
## S3 method for class 'igraph'
analyze.graph.measures(x)
```

Arguments

x	An igraph object.

Value

A list with named components that contain the betweenness measure, the edge.connectivity, the diameter, the degree distribution and the communities using the Fastgreedy algorithm.

Examples

```
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(15, 0.7, type="gnp"), "name", value=1:15)
analyze.graph.measures(g1)
```

analyze.similarity *Analyzing graph similarity.*

Description

analyze.similarity calculates a measure of graph similarity between two concept maps.

Usage

```
analyze.similarity(map1, map2)
```

Arguments

map1	A conceptmap object.
map2	A conceptmap object.

Value

A value between 0 and 1 that indicated the structural similariy of the underlying graphs. The graphs need not share the same set of nodes.

See Also

The structural similarity that is calculated is described in: Goldsmith, Timothy E.; Davenport, Daniel M. (1990): Assessing Structural Similarity of Graphs. In: Roger W. Schvaneveldt (Hg.): Pathfinder associative networks. Studies in knowledge organizations. Norwood, N.J: Ablex Pub. Corp., S. 74-87.

Examples

```
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(15, 0.7, type="gnp"), "name", value=1:15)
g2 = set.vertex.attribute(erdos.renyi.game(15, 0.7, type="gnp"), "name", value=1:15)
analyze.similarity(conceptmap(g1), conceptmap(g2))
```

---

analyze.subgraphs *Analyzing small subgraph patterns.*

Description

`analyze.subgraphs` analyzes the frequency of subgraph patterns given a list of concepts and a set of maps in a conceptmaps object.

Usage

```
analyze.subgraphs(maps, concept.list)
```

Arguments

`maps`	A conceptmaps object.
`concept.list`	A list of concepts (as strings) that define the subgraphs to be analyzed. must be between 2 and 4 concepts in length.

Value

A list with two elements. The first is a vector that contains the absolute number of occurrences for each subgraph pattern. The second element is a list of igraph objects of the pattern themselves.

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))

analyze.subgraphs(simple_cms, c("1", "2", "3"))
```

---

as.matrix.conceptmap *Convert a conceptmap object to a matrix*

Description

`as.matrix` converts a conceptmap object into a matrix. The output can be fed back into `conceptmap.matrix`.

Usage

```
## S3 method for class 'conceptmap'
as.matrix(x, ...)
```

Arguments

`x`	A conceptmap object.
`...`	-

Value

A matrix with 3 columns and one row for each proposition of the concept map. The 3 columns contain the start and end node of each proposition as well as the label of the edge (as character vectors).

---

as.matrix.conceptmaps *Convert a conceptmaps object to a matrix*

Description

`as.matrix` converts a conceptmaps object into a matrix. The output can be fed back into `conceptmaps.matrix`.

Usage

```
## S3 method for class 'conceptmaps'
as.matrix(x, ...)
```

Arguments

x	A conceptmaps object.
...	-

Value

A matrix with 4 columns and one row for each proposition of one of the concept maps. The 4 columns contain an id of the map (starting from 1) and then the start and end node of each proposition as well as the label of the edge (as character vectors).

Examples

```
data = rbind(
 cbind("1", "Object", "Class", "is instance of"),
 cbind("1", "Object", "Attribute", "has"),
 cbind("2", "Class", "Attribute", "possesses"),
 cbind("2", "Attribute", "Data-type", "has"),
 cbind("3", "Object", "Class", "is instance of")
 )
 cms = conceptmaps(data)
 as.matrix(cms)
```

---

clustering *Clustering maps of a conceptmaps object*

Description

`clustering` is a convenience function that implements two frequently used ways of clustering conceptmaps directly. The first is clustering using the MBMM algorithm and the concept matrix, the second is clustering using the PAM algorithm and the graph similarity matrix.

Usage

```
clustering(maps, method = c("MBMM", "PAM"), min = 1, max = 10)
```

Arguments

maps	A conceptmaps object.
method	Either "PAM" or "MBMM", indicating which algorithm should be used.
min	The minimal number of components that is tested. For the PAM algorithm, 1 is not allowed.
max	The maximal number of components that is tested.

Value

The return value of either `MBM.cluster` or `PAM.cluster`, depending on the value of `method`.

