Package 'relations'

July 23, 2025

Version 0.6-15

Encoding UTF-8

Title Data Structures and Algorithms for Relations

Description Data structures and algorithms for k-ary relations with arbitrary domains, featuring relational algebra, predicate functions, and fitters for consensus relations.

LazyData yes

Depends R (>= 3.2.0)

Imports cluster, stats, slam, sets (>= 1.0-16), graphics, grDevices

Suggests Rgraphviz, clue (>= 0.3-49), Rglpk (>= 0.3-1), lpSolve (>= 5.6.3), Rsymphony (>= 0.1-9), methods

Enhances seriation, Rcplex

License GPL-2

NeedsCompilation no

Author David Meyer [aut] (ORCID: <https://orcid.org/0000-0002-5196-3048>), Kurt Hornik [aut, cre] (ORCID: <https://orcid.org/0000-0003-4198-9911>), Christian Buchta [ctb]

Maintainer Kurt Hornik <Kurt.Hornik@R-project.org>

Repository CRAN

Date/Publication 2025-02-19 08:34:40 UTC

Contents

`algebra` *Relational Algebra*

Description

Various "relational algebra"-like operations.

Usage

```
relation_projection(x, margin = NULL)
relation_selection(x, subset)
relation_cartesian(x, y, ...)
relation_complement(x, y)
relation_intersection(x, y, ...)
relation_union(x, y, ...)
relation_symdiff(x, y)
relation_division(x, y)
relation_remainder(x, y)
relation_join(x, y, ...)
relation_semijoin(x, y, ...)
relation_antijoin(x, y, ...)
```

Arguments

x, y	Relation objects.
margin	Either a character vector of domain names, or an integer vector of domain indices.
subset	Expression resulting in a logical vector of length equal to the number of tuples in the graph.
...	Relation objects for `relation_cartesian()`, `relation_intersection()`, and `relation_union()`. Otherwise, passed to `merge()`.

Details

These functions provide functionality similar to the corresponding operations defined in relational algebra theory as introduced by Codd (1970). Note, however, that domains in database relations, unlike the concept of relations we use here, are unordered. In fact, a database relation ("table") is defined as a set of elements called "tuples", where the "tuple" components are named, but unordered. So in fact, a "tuple" in this sense is a set of mappings from the attribute names into the union of the attribute domains.

The *projection* of a relation on a specified margin (i.e., a vector of domain names or indices) is the relation obtained when all tuples are restricted to this margin. As a consequence, duplicate tuples are removed.

The *selection* of a relation is the relation obtained by taking a subset of the relation graph, defined by some logical expression.

The *Cartesian product* of two relations is obtained by basically building the Cartesian product of all graph elements, but combining the resulting pairs into single tuples.

The *union* of two relations simply combines the graph elements of both relations; the *complement* of two relations R and S removes the tuples of S from R.

The *intersection* (*symmetric difference*) of two relations is the relation with all tuples they have (do not have) in common.

The *division* of relation R by relation S is the reversed Cartesian product. The result is a relation with the domain unique to R and containing the maximum number of tuples which, multiplied by S, are contained in R. The *remainder* of this operation is the complement of R and the division of R by S. Note that for both operations, the domain of S must be contained in the domain of R.

The (natural) *join* of two relations is their Cartesian product, restricted to the subset where the elements of the common attributes do match. The left/right/full outer join of two relations R and S is the union of $R/S/R$ and S, and the inner join of R and S. The implementation uses `merge()`, and so the left/right/full outer joins are obtained by setting `all.x`/`all.y`/`all` to `TRUE` in `relation_join()`. The domains to be matched are specified using `by`.

The left (right) *semijoin* of two relations R and S is the join of these, projected to the attributes of R (S). Thus, it yields all tuples of R (S) participating in the join of R and S.

The left (right) *antijoin* of two relations R and S is the complement of R (S) and the join of both, projected to the attributes of R (S). Thus, it yields all tuples of R (S) *not* participating in the join of R and S.

The operators `%><%`, `%=><%`, `%><=%`, `%=><=%`, `%|><%`, `%><|%`, `%|><|%`, `%|>%`, `%<|%`, and `%U%` can be used for the Cartesian product, left outer join, right outer join, full outer join, left semi-join, right semi-join, join, left antijoin, right antijoin, and union, respectively.

References

E. F. Codd (1970), A relational model of data for large shared data banks. *Communications of the ACM*, **13**/6, 377–387. doi:10.1145/362384.362685.

See Also

relation()

Examples

```
## projection
Person <-
    data.frame(Name = c("Harry", "Sally", "George", "Helena", "Peter"),
               Age = c(34, 28, 29, 54, 34),
               Weight = c(80, 64, 70, 54, 80),
               stringsAsFactors = FALSE)
Person <- as.relation(Person)
relation_table(Person)
relation_table(relation_projection(Person, c("Age", "Weight")))

## selection
relation_table(R1 <- relation_selection(Person, Age < 29))
relation_table(R2 <- relation_selection(Person, Age >= 34))
relation_table(R3 <- relation_selection(Person, Age == Weight))

## union
relation_table(R1 %U% R2)

## works only for the same domains:
relation_table(R2 | R3)

## complement
relation_table(Person - R2)

## intersection
relation_table(relation_intersection(R2, R3))

## works only for the same domains:
relation_table(R2 & R3)

## symmetric difference
relation_table(relation_symdiff(R2, R3))

## Cartesian product
Employee <-
    data.frame(Name =
               c("Harry", "Sally", "George", "Harriet", "John"),
               EmpId = c(3415, 2241, 3401, 2202, 3999),
               DeptName =
               c("Finance", "Sales", "Finance", "Sales", "N.N."),
       stringsAsFactors = FALSE)
Employee <- as.relation(Employee)
```

```
relation_table(Employee)
Dept <- data.frame(DeptName = c("Finance", "Sales", "Production"),
                   Manager = c("George", "Harriet", "Charles"),
                   stringsAsFactors = FALSE)
Dept <- as.relation(Dept)
relation_table(Dept)

relation_table(Employee %><% Dept)

## Natural join
relation_table(Employee %|><|% Dept)

## left (outer) join
relation_table(Employee %=><% Dept)

## right (outer) join
relation_table(Employee %><=% Dept)

## full outer join
relation_table(Employee %=><=% Dept)

## antijoin
relation_table(Employee %|>% Dept)
relation_table(Employee %<|% Dept)

## semijoin
relation_table(Employee %|><% Dept)
relation_table(Employee %><|% Dept)

## division
Completed <-
    data.frame(Student = c("Fred", "Fred", "Fred", "Eugene",
                           "Eugene", "Sara", "Sara"),
               Task = c("Database1", "Database2", "Compiler1",
                        "Database1", "Compiler1", "Database1",
                        "Database2"),
               stringsAsFactors = FALSE)
Completed <- as.relation(Completed)
relation_table(Completed)
DBProject <- data.frame(Task = c("Database1", "Database2"),
                        stringsAsFactors = FALSE)
DBProject <- as.relation(DBProject)
relation_table(DBProject)

relation_table(Completed %/% DBProject)

## division remainder
relation_table(Completed %% DBProject)
```

Cetacea *Cetacea Data*

Description

A data set with 16 variables on 36 different types of cetacea.

Usage

```
data("Cetacea")
```

Format

A data frame with 36 observations on 16 categorical variables. The first 15 variables relate to morphology, osteology, or behavior, and have both self-explanatory names and levels. The last (CLASS) gives a common zoological classification.

Source

G. Vescia (1985). Descriptive classification of Cetacea: Whales, porpoises, and dolphins. In: J. F. Marcotorchino, J. M. Proth, and J. Janssen (eds.), Data analysis in real life environment: ins and outs of solving problems. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands.

Examples

```
data("Cetacea")
summary(Cetacea)
## Show the cetacea types grouped by class.
split(rownames(Cetacea), Cetacea$CLASS)
```

---

charfun *Relation Characteristic Functions*

Description

Determine the characteristic function of a relation.