Examples

```
## Not run:
#Assuming that there are concept maps in folder "~/maps"
cms = read.folder.tgf("~/maps")

clustering(cms, method="MBMM")

## End(Not run)
```

`concept.vector` *Forming the concept vector of a conceptmap object*

Description

`concept.vector` transforms a concept map into a numeric vector that contains for each occuring concept the number of adjacent edges.

Usage

```
concept.vector(x)
```

Arguments

x	A conceptmap object.

Value

A numeric vector. The columns are named after the concepts and sorted alphabetically.

Examples

```
#Create concept map from a random graph
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
cm = conceptmap(g1)

concept.vector(cm)
```

conceptmap *Constructing a conceptmap object*

Description

conceptmap creates an object that encompasses a concept maps. For actual implementations, see conceptmap.default, conceptmap.igraph, or conceptmap.matrix.

Usage

```
conceptmap(x, ...)
```

Arguments

x	-
...	-

Value

A conceptmap object.

conceptmap.default *Basic creation of a conceptmap object*

Description

conceptmap creates a conceptmap object based on either an empty concept map or on (matrix) data.

Usage

```
## Default S3 method:
conceptmap(x, ...)
```

Arguments

x	The concept map data. For NULL, an empty concept map is created, otherwise the object is cooerced into a matrix.
...	-

Value

A conceptmap object.

Examples

```
empty_cm = conceptmap(NULL)
```

conceptmap.igraph *Creation of a conceptmap object from an existing graph*

Description

conceptmap takes an existing igraph object and tries of coerce it into a conceptmap object (encompassing the igraph object).

Usage

```
## S3 method for class 'igraph'
conceptmap(x, strip = TRUE, ...)
```

Arguments

x	An igraph object. It must have an attribute called "name" for both vertices and edges. Additional attributes are preserved for graph, vertices and edges.
strip	If TRUE, nodes without adjacent edges are removed from the graph / concept map.
...	-

Value

A conceptmap object.

Examples

```
#Create conceptmap from a complete graph with 5 nodes
require("igraph")
graph = graph.full(5)
graph = set.vertex.attribute(graph, "name", value=1:5)
simple_cm = conceptmap(graph)
```

conceptmap.matrix *Creation of a conceptmap object from matrix data*

Description

conceptmap creates a conceptmap object from a given matrix of a particular format (see below).

Usage

```
## S3 method for class 'matrix'
conceptmap(x, ...)
```

Arguments

x	A matrix of character vectors with at least 3 columns. Each row is of the form: start, end, label, (edge attribute 1), ..., (edge attribute m). Each such row will be interpreted as a directed edge from concept "start" to concept "end" with the name "label" and (optional) m additional edge attributes. The colum names of `map.data`, if present, will be preserved as the names for the attributes.
...	-

Value

A conceptmap object.

Examples

```
data = rbind(cbind("Object", "Class", "is instance of"), cbind("Class", "Attribute", "has"))
cm = conceptmap(data)
```

---

conceptmaps *Constructing a conceptmaps object*

Description

`conceptmaps` creates an object that encompasses a set of concept maps. For actual implementations, see `conceptmaps.default`, `conceptmaps.list`, `conceptmaps.matrix`.

Usage

```
conceptmaps(x, ...)
```

Arguments

x	-
...	-

Value

A conceptmaps object.

conceptmaps.default *Basic creation of a conceptmaps object*

Description

conceptmaps creates a conceptmaps object based on either an empty set of concept map or on a list of concept maps.

Usage

```
## Default S3 method:
conceptmaps(x, ...)
```

Arguments

x	The set of concept maps. For NULL, an empty set of concept maps is created, otherwise the object is cooerced into a list.
...	-

Value

A conceptmaps object.

Examples

```
empty_cms = conceptmaps(NULL)
```

conceptmaps.list *Creation of a conceptmaps object from a list*

Description

conceptmaps creates a conceptmaps object from a list of conceptmap objects.

Usage

```
## S3 method for class 'list'
conceptmaps(x, filter = T, ...)
```

Arguments

x	A list of conceptmap objects.
filter	If TRUE, empty concept maps (i.e. concept maps without any proposition) are not contained in the resulting set.
...	-

Value

A conceptmaps object.