Usage

```
relation_charfun(x, components = FALSE)
```

Arguments

x	an object inheriting from class relation.
components	a logical indicating whether the characteristic function created should take vectors (each vector corresponding to one domain) as argument, or a data frame (with the elements in the rows). In the former case, all vectors are recycled to fit the longest vector in case of binary relations.

See Also

relation()

Examples

```
## Relation 'a divides b':
divides <- function(a, b) b %% a == 0
R <- relation(domain = list(1 : 10, 1 : 10), charfun = divides)
R
## 'Recover' characteristic function:
"%|%" <- relation_charfun(R)

## Use it.
2L %|% 6L
2:4 %|% 6L
2L %|% c(2:3, 6L)

## This also works:
"%|%"(2L, 6L)
## (and more generally does for arities > 2).
```

`choice` *Relation-Based Choices*

Description

Choose objects based on an ensemble of relations between these.

Usage

```
relation_choice(x, method = "symdiff", weights = 1,
                control = list(), ...)
```

Arguments

`x`	an ensemble of endorelations.
`method`	a character string specifying one of the built-in methods, or a function to be taken as a user-defined method. See **Details** for available built-in methods.
`weights`	a numeric vector with non-negative case weights. Recycled to the number of elements in the ensemble given by `x` if necessary.
`control`	a list of control parameters. See **Details**.
`...`	a list of control parameters (overruling those specified in `control`).

Details

A social choice function is a rule for *choosing* from a set X of objects, i.e., selecting suitable subsets of X. Voting rules used in elections are the most prominent example of such functions, which typically aggregate individual preferences (e.g., of voters).

Choice methods `"symdiff"`, `"CKS"`, `"PC"` and `"euclidean"` choose a given number k of objects ("winners") by determining a relation R minimizing $\sum_b w_b d(R_b, R)^e$ over all relations for which

winners are always strictly preferred to losers, without any further constraints on the relations between pairs of winners or pairs of losers, where d is symmetric difference (symdiff, "Kemeny-Snell"), Cook-Kress-Seiford (CKS), generalized paired comparison, or Euclidean dissimilarity, respectively, and w_b is the case weight given to R_b. For symdiff, CKS and PC choice, the R_b must be crisp endorelations, and $e = 1$; for Euclidean choice, the R_b can be crisp or fuzzy endorelations, and $e = 2$. (Note that solving such a choice problem is different from computing consensus preference relations.) See `relation_dissimilarity()` for more information about these dissimilarities.

Available control options include:

`k` an integer giving the number of objects/winners to be chosen.

`n` the maximal number of optimal choices to be obtained, with `NA` constants or `"all"` indicating to obtain all optimal choices. By default, only a single optimal choice is computed.

For the general PC case, the discrepancies can be specified via the `delta` control option.

Choice method `"Schulze"` implements the Schulze method for selecting winners from (votes expressing) preferences. See e.g. https://en.wikipedia.org/wiki/Schulze_method for details. Currently, the Schulze heuristic is used, and the set of all possible winners is returned.

Value

A set with the chosen objects, or a list of such sets.

Examples

```
data("SVM_Benchmarking_Classification")
## Determine the three best classification learners in the above sense.
if(requireNamespace("Rglpk", quietly = TRUE)) {
    relation_choice(SVM_Benchmarking_Classification, k = 3)
}
```

`classes` *Relation Equivalence Classes*

Description

Provide class ids or classes, respectively, for an equivalence relation or the indifference relation of a weak order.

Usage

```
relation_class_ids(x)
relation_classes(x)
```

Arguments

`x`	an object inheriting from class `relation` representing a crisp endorelation.

Value

For `relation_class_ids()`, a numeric vector with class ids corresponding to the classes of the equivalence relation, or the indifference relation of the weak order with ids ordered according to increasing preference.

For `relation_classes()`, an object of class `relation_classes_of_objects`, which is a list of sets giving the elements in the corresponding classes, named by the class ids.

Examples

```
## Equivalence.
f <- factor(rep(c("Good", "Bad", "Ugly"), c(3, 2, 1)))
R <- as.relation(f)
relation_is(R, "equivalence")
table(ids = relation_class_ids(R), orig = f)

relation_classes(R)

## Weak order ("weak preference").
f <- ordered(f, levels = c("Ugly", "Bad", "Good"))
R <- as.relation(f)
relation_is(R, "weak_order")
table(ids = relation_class_ids(R), orig = f)

relation_classes(R)
```

---

closure *Transitive and Reflexive Closure*

Description

Computes transitive and reflexive closure of an endorelation.

Usage

```
transitive_closure(x)
reflexive_closure(x)
## S3 method for class 'relation'
closure(x, operation = c("transitive", "reflexive"), ...)
```

Arguments

`x`	an R object inheriting from class `relation`, representing an endorelation.
`operation`	character string indicating the kind of closure.
`...`	currently not used.

Details

Let R be an endorelation on X and n be the number of elements in X.

The *transitive closure* of R is the smallest transitive relation on X that contains R. The code implements Warshall's Algorithm which is of complexity $O(n^3)$.

The *reflexive closure* of R is computed by setting the diagonal of the incidence matrix to 1.

References

S. Warshall (1962), A theorem on Boolean matrices. *Journal of the ACM*, **9**/1, 11–12. doi:10.1145/321105.321107.

See Also

relation(), reflexive_reduction(), transitive_reduction(), closure().

Examples

```
R <- as.relation(1 : 5)
relation_incidence(R)

## transitive closure/reduction
RR <- transitive_reduction(R)
relation_incidence(RR)
R == transitive_closure(RR)

## same
require("sets") # closure() and reduction()
R == closure(reduction(R))

## reflexive closure/reduction

RR <- reflexive_reduction(R)
relation_incidence(RR)
R == reflexive_closure(RR)
## same:
R == closure(reduction(R, "reflexive"), "reflexive")
```

---

components *Connected components*

Description

Computes (strongly or weakly) connected components of an endorelation.

Usage

```
relation_connected_components(x, type = c("strongly", "weakly"))
relation_condensation(x)
relation_component_representation(x)
```

Arguments

x	an R object inheriting from class relation, representing a crisp endorelation without missings.
type	character string indicating the kind of components sought.

Details

Let G be the graph of an endorelation R.

A *weakly* connected component of some node k in G is the set of all nodes reachable from k. A *strongly* connected component of some node k is the set of all nodes reachable from k, from which k can be reached. Each component is represented by some element, the *leader*.

The *component representation* graph of a cyclic endorelation R is composed of directed cycles, one for each strongly connected component of R containing more than one element, linking all corresponding elements.

The *condensation* of R is the graph of all leaders of R.

Value

For relation_connected_components(), an object of class relation_classes_of_objects, i.e., a list of sets giving the elements of the corresponding connected components, named by the leaders' character representation. The list of leaders is added as a leaders attribute.

For relation_condensation(), an (acyclic) endorelation.

For relation_component_representation(), an endorelation with same domain as x.

References

S. Warshall (1962), A theorem on Boolean matrices. *Journal of the ACM*, **9**/1, 11–12. doi:10.1145/321105.321107.

J. A. La Poutré and J. van Leeuwen (1988), Maintenance of Transitive Closures and Transitive Reductions of Graphs. *Proceedings of the International Workshop of Graph-Theoretic Concepts in Computer Science*, Springer, London, 106–120.

See Also

plot.relation(), transitive_reduction()

Examples

```
## example from La Poutre and van Leeuwen:

require("sets") # set(), pair() etc.

G <- set(pair(1L, 2L), pair(2L, 1L), pair(1L, 3L), pair(3L, 1L),
         pair(3L, 7L), pair(2L, 5L), pair(2L, 6L), pair(6L, 5L),
         pair(5L, 7L), pair(4L, 6L), pair(5L, 4L), pair(4L, 7L))

R <- endorelation(graph = G)
```

```
relation_connected_components(R)
relation_graph(relation_condensation(R))
relation_graph(relation_component_representation(R))
```

---

consensus *Consensus Relations*

Description

Compute consensus relations of a relation ensemble.