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))
```

---

conceptmaps.matrix *Creation of a conceptmaps object from a matrix*

Description

conceptmaps creates a conceptmaps object from a set of concept maps represented as a matrix.

Usage

```
## S3 method for class 'matrix'
conceptmaps(x, filter = T, ...)
```

Arguments

x	A matrix that represents a set of concept maps. The first colum is taken to identify the map, i.e. for each value occuring in the first column, the rows with identical values are extracted and conceptmap.matrix is called on the matrix of these rows and the remaining columns to create a conceptmap object.
filter	If TRUE, empty concept maps (i.e. concept maps without any proposition) are not contained in the resulting set.
...	-

Value

A conceptmaps object.

Examples

```
data = rbind(
 cbind("1", "Object", "Class", "is instance of"),
 cbind("1", "Object", "Attribute", "has"),
 cbind("2", "Class", "Attribute", "possesses"),
 cbind("2", "Attribute", "Data-type", "has"),
 cbind("3", "Object", "Class", "is instance of")
```

```
)
cms = conceptmaps(data)
```

---

edge.vector *Forming the edge vector of a conceptmap object*

Description

edge.vector transforms a concept map into a numeric vector that contains for each occuring pair of concepts whether or not this pair is connected by a proposition in the concept map.

Usage

```
edge.vector(x)
```

Arguments

x	A conceptmap object.

Value

A numeric vector. The columns are named after the concept-pairs which are sorted alphabetically.

Examples

```
#Create concept map from a random graph
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
cm = conceptmap(g1)

edge.vector(cm)
```

---

get.unified.concepts *Finding all concepts used.*

Description

get.unified.concepts identifies the common superset of concepts that is used by the maps of a conceptmaps object.

Usage

```
get.unified.concepts(maps)
```

Arguments

maps	A conceptmaps object.

Value

A vector of strings of each concepts that appears in at least one of the maps of `maps`.

`Hopkins.index` *Non-randomness of data*

Description

`Hopkins.index` calculates the Hopkins index that can be used as an indicator of the non-randomness of data prior to clustering.

Usage

```
Hopkins.index(data)
```

Arguments

`data`	A numeric matrix.

Value

The Hopkins index as a numeric value

See Also

The index is described in, e.g.: Han, Jiawei; Kamber, Micheline (2010): Data mining. Concepts and techniques. 2nd ed., Amsterdam: Elsevier/Morgan Kaufmann (The Morgan Kaufmann series in data management systems).

Examples

```
## Not run:
#Random data generation, 10 dimensions, 500 observations, 2 clusters,
#Multivariate-Bernoulli distributed
require("gtools")
data = c()
p = 0.0
for (i in 1:2)
{
temp = c()
for (j in 1:10)
temp = cbind(temp, rbinom(250, 1, p+(i-1)*0.5+(0.025*i)*j))
data=rbind(data, temp)
}
data = data[permute(1:500),]

Hopkins.index(data)

## End(Not run)
```

`landscape` *Aggregating the maps of a conceptmaps object into a concept landscape*

Description

`landscape` transforms a set of concept maps into a concept landscape using one of several possible methods. Depending on the value of `result` and accumulation or amalgamation is performed on the concept map data. The amalgamation forms a weighted graph based on the unified set of concepts. An accumulation transforms each concept map into a vector and returns a matrix of these vectors. Using `FUN` the process of transformation can be influenced in both cases.

Usage

```
landscape(maps, result = c("graph", "matrix"), mode, FUN = NULL)
```

Arguments

`maps`	A conceptmaps object.
`result`	Either "graph" or "matrix" defines the return type and the method of aggregation. An amalgamation results in a weighted graph and an accumulation results a matrix.
`mode`	If `result` is "graph", it can be either "directed" or "undirected" deciding how the graph should be formed. First, a weight matrix is formed from the set of concept maps. If `FUN` is NULL, the weights simply reflect the number of maps in which a given edge is present. Otherwise, `FUN` must be a function that accepts three parameters and returns a numeric value. For the first parameter the current conceptmap object will be passed and for the second and third parameters the start and end concepts of an edge is passed. The return value of the function will then be added to the weight matrix. If `result` is "matrix", it can be one of "graph.sim", "concept.vector", "edge.vector" or "custom". "graph.sim" return the graph similarity matrix, "concept.vector" and "edge.vector" return the concept matrix or edge matrix respectively. Finally, "custom" can be used for arbitrary transformations: For each map, the function passed to `FUN` and the resulting vector is forming a row of the returned matrix.
`FUN`	If `result` is "matrix" and `mode` is "custom", a function with one parameter of type conecptmap must be given that is called for each of the consituent maps. The function must return a numeric vector of equal length for all maps of a conceptmaps object.

Value

Depending on `result` either an igraph object or a numeric matrix.

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))

landscape(cms, result="graph", mode="undirected")

landscape(cms, result="matrix", mode="concept.vector")
```