Usage

```
relation_consensus(x, method = NULL, weights = 1,
                   control = list(), ...)
```

Arguments

x	an ensemble of relations (see relation_ensemble()), or something which can be coerced to such.
method	a character string specifying one of the built-in methods for computing consensus relations, or a function to be taken as a user-defined method, or NULL (default value). If a character string, its lower-cased version is matched against the lower-cased names of the available built-in methods using pmatch(). See **Details** for available built-in methods and defaults.
weights	a numeric vector with non-negative case weights. Recycled to the number of elements in the ensemble given by x if necessary.
control	a list of control parameters. See **Details**.
...	a list of control parameters (overruling those specified in control).

Details

Consensus relations "synthesize" the information in the elements of a relation ensemble into a single relation, often by minimizing a criterion function measuring how dissimilar consensus candidates are from the (elements of) the ensemble (the so-called "optimization approach"), typically of the form $\Phi(R) = \sum w_b d(R_b, R)^e$, where d is a suitable dissimilarity measure (see relation_dissimilarity()), w_b is the case weight given to element R_b of the ensemble, and $e \geq 1$. Such consensus relations are called "central relations" in Régnier (1965). For $e = 1$, we obtain (generalized) medians; $e = 2$ gives (generalized) means (least squares consensus relations).

Available built-in methods are as follows. Apart from Condorcet's and the unrestricted Manhattan and Euclidean consensus methods, these are applicable to ensembles of endorelations only.

"Borda" the consensus method proposed by Borda (1781). For each relation R_b and object x, one determines the Borda/Kendall scores, i.e., the number of objects y such that yR_bx. These are then aggregated across relations by weighted averaging. Finally, objects are ordered according to their aggregated scores. Note that this may result in a weak order (i.e., with objects being tied).

One can enforce a linear order by setting the control parameter `L` to `TRUE`, and obtain a relation ensemble with up to *n* or all such solutions by additionally setting the control parameter `n` to some positive integer or `"all"`, respectively.

`"Copeland"` the consensus method proposed by Copeland (1951). For each relation R_b and object x, one determines the Copeland scores, i.e., the number of objects y such that yR_bx, minus the number of objects y such that xR_by. Like the Borda method, these are then aggregated across relations by weighted averaging. Finally, objects are ordered according to their aggregated scores. Note that this may result in a weak order (i.e., with objects being tied).

One can enforce a linear order by setting the control parameter `L` to `TRUE`, and obtain a relation ensemble with up to *n* or all such solutions by additionally setting the control parameter `n` to some positive integer or `"all"`, respectively.

`"Condorcet"` the consensus method proposed by Condorcet (1785). For a given ensemble of crisp relations, this minimizes the criterion function Φ with d as symmetric difference distance and $e = 1$ over all possible crisp relations. In the case of endorelations, consensus is obtained by weighting voting, such that xRy if the weighted number of times that xR_by is no less than the weighted number of times that this is not the case. Even when aggregating linear orders, this can lead to intransitive consensus solutions ("effet Condorcet").

One can obtain a relation ensemble with up to *n* or all such solutions consensus relations by setting the control parameter `n` to some positive integer or `"all"`, respectively.

`"CS"` the consensus method of Cook and Seiford (1978) which determines a linear order minimizing the criterion function Φ with d as generalized Cook-Seiford (ranking) distance and $e = 1$ via solving a linear sum assignment problem.

One can obtain a relation ensemble with up to *n* or all such consensus relations by setting the control parameter `n` to some positive integer or `"all"`, respectively.

`"symdiff/`*F*`"` an exact solver for determining the consensus relation of an ensemble of crisp endorelations by minimizing the criterion function Φ with d as symmetric difference ("symdiff") distance and $e = 1$ over a suitable class ("Family") of crisp endorelations as indicated by *F*, with values:

`G` general (crisp) endorelations.

`A` antisymmetric relations.

`C` complete relations.

`E` equivalence relations: reflexive, symmetric, and transitive.

`L` linear orders: complete, reflexive, antisymmetric, and transitive.

`M` matches: complete and reflexive.

`O` partial orders: reflexive, antisymmetric and transitive.

`S` symmetric relations.

`T` tournaments: complete, irreflexive and antisymmetric (i.e., complete and asymmetric).

`W` weak orders (complete preorders, preferences, "orderings"): complete, reflexive and transitive.

`preorder` preorders: reflexive and transitive.

`transitive` transitive relations.

Can also be referred to as `"SD/`*F*`"`.

Consensus relations are determined by reformulating the consensus problem as a binary program (for the relation incidences), see Hornik and Meyer (2007) for details. The solver

employed can be specified via the control argument `solver`, with currently possible values `"glpk"`, `"lpsolve"`, `"symphony"` or `"cplex"` or a unique abbreviation thereof, specifying to use the solvers from packages **Rglpk** (default), **lpSolve**, **Rsymphony**, or **Rcplex**, respectively. Unless control option `sparse` is false, a sparse formulation of the binary program is used, which is typically more efficient.

For fitting equivalences and weak orders (cases `E` and `W`) it is possible to specify the number of classes k using the control parameter `k`. For fitting weak orders, one can also specify the number of elements in the classes via control parameter `l`.

Additional constraints on the incidences of the consensus solution can be given via the control parameter `constraints`, in the form of a 3-column matrix whose rows give row and column indices i and j and the corresponding incidence I_{ij}. (I.e., incidences can be constrained to be zero or one on an object by object basis.)

One can obtain a relation ensemble with up to *n* or all such consensus relations by setting the control parameter `n` to some positive integer or `"all"`, respectively. (See the examples.)

`"manhattan"` the (unrestricted) median of the ensemble, minimizing Φ with d as Manhattan (symmetric difference) distance and $e = 1$ over all (possibly fuzzy) relations.

`"euclidean"` the (unrestricted) mean of the ensemble, minimizing Φ with d as Euclidean distance and $e = 2$ over all (possibly fuzzy) relations.

`"euclidean/`F`"` an exact solver for determining the restricted least squares Euclidean consensus relation of an ensemble of endorelations by minimizing the criterion function Φ with d as Euclidean difference distance and $e = 2$ over a suitable family of crisp endorelations as indicated by F, with available families and control parameters as for methods `"symdiff/`F`"`.

`"majority"` a generalized majority method for which the consensus relation contains of all tuples occurring with a relative frequency of more than $100p$ percent (of 100 percent if $p = 1$). The fraction p can be specified via the control parameter `p`. By default, $p = 1/2$ is used.

`"CKS/`F`"` an exact solver for determining the consensus relation of an ensemble of crisp endorelations by minimizing the criterion function Φ with d as Cook-Kress-Seiford ("CKS") distance and $e = 1$ over a suitable class ("Family") of crisp endorelations as indicated by F, with available families and control parameters as for methods `"symdiff/`F`"`.

For fitting equivalences and weak orders (cases `E` and `W`) it is possible to specify the number of classes k using the control parameter `k`.

One can obtain a relation ensemble with up to *n* or all such consensus relations by setting the control parameter `n` to some positive integer or `"all"`, respectively.

`"PC/`F`"` an exact solver for determining the consensus relation of an ensemble of crisp endorelations by minimizing the criterion function Φ with d as (generalized) paired comparison ("PC") distance and $e = 1$ over a suitable class ("Family") of crisp endorelations as indicated by F, with available families and control parameters as for methods `"symdiff/`F`"`, and control option `delta` for specifying the paired comparison discrepancies.

For fitting equivalences and weak orders (cases `E` and `W`) it is possible to specify the number of classes k using the control parameter `k`.

One can obtain a relation ensemble with up to *n* or all such consensus relations by setting the control parameter `n` to some positive integer or `"all"`, respectively.

Value

The consensus relation(s).

References

J. C. Borda (1781), Mémoire sur les élections au scrutin. Histoire de l'Académie Royale des Sciences.

W. D. Cook and M. Kress (1992), *Ordinal information and preference structures: decision models and applications*. Prentice-Hall: New York. ISBN: 0-13-630120-7.