`MBM.cluster` *MBMM clustering*

Description

`MBM.cluster` calculates a model based clustering using multivariate Bernoulli-mixtures as probabilistic model of the data. The quality of the clustering is judged using the AIC criterion.

Usage

```
MBM.cluster(data, min = 1, max = 10)
```

Arguments

`data`	A numeric matrix.
`min`	The minimal number of components that is tested.
`max`	The maximal number of components that is tested.

Value

A list with 3 elements. The first element is the minimal AIC value for each tested number of components. The second element is a vector of all AIC values. The third is the actual clustering as returned by the EM algorithm using the optimal number of components according to AIC. The element is again a list that contains the mixture coefficients, the actual parameters of the mutlivariate Benroulli distributions, the probability matrix of each observation (i.e. row if `data`) and component and the number of iterations that the EM algorithm needed to converge.

Examples

```
#Random data generation, 100 observations, 5 dimensions, dependencies within the dimensions
data = cbind(round(runif(100)), round(runif(100)), round(runif(100)))
data = cbind(data, data[,2], 1-data[,3])

#Noisy data:
```

```
s = round(runif(2, 1, 100))
data[s, c(4,5)] = 1 - data[s, c(4,5)]

#MBMM Clustering
res = MBM.cluster(data, 1,8)
```

`merge.conceptmaps` *Unify sets of conceptmaps*

Description

merge takes two conceptsmaps objects and merges the underyling sets of conceptmaps.

Usage

```
## S3 method for class 'conceptmaps'
merge(x, y, ...)
```

Arguments

x	A conceptmaps object.
y	A conceptmaps object.
...	-

Value

A conceptmaps object that consist of the maps of x and y.

Examples

```
data = rbind(
cbind("1", "Object", "Class", "is instance of"),
cbind("1", "Object", "Attribute", "has"),
cbind("2", "Class", "Attribute", "possesses"),
cbind("2", "Attribute", "Data-type", "has"),
cbind("3", "Object", "Class", "is instance of")
)
cm1 = conceptmaps(data[1:2,])
cm2 = conceptmaps(data[3:5,])
merge(cm1, cm2)
```

modify.concepts *Modify the concepts of concept maps*

Description

modify.concepts modifies the list of concept of a concept map or of all maps of a set. For actual implementations see modify.concepts.conceptmap, or modify.concepts.conceptmaps.

Usage

```
modify.concepts(x, concept.list, ...)
```

Arguments

x	A conceptmap or conceptmaps object.
concept.list	A list of concepts.
...	-

Value

A conceptmaps or conceptmap object.

modify.concepts.conceptmap *Adapt list of concepts of a conceptmap object*

Description

modify.concepts modifes the list of concepts according to a given list. This includes removing concepts and adjacent propositions as well as adding (unconnected) concepts.

Usage

```
## S3 method for class 'conceptmap'
modify.concepts(x, concept.list, ...)
```

Arguments

x	A conceptmap object.
concept.list	A vector of strings that contains the list of concepts.
...	-

Value

A conceptmap object that encompasses exactly the concepts of `concept.list`. Concepts not originally in `map` are added as isolated nodes/concepts. Concepts of `map` that are not in `concept.list` are removed together with their adjacent propositions.

Examples

```
data = rbind(cbind("Object", "Class", "is instance of"), cbind("Class", "Attribute", "has"))
cm = conceptmap(data)
modify.concepts(cm, c("Object", "Class", "Method"))
```

`modify.concepts.conceptmaps` *Modifying the concepts of all maps of a conceptmaps object.*

Description

`modify.concepts` calls `modify.concepts.conceptmap` for each conceptmap object of a conceptmaps object. Therefore, all concept maps will share the same set of concepts afterwards.