W. D. Cook and L. M. Seiford (1978), Priority ranking and consensus formation. *Management Science*, **24**/16, 1721–1732. doi:10.1287/mnsc.24.16.1721.

M. J. A. de Condorcet (1785), Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. Paris.

A. H. Copeland (1951), A Reasonable Social Welfare Function. *mimeo*, University of Michigan.

E. J. Emond and D. W. Mason (2000), *A new technique for high level decision support*. Technical Report ORD Project Report PR2000/13, Operational Research Division, Department of National Defence, Canada.

K. Hornik and D. Meyer (2007), Deriving consensus rankings from benchmarking experiments. In R. Decker and H.-J. Lenz, *Advances in Data Analysis*. Studies in Classification, Data Analysis, and Knowledge Organization. Springer-Verlag: Heidelberg, 163–170.

F. Marcotorchino and P. Michaud (1982). Agrégation de similarités en classification automatique. *Revue de Statistique Appliquée*, **30**/2, 21–44. https://eudml.org/doc/106132.

S. Régnier (1965), Sur quelques aspects mathématiques des problèmes de classification automatique. *ICC Bulletin*, **4**, 175–191.

Examples

```
## Consensus equivalence.
## (I.e., in fact, consensus partition.)
## Classification of 30 felines, see Marcotorchino and Michaud (1982).
data("Felines")
## Consider each variable an equivalence relation on the objects.
relations <- as.relation_ensemble(Felines)
## This gives a relation ensemble of length 14 (number of variables in
## the data set).
if(requireNamespace("Rglpk", quietly = TRUE)) {
    ## Now fit an equivalence relation to this:
    E <- relation_consensus(relations, "symdiff/E")
    ## And look at the equivalence classes:
    ids <- relation_class_ids(E)
    ## Or, more nicely:
    split(rownames(Felines), ids)
    ## Which is the same as in the paper ...
}

## Consensus linear order.
## Example from Cook and Kress, pages 48ff.
## Relation from paired comparisons.
pm <- matrix(c(0, 1, 0, 1, 1,
               0, 0, 0, 1, 1,
               1, 1, 0, 0, 0,
               0, 0, 1, 0, 0,
```

```
              0, 0, 1, 1, 0),
            nrow = 5,
            byrow = TRUE,
            dimnames = list(letters[1:5], letters[1:5]))
## Note that this is a Cook and Kress "preference matrix" where entry
## (i,j) is one iff object i is preferred to object j (i > j).
## Set up the corresponding '<' relation:
R <- as.relation(t(pm))
relation_incidence(R)
relation_is(R, "tournament")
if(requireNamespace("Rglpk", quietly = TRUE)) {
    ## Closest linear order:
    L <- relation_consensus(R, "symdiff/L")
    relation_incidence(L)
    ## Visualize provided that Rgraphviz is available.
    if(require("Rgraphviz")) plot(L)
    ## But note that this linear order is not unique.
    L <- relation_consensus(R, "symdiff/L", control = list(n = "all"))
    print(L)
    if(require("Rgraphviz")) plot(L)
    ## (Oh no: c is once first and once last.)
    ## Closest weak order relation with at most 3 indifference classes:
    W3 <- relation_consensus(R, "symdiff/W", control = list(k = 3))
    relation_incidence(W3)
}

## Consensus weak orders.
## Example from Emond and Mason, pages 28f.
## The reference provides 21 partial rankings of 15 objects,
## in 3 groups of 7 rankings (corresponding to three different
## ranking criteria) with respective weights 4, 5, and 7.
wei <- rep.int(c(4, 5, 7), rep(7, 3))
## The rankings are written by listing the object labels from the
## best to the worst, with a leading minus indicating a tie with
## the previous object:
EM_inputs <-
    c("6 1 -7 -9 10 3 8 11 5 -12 2 -4 -13",
      "6 10 9 3 4 -8 7 1 -5 -11 2 12 13 14 15",
      "6 10 3 7 8 11 5 14 15 12 1 -4 -13 2 -9",
      "6 9 -11 10 3 14 12 7 4 5 2 1 8 13 15",
      "10 6 7 1 11 -13 4 2 3 9 12 14 -15 8 5",
      "6 9 8 -10 11 4 1 5 7 15 2 12 14 13 3",
      "1 -6 -10 7 -12 9 3 4 -11 -14 -15 2 -13 8",
      "4 -10 1 -7 6 -9 -13 5 -14 3 12 8 11 -15 2",
      "4 -9 5 1 14 11 8 3 6 2 -13 10 12 7 15",
      "4 2 -5 8 15 7 11 -14 1 -12 -13 10 9 6",
      "2 -11 -12 -14 -15 6 -13 3 -4 9 8 -10 1 -5 -7",
      "4 14 10 2 5 3 1 13 12 7 15 8 11 6 9",
      "4 2 5 1 15 7 13 14 3 -12 8 11 6 9 10",
      "12 1 3 -4 2 11 -13 -15 9 14 6 8 7 -10 5",
      "5 4 9 2 -7 14 8 -11 3 1 15 12 6 10 13",
      "11 9 -14 15 12 3 4 13 8 6 7 10 5",
      "12 11 2 1 3 9 8 10 13 -14 6 4 -15 5 7",
```

```
      "4 -5 10 -12 3 8 -11 6 -7 -9 13 14 15",
      "12 5 -13 14 3 8 15 4 9 -10 11 6 7",
      "4 -5 -8 11 6 14 7 1 -2 -15 10 3 13 9 -12",
      "10 8 5 -11 6 -14 9 4 -13 -15 3 -12 2 1")
## Using the Emond-Mason paired comparison dissimilarity, there
## are three consensus rankings when using the above weights:
EM_solutions <-
    c("4 10 5-11 1 -2-14 3-12 9 8 6 7 13-15",
      "4 10 5-11 1 -2 9 14 3-12 8 6 7 13-15",
      "4 10 5-11 2-14 1 3-12 9 8 6 7 13-15")
## We can reproduce this as follows.
## We first provide a reader for the rankings, and a maker for
## creating the (possibly partial) ranking with the appropriate
## domain:
reader <- function(s) {
    strsplit(unlist(strsplit(gsub(" *-", "-", s),
                             " +")),
             "-",
             fixed = TRUE)
}
maker <- function(s) {
    ranking(lapply(reader(s), as.numeric),
            domain = as.numeric(1 : 15))
}
EM_inputs <- lapply(EM_inputs, maker)
EM_solutions <- lapply(EM_solutions, maker)
## Package 'relations' uses NA for non-diagonal incidences
## featuring unranked objects.
## Following the reference, we impute these by zeroes:
ens <- relation_impute(relation_ensemble(list = EM_inputs), "omit")
if(requireNamespace("Rglpk", quietly = TRUE)) {
    ## We can now obtain all consensus weak orders (corresponding to
    ## complete rankings) as follows:
    con <- relation_consensus(ens, "PC/W", wei, delta = "EM", all = TRUE)
    ## To verify that these agree with the solutions given in the
    ## reference:
    sets::set_outer(con, relation_ensemble(list = EM_solutions), `==`)
}
```

---

cover *Covering Relations*

Description

Compute the covering relation of an endorelation.

Usage

```
relation_cover(x)
```

Arguments

x	an endorelation.

Details

Let R be an endorelation with domain (X, X) and P be the asymmetric part of R for which xPy iff xRy and not yRx. (If R is a $\leq$ order relation, P is the associated strict order.) We say that x is covered by y if xPy and there is no z such that xPz and zPy. One also says that y covers x, or that it is a successor of x.

The covering relation of R consists of all pairs (x, y) for which x is covered by y.

---

dissimilarity *Dissimilarity Between Relations*

Description

Compute the dissimilarity between (ensembles of) relations.