Usage

```
## S3 method for class 'conceptmaps'
modify.concepts(x, concept.list, filter = F, ...)
```

Arguments

`x`	A conceptmaps object.
`concept.list`	A vector of strings that contains the list of concepts.
`filter`	If TRUE, concept maps that contain no propositions after the concept modification are removed from the result.
`...`	-

Value

A conceptmaps object that contains (possibly a subset of) the maps of `maps` in which every map contains the concepts of `concept.list`.

Examples

```
data = rbind(
cbind("1", "Object", "Class", "is instance of"),
cbind("1", "Object", "Attribute", "has"),
cbind("2", "Class", "Attribute", "possesses"),
cbind("2", "Attribute", "Data-type", "has"),
cbind("3", "Object", "Class", "is instance of")
)
```

```
cms = conceptmaps(data)

modify.concepts(cms, c("Object", "Class"), filter=TRUE)
```

`PAM.cluster` *Similarity based clustering*

Description

`PAM.cluster` calculates a clustering using the PAM algorithm (k-medoids). The quality of the clustering is judged using the G1 index.

Usage

```
PAM.cluster(data, min = 2, max = 10, metric = "manhattan")
```

Arguments

`data`	A numeric matrix.
`min`	The minimal number of components that is tested. Must be at least 2.
`max`	The maximal number of components that is tested.
`metric`	If empty, data will be treated as a distance matrix. Otherwise, the value will be passed to the call of `dist` to compute the distance matrix from `data`

Value

A list with 3 elements. The first element contains the optimal number of components according to the G1 index. The second element contains a vector of the G1 values. The thrid element contains the clustering itself, i.e. the return value of `PAM`.

Examples

```
## Not run:
#Random data generation, 10 dimensions, 500 observations, 2 clusters
require("gtools")
data = c()
p = 0.0
for (i in 1:2)
{
temp = c()
for (j in 1:10)
temp = cbind(temp, rbinom(250, 1, p+(i-1)*0.5+(0.025*i)*j))
data=rbind(data, temp)
}
data = data[permute(1:500),]

PAM.cluster(data)

## End(Not run)
```

pathfinder *Construct a Pathfinder network from a conceptmap or a concept landscape*

Description

pathfinder creates Pathfinder network. For more information and actual implementations see pathfinder.matrix, pathfinder.conceptmaps, or pathfinder.igraph.

Usage

```
pathfinder(data, q, r, ...)
```

Arguments

data	The input data.
q	The q parameter of the Pathfinder algorithm.
r	The r parameter of the Pathfinder algorithm.
...	-

Value

The Pathfinder network of the input data.

pathfinder.conceptmaps *Creating a Pathfinder network from a conceptmaps object*

Description

pathfinder creates the Pathfinder network from a given set of conceptmaps. The conecpts of each concept map are unified, then the concept maps are transformed into a weight matrix and pathfinder.matrix is called on the data.

Usage

```
## S3 method for class 'conceptmaps'
pathfinder(data, q = 2, r = 1, threshold = 0,
  directed = F, prune.edges = F, return.cm = F, filename = "", ...)
```

Arguments

data	A conceptmaps object.
q	The parameter q used in the Pathfinder algorithm. The resulting graph will be q-triangular.
r	The parameter r used in the Pathfinder algorithm for the r-metric.
threshold	A numeric value used for pruning the graph before the Pathfinder algorithm. The pruning works in conjunction with the value of prune.edges.
directed	if TRUE, the direction of the edges will be kept and the resulting Pathfinder network will be directed as well.
prune.edges	If TRUE, each entry of the weight matrix that is lower than threshold will be set to 0 and columns with a resulting sum of 0 are removed. If FALSE, only columns of the weight matrix with a sum of less than threshold will be removed.
return.cm	If TRUE, a conceptmap object will be returned. Otherwise, an igrpah object will be returned.
filename	Optional. If specified, the resulting Pathfinder network will be stored in TGF format in the given file.
...	-

Value

Depending on return.cm either an igraph object or a conceptmap object that represents the Pathfinder network. If an igraph object is returned, the graph will be weighted.

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))

#Create Pathfinder network from data and return a conceptmap object
cm = pathfinder(simple_cms, q=1, return.cm=TRUE)
```

---

pathfinder.igraph *Creating a Pathfinder network from an igraph object*

Description

pathfinder creates the Pathfinder network from a weighted graph based on pathfinder.matrix. It is a convenience method that can be called on the result of a call to landscape

Usage

```
## S3 method for class 'igraph'
pathfinder(data, q = 2, r = 1, threshold = 0,
  prune.edges = F, filename = "", ...)
```

Arguments

data	An igraph object.
q	The parameter q used in the Pathfinder algorithm. The resulting graph will be q-triangular.
r	The parameter r used in the Pathfinder algorithm for the r-metric.
threshold	A numeric value used for pruning the graph before the Pathfinder algorithm. The pruning works in conjunction with the value of prune.edges.
prune.edges	If TRUE, each entry of the weight matrix that is lower than threshold will be set to 0 and columns with a resulting sum of 0 are removed. If FALSE, only columns of the weight matrix with a sum of less than threshold will be removed.
filename	Optional. If specified, the resulting Pathfinder network will be stored in TGF format in the given file.
...	-

Value

An igraph object that represents the Pathfinder network as a weighted graph.

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))
pathfinder(landscape(simple_cms, result="graph", mode="undirected"))
```

---

pathfinder.matrix *Creating a Pathfinder network from a matrix*

Description

pathfinder creates the Pathfinder network from a given weight matrix.

Usage

```
## S3 method for class 'matrix'
pathfinder(data, q, r, ...)
```

Arguments

data	A non-negative weight matrix of a graph that can be either directed or undirected.
q	The parameter q used in the Pathfinder algorithm. The resulting graph will be q-triangular.
r	The parameter r used in the Pathfinder algorithm for the r-metric.
...	-

Value

A numeric weight matrix that represented the Pathfinder graph of the input graph.

See Also

The Pathfinder algorithm is implemented based on the description in: Dearholt, Donald W.; Schvaneveldt, Roger W. (1990): Properties of Pathfinder Netowrks. In: Roger W. Schvaneveldt (Hg.): Pathfinder associative networks. Studies in knowledge organizations. Norwood, N.J: Ablex Pub. Corp., S. 1-30.

Examples

```
#Manually create a weighted graph
data = matrix(data = 0, nrow = 6, ncol=6)
colnames(data) <- c("Object", "Class", "Method", "Attribute", "Visibility", "Algorithm")
rownames(data) <- c("Object", "Class", "Method", "Attribute", "Visibility", "Algorithm")
data["Object", "Class"] = 3
data["Object", "Method"] = 3
data["Object", "Attribute"] = 10
data["Object", "Visibility"] = Inf
data["Object", "Algorithm"] = 9
data["Class", "Method"] = 7
data["Class", "Attribute"] = 6
data["Class", "Visibility"] = 8
data["Class", "Algorithm"] = 10
data["Method", "Attribute"] = 4
data["Method", "Visibility"] = 9
data["Method", "Algorithm"] = 3
data["Attribute", "Visibility"] = 5
data["Attribute", "Algorithm"] = 10
data["Visibility", "Algorithm"] = Inf

data = data + t(data)

#Run the Pathfinder algorithm with several different parameters
pathfinder(data, 5, 1)
```

```
pathfinder(data, 2, 1)
pathfinder(data, 5,Inf)
pathfinder(data, 2, Inf)
```

`plot.conceptmap` *Plotting a conceptmap*

Description

`plot` plots a concept map. Includes finding a good layout based on communities and a circular layout. Is especially suited for plotting larger concept maps, in particular amalgamations.

Usage

```
## S3 method for class 'conceptmap'
plot(x, edge.labels = T, max.label.len = 25,
  scale = 1, layout = NULL, ...)
```

Arguments

`x`	A conceptmap object.
`edge.labels`	If TRUE, the labels of edges will be plotted as well.
`max.label.len`	The maximal length of labels (in characters) that are plotted completely. Longer labels will be shortend by "...".
`scale`	Overall scaling factor that is applied to the plot.
`layout`	If not NULL, must either be one of "fruchterman.reingold", "kamada.kawai", "spring" or "reingold.tilford" or a numeric matrix. If it is a string, the corresponding layouting algorithm of the igraph package will be called. If it is a numeric matrix, it must contain a row for each concept and two columns that determine the x and y coordinates of this concept. Then this layout will be used directly.
`...`	-

Value

-

Examples

```
#Create concept map from a random graph
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
E(g1)$name <- rep("", length(E(g1)))
plot(conceptmap(g1), edge.labels=FALSE, layout="kamada.kawai")
```

plot.conceptmaps *Plotting a series of concept maps*

Description

plot plots a set of concept maps. The layout is determined based on the union of all concept maps, then each map is individually plotted using this fixed layout. Is escpecially useful for visualizing horizontal landscapes.