Usage

```
relation_dissimilarity(x, y = NULL, method = "symdiff", ...)
```

Arguments

x	an ensemble of relations (see `relation_ensemble()`), or something which can be coerced to such.
y	`NULL` (default), or as for `x`.
method	a character string specifying one of the built-in methods for computing dissimilarity, or a function to be taken as a user-defined method. If a character string, its lower-cased version is matched against the lower-cased names of the available built-in methods using `pmatch()`. See **Details** for available built-in methods.
...	further arguments to be passed to methods.

Details

Available built-in methods are as follows.

`"symdiff"` symmetric difference distance. This computes the cardinality of the symmetric difference of two relations, i.e., the number of tuples contained in exactly one of two relations. For preference relations, this coincides with the *Kemeny-Snell* metric (Kemeny and Snell, 1962). For linear orders, it gives Kendall's τ metric (Diaconis, 1988).

Can also be referred to as `"SD"`.

Only applicable to crisp relations.

`"manhattan"` the Manhattan distance between the incidences.

`"euclidean"` the Euclidean distance between the incidences.

`"CS"` Cook-Seiford distance, a generalization of the distance function of Cook and Seiford (1978). Let the generalized ranks of an object a in the (first) domain of an endorelation R be defined as the number of objects b dominating a (i.e., for which aRb and not bRa), plus half the number of objects b equivalent to a (i.e., for which aRb and bRa). For preference relations, this gives the usual Kendall ranks arranged according to decreasing preference (and averaged for ties). Then the generalized Cook-Seiford distance is defined as the l_1 distance between the generalized ranks. For linear orders, this gives Spearman's footrule metric (Diaconis, 1988).

Only applicable to crisp endorelations.

`"CKS"` Cook-Kress-Seiford distance, a generalization of the distance function of Cook, Kress and Seiford (1986). For each pair of objects a and b in an endorelation R, we can have aRb and not bRa or vice versa (cases of "strict preference"), aRb and bRa (the case of "indifference"), or neither aRb nor bRa (the case of "incomparability"). (Only the last two are possible if $a = b$.) The distance by Cook, Kress and Seiford puts indifference as the metric centroid between both preference cases and incomparability (i.e., indifference is at distance one from the other three, and each of the other three is at distance two from the others). The generalized Cook-Kress-Seiford distance is the paired comparison distance (i.e., a metric) based on these distances between the four paired comparison cases. (Formula 3 in the reference must be slightly modified for the generalization from partial rankings to arbitrary endorelations.)

Only applicable to crisp endorelations.

`"score"` score-based distance. This computes $\Delta(s(x), s(y))$ for suitable score and distance functions s and Δ, respectively. These can be specified by additional arguments `score` and `Delta`. If `score` is a character string, it is taken as the method for `relation_scores`. Otherwise, if given it must be a function giving the score function itself. If `Delta` is a number $p \geq 1$, the usual l_p distance is used. Otherwise, it must be a function giving the distance function. The defaults correspond to using the default relation scores and $p = 1$, which for linear orders gives Spearman's footrule distance.

Only applicable to endorelations.

`"Jaccard"` Jaccard distance: 1 minus the ratio of the cardinalities of the intersection and the union of the relations.

`"PC"` (generalized) paired comparison distance. This generalizes the symdiff and CKS distances to use a general set of discrepancies δ_{kl} between the possible paired comparison results with $a, b/b, a$ incidences 0/0, 1/0, 0/1, and 1/1 numbered from 1 to 4 (in a preference context with a $\leq$ encoding, these correspond to incompatibility, strict $<$ and $>$ preference, and indifference), with δ_{kl} the discrepancy between possible results k and l. The distance is then obtained as the sum of the discrepancies from the paired comparisons of distinct objects, plus half the sum of discrepancies from the comparisons of identical objects (for which the only possible results are incomparability and indifference). The distance is a metric provided that the δ_{kl} satisfy the metric conditions (non-negativity and zero iff $k = l$, symmetry and sub-additivity).

The discrepancies can be specified via the additional argument `delta`, either as a numeric vector of length 6 with the non-redundant values $\delta_{21}, \delta_{31}, \delta_{41}, \delta_{32}, \delta_{42}, \delta_{43}$, or as a character string partially matching one of the following built-in discrepancies with corresponding parameter vector δ:

`"symdiff"` symmetric difference distance, with discrepancy between distinct results two between either opposite strict preferences or indifference and incomparability, and one otherwise: $\delta = (1, 1, 2, 2, 1, 1)$ (default).
Can also be referred to as `"SD"`.

`"CKS"` Cook-Kress-Seiford distance, see above: $\delta = (2, 2, 1, 2, 1, 1)$.

`"EM"` the distance obtained from the generalization of the Kemeny-Snell distance for complete rankings to partial rankings introduced in Emond and Mason (2000). This uses a discrepancy of two for opposite strict preferences, and one for all other distinct results: $\delta = (1, 1, 1, 2, 1, 1)$.

`"JMB"` the distance with parameters as suggested by Jabeur, Martel and Ben Khélifa (2004): $\delta = (4/3, 4/3, 4/3, 5/3, 1, 1)$.

`"discrete"` the discrete metric on the set of paired comparison results: $\delta = (1, 1, 1, 1, 1, 1)$.

Only applicable to crisp endorelations.

Methods `"symdiff"`, `"manhattan"`, `"euclidean"` and `"Jaccard"` take an additional logical argument `na.rm`: if true (default: false), tuples with missing memberships are excluded in the dissimilarity computations.

Value

If `y` is `NULL`, an object of class `dist` containing the dissimilarities between all pairs of elements of `x`. Otherwise, a matrix with the dissimilarities between the elements of `x` and the elements of `y`.

References

W. D. Cook, M. Kress and L. M. Seiford (1986), Information and preference in partial orders: a bimatrix representation. *Psychometrika* **51**/2, 197–207. doi:10.1007/BF02293980.

W. D. Cook and L. M. Seiford (1978), Priority ranking and consensus formation. *Management Science*, **24**/16, 1721–1732. doi:10.1287/mnsc.24.16.1721.

P. Diaconis (1988), *Group Representations in Probability and Statistics*. Institute of Mathematical Statistics: Hayward, CA.

E. J. Emond and D. W. Mason (2000), *A new technique for high level decision support*. Technical Report ORD Project Report PR2000/13, Operational Research Division, Department of National Defence, Canada.

K. Jabeur, J.-M. Martel and S. Ben Khélifa (2004). A distance-based collective preorder integrating the relative importance of the groups members. *Group Decision and Negotiation*, **13**, 327–349. doi:10.1023/B:GRUP.0000042894.00775.75.

J. G. Kemeny and J. L. Snell (1962), *Mathematical Models in the Social Sciences*, chapter "Preference Rankings: An Axiomatic Approach". MIT Press: Cambridge.

`domain` *Relation Domain, Arity, and Size*

Description

Determine the domain, domain names, arity, or size of a relation or a relation ensemble.

Usage

```
relation_arity(x)
relation_domain(x)
relation_domain_names(x)
relation_size(x)
```

Arguments

x	an R object inheriting from class relation or relation_ensemble.

Value

For determining the domain, an object of class relation_domain, inheriting from tuple.

See Also

tuple(); relation(); relation_domain<-() and relation_domain_names<-() for modifying the domain and domain names of a relation, respectively.

Examples

```
## A simple relation:
R <- as.relation(c(A = 1, B = 2, C = 3))
relation_incidence(R)
relation_arity(R)
relation_domain(R)
relation_domain_names(R)
relation_size(R)
```

---

elements *Elements of Relation Domains*

Description

Obtain elements of endorelation domains which have certain properties.

Usage

```
relation_elements(x, which, ...)
```

Arguments

x	an endorelation.
which	a character string specifying the property to be tested for. Currently, one of "minimal", "first", "last", or "maximal", or a unique abbreviation thereof.
...	additional arguments to be employed in the property tests.

Details

Let R be an endorelation with domain (X, X) and consider elements x and y of X. We say that x is

minimal: there is no $y \neq x$ with yRx.

a first element: xRy for all $y \neq x$.

a last element: yRx for all $y \neq x$.

maximal: there is no $y \neq x$ with xRy.