Usage

```
## S3 method for class 'conceptmaps'
plot(x, edge.labels = T, max.label.len = 25,
  scale = 1, layout = NULL, ...)
```

Arguments

x	A conceptmaps object.
edge.labels	If TRUE, the labels of edges will be plotted as well.
max.label.len	The maximal length of labels (in characters) that are plotted completely. Longer labels will be shortend by "...".
scale	Overall scaling factor that is applied to the plot.
layout	If not NULL, must be one of "fruchterman.reingold", "kamada.kawai", "spring" or "reingold.tilford". The corresponding layouting algorithm of the igraph package will be called. If it is NULL, the layouting based on communities and a circular layout will be used.
...	-

Value

-

Examples

```
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
E(g1)$name <- rep("", length(E(g1)))
E(g2)$name <- rep("", length(E(g2)))
E(g3)$name <- rep("", length(E(g3)))
#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))

plot(simple_cms, layout="spring")
```

print.conceptmap *Display basic information of a conceptmap object*

Description

print displays basic information. For plotting, see plot.conceptmap

Usage

```
## S3 method for class 'conceptmap'
print(x, ...)
```

Arguments

x	A conceptmap object.
...	-

Value

-

print.conceptmaps *Display basic information of a conceptmaps object*

Description

print displays basic information. For plotting, see plot.conceptmaps

Usage

```
## S3 method for class 'conceptmaps'
print(x, ...)
```

Arguments

x	A conceptmaps object.
...	-

Value

-

read.folder.tgf *Importing a set of concept maps from TGF files.*

Description

`read.folder.tgf` reads several TGF files and imports them as a conceptmaps object.

Usage

```
read.folder.tgf(folder, strip = TRUE)
```

Arguments

`folder`	The path of a folder in which every TGF file is read.
`strip`	Passed to the call of `read.tgf` that is used to import the single concept maps.

Value

A list consisting of a conceptmaps object and the list of filenames (in the same order as the maps in the conceptmaps object).

Examples

```
## Not run:
#Assuming that the data is in the folder "~/cmaps"
cm = read.folder.tgf("~/cmaps")

## End(Not run)
```

read.folder.yEd *Importing a set of concept maps from GraphML files.*

Description

`read.folder.yEd` reads several graphML files that were created by yEd and imports them as a conceptmaps object.

Usage

```
read.folder.yEd(folder, strip = TRUE)
```

Arguments

`folder`	The path of a folder in which every graphML file is read.
`strip`	Passed to the call of `read.yEd` that is used to import the single concept maps.

Value

A list consisting of a conceptmaps object and the list of filenames (in the same order as the maps in the conceptmaps object).

Examples

```
## Not run:
#Assuming that the data is in the folder "~/cmaps"
cm = read.folder.yEd("~/cmaps")

## End(Not run)
```

`read.tgf` *Importing a concept map from a TGF file.*

Description

`read.tgf` reads a TGF file and imports the graph as a conceptmap object.

Usage

```
read.tgf(file, strip = TRUE)
```

Arguments

`file`	The filename and path that should be read.
`strip`	Passed to the call of `conceptmap.igraph` that is used to create the conceptmap object.

Value

A conceptmap object.

Examples

```
## Not run:
#Assuming that the data is in "~/cmap.tgf"
cm = read.tgf("~/cmap.tgf")

## End(Not run)
```

read.yEd *Importing a concept map from a GraphML file.*

Description

read.yEd reads a graphML file that was created by yEd and imports the graph as a conceptmap object.

Usage

```
read.yEd(file, strip = TRUE)
```

Arguments

file	The filename and path that should be read.
strip	Passed to the call of conceptmap.igraph that is used to create the conceptmap object.

Value

A conceptmap object.