When computing the tests for the above properties, an additional `na.rm` argument can be given to control the handling of missing incidences. By default, these are treated as false, to the effect that they invalidate "for all" tests (corresponding to `na.rm = FALSE`) and pass the "there is no" tests (corresponding to `na.rm = TRUE`).

Value

A set with the elements having the specified property.

`ensemble` *Relation Ensembles*

Description

Creation and manipulation of relation ensembles.

Usage

```
relation_ensemble(..., list = NULL)
as.relation_ensemble(x)
is.relation_ensemble(x)
```

Arguments

`...`	R objects representing relations, or coercible to such.
`list`	a list of R objects as in `...`.
`x`	for coercion with `as.relation_ensemble()`, an R object as in `...`; for testing with `is.relation_ensemble()`, an arbitrary R object.

Details

`relation_ensemble()` creates non-empty "relation ensembles", i.e., collections of relations $R_i = (D, G_i)$ with the same domain D and possibly different graphs G_i.

Such ensembles are implemented as suitably classed lists of relation objects, making it possible to use `lapply()` for computations on the individual relations in the ensemble. Available methods for relation ensembles include those for subscripting, `c()`, `t()`, `rep()`, and `print()`.

Examples

```
data("Cetacea")
## Consider each variable an equivalence relation on the objects.
## Note that 2 variables (LACHRYMAL_AND_JUGAL_BONES and HEAD_BONES) have
## missing values, and hence are excluded.
ind <- sapply(Cetacea, function(s) all(!is.na(s)))
relations <- as.relation_ensemble(Cetacea[, ind])
## This gives a relation ensemble of length 14 (number of complete
## variables in the data set).
print(relations)
## Are there any duplicated relations?
any(duplicated(relations))
## Replicate and combine ...
thrice <- c(rep(relations, 2), relations)
## Extract unique elements again:
all.equal(unique(thrice), relations)
## Note that unique() does not preserve attributes, and hence names.
## In case we want otherwise:
all.equal(thrice[!duplicated(thrice)], relations)
## Subscripting:
relation_dissimilarity(relations[1 : 2], relations["CLASS"])
## Which relation is "closest" to the classification?
d <- relation_dissimilarity(relations)
sort(as.matrix(d)[, "CLASS"])[-1]
```

Felines *Felines Data*

Description

A data set with 14 variables on 30 different types of felines.

Usage

```
data("Felines")
```

Format

A data frame with 30 observations on 14 categorical variables (the first 10 morphological, the last 4 behavioral), with names in French and numeric levels as in the reference. Names, descriptions in French and English and the numbers of levels are as follows.

`TYPPEL`**:** Aspect du pelage; coat; 4.

`LONGPOIL`**:** Fourrure; fur; 2.

`OREILLES`**:** Oreilles; ears; 2.

`TAILLE`**:** Taille au garrot; waist; 3.

`POIDS`**:** Poids; weight; 3.

`LONGUEUR`**:** Longueur du corps; body length; 3.

QUEUE: Longueur de la queue; tail length; 3.

DENTS: Canines développées; carnassials; 2.

LARYNX: Os hyaoide; larynx; 2.

RETRACT: Griffes rétractiles; retractible claws; 2.

COMPORT: Comportement prédateur; predatory behavior; 3.

TYPPROIE: Type de la proie; type of prey; 3.

ARBRES: Monte ou non aux arbres; climbs trees or not; 2.

CHASSE: Chasse (courre ou affut); chases via chivy or ambush; 2.

Source

F. Marcotorchino and P. Michaud (1982), Agregation de similarités en classification automatique. *Revue de Statistique Appliquée*, **30**(2), 21–44.

Examples

```
data("Felines")
summary(Felines)
```

---

graph *Relation Graph*

Description

Determine the graph of a relation.

Usage

```
relation_graph(x)
```

Arguments

x	an R object inheriting from class relation.

Value

An object of class relation_graph, inheriting from set.

See Also

set(); relation(); relation_graph<-() for modifying the graph.

Examples

```
## A simple relation:
R <- as.relation(c(A = 1, B = 2, C = 3))
relation_graph(R)
```

impute *Impute relations*

Description

Impute missing incidences in relations by averaging all possible completions within a specified family.

Usage

```
relation_impute(x, method = NULL, control = list(), ...)
```

Arguments

x	an endorelation or an ensemble of endorelations.
method	character string specifying the method to be used (see **Details**). If NULL, it is guessed from the relation with missing *objects* removed.
control	named list of control arguments. Currently, only n is accepted by the any/F methods, indicating the number of solutions to be returned. Default is 1; "all" or NA will generate all possible completions. Note that n is currently ignored if x is a relation ensemble.
...	named list of control arguments, overriding the ones in control.

Details

Endorelations with missing elements (i.e., whose incidence is NA) are imputed using one of the methods described as follows.

"omit" Missing incidences are replaced by zeros, i.e., the corresponding elements are removed from the graph.

"any/F" The incidences are replaced by arbitrary values suitable for family F, with possible values:

- G General (unrestricted) relations.
- L Linear orders.
- W Weak orders.
- O Partial orders.

L, W, and O can optionally be complemented by /first or /last to further restrict imputed elements to be placed on top or bottom of the given ordering.

"average/F" Computes the relation with average incidences, based on all possible completions as indicated for the any/F methods. Note that these completions are not explicitly generated to compute the averages, and that the resulting relation will typically be fuzzy.

Value

If x is an ensemble or more than one solution is requested using the n control argument: an ensemble of endorelations. An endorelation otherwise.

Examples

```
## create a relation with a missing object
R <- ranking(1:2, 1:3)
print(R)
R <- as.relation(R)

## find all suitable completions within L
ens <- relation_impute(R, method = "any/L", n = "all")
lapply(ens, as.ranking)
if(require("Rgraphviz")) plot(ens)

## find 3 suitable partial orders
ens <- relation_impute(R, method = "any/O", n = 3)
lapply(ens, relation_incidence)
if(require("Rgraphviz")) plot(ens)

## compute average completion
R1 <- relation_impute(R, method = "average/O")
relation_incidence(R1)

## check correctness of averaging
R2 <- mean(relation_impute(R, "any/O", n = "all"))
stopifnot(all.equal(R1, R2))
```

`incidence` *Relation Incidences*

Description

Determine the incidences of a relation.

Usage

```
relation_incidence(x, ...)
```

Arguments

`x`	an object inheriting from class `relation`.
`...`	Further arguments passed to the labeling function used for creating the dimnames of the incidence matrix.

Value

For a k-ary relation, a k-dimensional numeric array with values in the unit interval inheriting from class `relation_incidence` whose elements give the memberships of the corresponding k-tuples are contained in the relation (for a crisp relation, a binary (0/1) array with elements indicating whether the corresponding tuples are contained in the relation or not).

See Also

relation(); relation_incidence<-() for modifying the incidences.

Examples

```
R <- as.relation(c(A = 1, B = 2, C = 3))
relation_incidence(R)
```

---

pclust *Prototype-Based Partitions of Relations*

Description

Compute prototype-based partitions of a relation ensemble by minimizing $\sum w_b u_{bj}^m d(x_b, p_j)^e$, the sum of the case-weighted and membership-weighted e-th powers of the dissimilarities between the elements x_b of the ensemble and the prototypes p_j, for suitable dissimilarities d and exponents e.

Usage

```
relation_pclust(x, k, method, m = 1, weights = 1,
                control = list())
```

Arguments

x	an ensemble of relations (see relation_ensemble()), or something which can be coerced to this.
k	an integer giving the number of classes to be used in the partition.
method	the consensus method to be employed, see relation_consensus().
m	a number not less than 1 controlling the softness of the partition (as the "fuzzification parameter" of the fuzzy c-means algorithm). The default value of 1 corresponds to hard partitions obtained from a generalized k-means problem; values greater than one give partitions of increasing softness obtained from a generalized fuzzy c-means problem.
weights	a numeric vector of non-negative case weights. Recycled to the number of elements in the ensemble given by x if necessary.
control	a list of control parameters. See **Details**.

Details

For $m = 1$, a generalization of the Lloyd-Forgy variant of the k-means algorithm is used, which iterates between reclassifying objects to their closest prototypes, and computing new prototypes as consensus relations (generalized "central relations", Régnier (1965)) for the classes. This procedure was proposed in Gaul and Schader (1988) as the "Clusterwise Aggregation of Relations" (CAR).

For $m > 1$, a generalization of the fuzzy c-means recipe is used, which alternates between computing optimal memberships for fixed prototypes, and computing new prototypes as the consensus relations for the classes.

This procedure is repeated until convergence occurs, or the maximal number of iterations is reached.

Consensus relations are computed using `relation_consensus()`.

Available control parameters are as follows.

`maxiter` an integer giving the maximal number of iterations to be performed. Defaults to 100.

`reltol` the relative convergence tolerance. Defaults to `sqrt(.Machine$double.eps)`.

`control` control parameters to be used in `relation_consensus()`.

The dissimilarities d and exponent e are implied by the consensus method employed, and inferred via a registration mechanism currently only made available to built-in consensus methods. For the time being, all optimization-based consensus methods use the symmetric difference dissimilarity (see `relation_dissimilarity()`) for d and $e = 1$.

The fixed point approach employed is a heuristic which cannot be guaranteed to find the global minimum. Standard practice would recommend to use the best solution found in "sufficiently many" replications of the base algorithm.

Value

An object of class `cl_partition`.

References

S. Régnier (1965). Sur quelques aspects mathématiques des problèmes de classification automatique. *ICC Bulletin*, **4**, 175–191.

W. Gaul and M. Schader (1988). Clusterwise aggregation of relations. *Applied Stochastic Models and Data Analysis*, **4**, 273–282. doi:10.1002/asm.3150040406.

`plot` *Visualize Relations*

Description

Visualize certain crisp endorelations by plotting a Hasse Diagram of their transitive reduction.

Usage

```
## S3 method for class 'relation'
plot(x,
     attrs = list(graph = list(rankdir = "BT"),
                  edge = list(arrowsize = NULL),
                  node = list(shape = "rectangle",
                              fixedsize = FALSE)),
     limit = 6L,
     labels = NULL,
     main = NULL,
     type = c("simplified", "raw"),
```

```
     ...)

## S3 method for class 'relation_ensemble'
plot(x,
     attrs = list(list(graph = list(rankdir = "BT"),
                       edge = list(arrowsize = NULL),
                       node = list(shape = "rectangle",
                                   fixedsize = FALSE))),
     type = "simplified",
     limit = 6L,
     labels = NULL,
     ...,
     layout = NULL, main = NULL)
```

Arguments

x	an R object inheriting from class relation or relation_ensemble.
attrs	argument passed to the plot method for class graphNEL. Note that for the relation_ensemble method, it is a *list* of such objects, recycled as needed.
type	character vector of either "simplified" or "raw" strings, one for each relation plotted. (See details.)
limit	Argument passed to the labeling function creating default labels for the nodes (see LABELS()). Recycled as needed for relation ensembles.
labels	Optional list of character vectors defining unique labels for the nodes. List of such lists for relation ensembles.
layout	integer vector of length 2 specifying the number of rows and columns of the screen layout. If NULL, the layout is square.
...	Other arguments passed to the graphNEL plot method.
main	character vector used for the main title(s). If NULL, the title(s) is (are) set to the type of the visualized relation(s).

Details

Visualization requires that package **Rgraphviz** is available. If type is "simplified" (default), the transitive reduction is first computed to reduce visual complexity (especially for transitive relations). For partial orders, a Hasse diagram is plotted. In case of (acyclic) transitive complete relations (i.e., weak orders, preferences), the dual is plotted. For all other acyclic relations, the asymmetric part is plotted. (Note that the default settings in these cases create a diagram with nodes ordered bottom-up and with no arrows.) For cyclic relations, a raw graph (with arrows) of the corresponding transitive reduction is computed. If type is "raw", a directed graph without any transformation is plotted from the relation.

See Also

relation(), transitive_reduction()

Examples

```
require("sets") # set() etc.
if(require("Rgraphviz")) {
  ## simple example
  plot(as.relation(1 : 5))

  ## inclusion on a power set:
  ps <- 2 ^ set("a", "b", "c")
  inc <- set_outer(ps, set_is_subset)
  R <- relation(incidence = inc)
  plot(relation_ensemble(R, R), type = c("simplified", "raw"))
}
```

---

predicates *Relation Predicates*

Description

Predicate functions for testing for binary relations and endorelations, and special kinds thereof.

Usage

```
relation_is(x, predicate, ...)
relation_is_Euclidean(x, na.rm = FALSE)
relation_is_Ferrers(x, na.rm = FALSE)
relation_is_acyclic(x)
relation_is_antisymmetric(x, na.rm = FALSE)
relation_is_asymmetric(x, na.rm = FALSE)
relation_is_bijective(x)
relation_is_binary(x)
relation_is_complete(x, na.rm = FALSE)
relation_is_coreflexive(x, na.rm = FALSE)
relation_is_crisp(x, na.rm = FALSE)
relation_is_cyclic(x)
relation_is_endorelation(x)
relation_is_equivalence(x, na.rm = FALSE)
relation_is_functional(x)
relation_is_homogeneous(x)
relation_is_injective(x)
relation_is_interval_order(x, na.rm = FALSE)
relation_is_irreflexive(x, na.rm = FALSE)
relation_is_left_total(x)
relation_is_linear_order(x, na.rm = FALSE)
relation_is_match(x, na.rm = FALSE)
relation_is_negatively_transitive(x, na.rm = FALSE)
relation_is_partial_order(x, na.rm = FALSE)
relation_is_preference(x, na.rm = FALSE)
```

```
relation_is_preorder(x, na.rm = FALSE)
relation_is_quasiorder(x, na.rm = FALSE)
relation_is_quasitransitive(x, na.rm = FALSE)
relation_is_quaternary(x)
relation_is_reflexive(x, na.rm = FALSE)
relation_is_right_total(x)
relation_is_semiorder(x, na.rm = FALSE)
relation_is_semitransitive(x, na.rm = FALSE)
relation_is_strict_linear_order(x, na.rm = FALSE)
relation_is_strict_partial_order(x, na.rm = FALSE)
relation_is_strongly_complete(x, na.rm = FALSE)
relation_is_surjective(x)
relation_is_symmetric(x, na.rm = FALSE)
relation_is_ternary(x)
relation_is_tournament(x, na.rm = FALSE)
relation_is_transitive(x, na.rm = FALSE)
relation_is_trichotomous(x, na.rm = FALSE)
relation_is_weak_order(x, na.rm = FALSE)
relation_has_missings(x)
```

Arguments

x	an object inheriting from class relation.
na.rm	a logical indicating whether tuples with missing memberships are excluded in the predicate computations.
predicate	character vector matching one of the following (see **details**): "binary", "ternary", "quaternary", "left_total", "right_total", "surjective", "functional", "injective", "bijective", "endorelation", "homogeneous", "crisp", "complete", "match", "strongly_complete", "reflexive", "irreflexive", "coreflexive", "symmetric", "asymmetric", "antisymmetric", "transitive", "negatively_transitive", "quasitransitive", "Ferrers", "semitransitive", "trichotomous", "Euclidean", "equivalence", "weak_order", "preference", "preorder", "quasiorder", "partial_order", "linear_order", "strict_partial_order", "strict_linear_order", "tournament", "interval_order", "semiorder", "acyclic" "cyclic"
...	Additional arguments passed to the predicate functions (currently, only na.rm for most predicates).

Details

This help page documents the predicates currently available. Note that the preferred way is to use the meta-predicate function relation_is(x, "FOO") instead of the individual predicates relation_is_FOO(x) since the latter will become deprecated in future releases.

A binary relation is a relation with arity 2.

A relation R on a set X is called *homogeneous* iff $D(R) = (X, \ldots, X)$.