Examples

```
## Not run:
#Assuming that the data is in "~/cmap.graphml"
cm = read.yEd("~/cmap.graphml")

## End(Not run)
```

splice *Select a subset of a set of conceptmaps*

Description

splice selects a subset of a set of concept maps and returns them as a new conceptmaps object.

Usage

```
splice(maps, keep)
```

Arguments

maps	A conceptmaps object.
keep	A numeric vector containing the indices of the maps in maps that should be retained in the subset. Regular R list indexing is used.

Value

A conceptmaps object that consist of the maps with indiced of `maps`.

Examples

```
data = rbind(
cbind("1", "Object", "Class", "is instance of"),
cbind("1", "Object", "Attribute", "has"),
cbind("2", "Class", "Attribute", "possesses"),
cbind("2", "Attribute", "Data-type", "has"),
cbind("3", "Object", "Class", "is instance of")
)
cms = conceptmaps(data)

splice(cms, c(1,3))
```

`summary.conceptmap` *Return basic information of a conceptmap object*

Description

`summary` returns basic information about a conceptmap object

Usage

```
## S3 method for class 'conceptmap'
summary(object, ...)
```

Arguments

`object`	A conceptmap object.
`...`	-

Value

A list with the number of concepts, edges and components of the concept map.

summary.conceptmaps *Return basic information of a conceptmaps object*

Description

summary returns basic information about a conceptmaps object

Usage

```
## S3 method for class 'conceptmaps'
summary(object, ...)
```

Arguments

object	A conceptmaps object.
...	-

Value

A matrix with one column for each concept map in the set and the number of concepts, edges, and components of this map respectively in 3 rows.

unify.concepts *Unifying the concepts of a conceptmap object.*

Description

unify.concepts first calls get.unified.concepts on the maps of a conceptmaps object and then calls modify.concepts.conceptmaps on each of the constituent maps. Afterwards, therefore, each map of the conceptmaps object will share the same common superset of concepts.

Usage

```
unify.concepts(maps)
```

Arguments

maps	A conceptmaps object.

Value

A conceptmaps object in of the same map of maps, in which every map shares the same concepts.

Examples

```
data = rbind(
cbind("1", "Object", "Class", "is instance of"),
cbind("1", "Object", "Attribute", "has"),
cbind("2", "Class", "Attribute", "possesses"),
cbind("2", "Attribute", "Data-type", "has"),
cbind("3", "Object", "Class", "is instance of")
)
cms = conceptmaps(data)

unify.concepts(cms)
```

---

write.tgf *Saving a concept map to a TGF file*

Description

write.tgf stores the graph underlying a conceptmap object into a file using the "Trivial Grpah Format" (TGF).

Usage

```
write.tgf(map, file, translation = NULL)
```

Arguments

map	A conceptmap object.
file	The location including filename where the file should be stored.
translation	If not NULL, a vector of strings of equal length as the number of concepts used in the concept map. Then, the names given in this vector will be used in the file instead of the original concepts. Can be used, for example, to translate the concepts into a different language for export.

Value

-

Examples

```
## Not run:
#Create concept map from a random graph
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

write.tgf(conceptmap(g1), "~/cmap.tgf",
          translation = c("Node_1", "Node_2", "Node_3", "Node_4", "Node_5"))

## End(Not run)
```

write.tgf.folder *Saving a set of concept maps to TGF files*

Description

write.tgf.folder stores the graphs underlying the maps of a conceptmaps object into a folder using the "Trivial Grpah Format" (TGF). The function calls write.tgf for each of the maps of a conceptmaps object. The files will be named "1.tgf", "2.tgf" and so on.

Usage

```
write.tgf.folder(maps, folder, translation = NULL)
```

Arguments

maps	A conceptmap object.
folder	The location where the files should be stored. The folder is created, if necessary.
translation	See write.tgf.

Value

-

Examples

```
## Not run:
#Create concept maps from three random graphs
require("igraph")
g1 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g2 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)
g3 = set.vertex.attribute(erdos.renyi.game(5, 0.7, type="gnp"), "name", value=1:5)

#Create conceptmaps object from three conceptmap objects
simple_cms = conceptmaps(list(conceptmap(g1), conceptmap(g2), conceptmap(g3)))

write.tgf.folder(simple_cms, "~/cmaps")

## End(Not run)
```

Index