An *endorelation* is a binary homogeneous relation.

For a crisp binary relation, let us write xRy iff (x, y) is contained in R.

A crisp binary relation R is called

left-total: for all x there is at least one y such that xRy.

right-total: for all y there is at least one x such that xRy.

functional: for all x there is at most one y such that xRy.

surjective: the same as right-total.

injective: for all y there is at most one x such that xRy.

bijective: left-total, right-total, functional and injective.

A crisp endorelation R is called

reflexive: xRx for all x.

irreflexive: there is no x such that xRx.

coreflexive: xRy implies $x = y$.

symmetric: xRy implies yRx.

asymmetric: xRy implies that not yRx.

antisymmetric: xRy and yRx imply that $x = y$.

transitive: xRy and yRz imply that xRz.

complete: for all distinct x and y, xRy or yRx.

strongly complete: for all x and y, xRy or yRx (i.e., complete and reflexive).

negatively transitive: not xRy and not yRz imply that not xRz.

Ferrers: xRy and zRw imply xRw or yRz.

semitransitive: xRy and yRz imply xRw or wRz.

quasitransitive: xRy and not yRx and yRz and not zRy imply xRz and not zRx (i.e., the asymmetric part of R is transitive).

trichotomous: exactly one of xRy, yRx, or $x = y$ holds.

Euclidean: xRy and xRz imply yRz.

acyclic: the transitive closure of R is antisymmetric.

cyclic: R is not acyclic.

Some combinations of these basic properties have special names because of their widespread use:

preorder: reflexive and transitive.

quasiorder: the same as preorder.

equivalence: a symmetric preorder (reflexive, symmetric, and transitive).

weak order: a complete preorder (complete, reflexive, and transitive).

preference: the same as weak order.

partial order: an antisymmetric preorder (reflexive, antisymmetric, and transitive).

strict partial order: irreflexive, antisymmetric, and transitive, or equivalently: asymmetric and transitive).

linear order: a complete partial order.

strict linear order: a complete strict partial order.

match: strongly complete.

tournament: complete and asymmetric.

interval order: complete and Ferrers.

semiorder: a semitransitive interval order.

If R is a weak order ("(weak) preference relation"), $I = I(R)$ defined by xIy iff xRy and yRx is an equivalence, the *indifference relation* corresponding to R.

There seem to be no commonly agreed definitions for order relations: e.g., Fishburn (1972) requires these to be irreflexive.

For a fuzzy binary relation R, let $R(x, y)$ denote the membership of (x, y) in the relation. Write T and S for the fuzzy t-norm (intersection) and t-conorm (disjunction), respectively (min and max for the "standard" Zadeh family). Then generalizations of the above basic endorelation predicates are as follows.

reflexive: $R(x, x) = 1$ for all x.

irreflexive: $R(x, x) = 0$ for all x.

coreflexive: $R(x, y) > 0$ implies $x = y$.

symmetric: $R(x, y) = R(y, x)$ for all $x \neq y$.

asymmetric: $T(R(x, y), R(y, x)) = 0$ for all x, y.

antisymmetric: $T(R(x, y), R(y, x)) = 0$ for all $x \neq y$.

transitive: $T(R(x, y), R(y, z)) \leq R(x, z)$ for all x, y, z.

complete: $S(R(x, y), R(y, x)) = 1$ for all $x \neq y$.

strongly complete: $S(R(x, y), R(y, x)) = 1$ for all x, y.

negatively transitive: $R(x, z) \leq S(R(x, y), R(y, z))$ for all x, y, z.

Ferrers: $T(R(x, y), R(z, w)) \leq S(R(x, w), R(z, y))$ for all x, y, z, w.

semitransitive: $T(R(x, w), R(w, y)) \leq S(R(x, z), R(z, y))$ for all x, y, z, w.

The combined predicates are obtained by combining the basic predicates as for crisp endorelations (see above).

A relation has missings iff at least one cell in the incidence matrix is `NA`. In addition to `relation_has_missings()`, an `is.na` method for relations is available, returning a matrix of logicals corresponding to the incidences tested for missingness.

References

P. C. Fishburn (1972), *Mathematics of decision theory*. Methods and Models in the Social Sciences 3. Mouton: The Hague.

H. R. Varian (2002), *Intermediate Microeconomics: A Modern Approach*. 6th Edition. W. W. Norton & Company.

Examples

```
require("sets")
R <- relation(domain = c(1, 2, 3), graph = set(c(1, 2), c(2, 3)))
summary(R)

## Note the possible effects of NA-handling:
relation_incidence(R)
relation_is(R, "transitive") ## clearly FALSE

relation_incidence(R)[1, 2] <- NA
relation_incidence(R)
relation_is(R, "transitive") ## clearly NA

## The following gives TRUE, since NA gets replaced with 0:
relation_is(R, "transitive", na.rm = TRUE)
```

`properties` *Relation Properties*

Description

Retrieve relation properties.

Usage

```
relation_properties(x)
relation_property(x, which)
```

Arguments

`x`	A relation.
`which`	Property name (character string).

Details

These functions are used for the *extrinsic* properties of relations. Others can be retrieved using the predicate functions.

See Also

`relation()` and `relation_is()` for all predicate functions defined on relations.

Examples

```
x <- as.relation(1 : 3)
relation_properties(x)
relation_property(x, "is_endorelation")
```

ranking *Rankings*

Description

Creates a ranking object.

Usage

```
ranking(x, domain = NULL, decreasing = TRUE, complete = FALSE)
as.ranking(x, ...)
is.ranking(x)
```

Arguments

x	For ranking(): either an atomic vector interpreted as labels of the ranked objects, or as their scores in case of a named numeric vector (with the names giving the labels), or a list of atomic vectors representing equivalence classes of labels. For as.ranking(): an R object coercible to a ranking object (including relation objects).
domain	object coercible to a set, from which the labels usable in x are derived. If NULL, it is created from x.
decreasing	logical indicating whether the ranking orders objects from the best to the worst (TRUE), or the other way round.
complete	logical specifying whether missing values should be imputed, if any. Missing elements are those from domain not used in x. If decreasing is TRUE (FALSE), all missings are ranked tied behind (ahead) the worst (best) ranked object.
...	currently not used.

Value

An object of class ranking.

See Also

relation()

Examples

```
## simple rankings
OBJECTS <- c("Apples", "Bananas", "Oranges", "Lemons")
print(R <- ranking(OBJECTS))
ranking(OBJECTS[2:4], domain = OBJECTS)
ranking(OBJECTS[2:4], domain = OBJECTS, complete = TRUE)

## ranking with ties (weak orders)
ranking(list(c("PhD", "MD"), "MSc", c("BSc", "BA")))
```

```
## ranking A > B ~ C with D missing:
ranking(c(A = 1, B = 2, C = 2, D = NA))

## coercion functions
identical(as.ranking(as.relation(R)), R)
```

`reduction` *Transitive and Reflexive Reduction*

Description

Computes transitive and reflexive reduction of an endorelation.

Usage

```
transitive_reduction(x)
reflexive_reduction(x)
## S3 method for class 'relation'
reduction(x, operation = c("transitive", "reflexive"), ...)
```

Arguments

`x`	an R object inheriting from class `relation`, representing an endorelation.
`operation`	character string indicating the kind of reduction.
`...`	currently not used.

Details

Let R be an endorelation on X and n be the number of elements in X.

The *transitive reduction* of R is the smallest relation R' on X so that the transitive closure of R' is the same than the transitive closure of R.

The transitive reduction of an *acyclic* relation can be obtained by subtracting from R the composition of R with its transitive closure.

The transitive reduction of a *cyclic* relation is the transitive reduction of the condensation, combined with the component representation of R. (Note that the transitive reduction of a cyclic relation is cyclic.)

The *reflexive reduction* of R is computed by setting the diagonal of the incidence matrix to 0.

References

S. Warshall (1962), A theorem on Boolean matrices. *Journal of the ACM*, **9**/1, 11–12. doi:10.1145/321105.321107.

J. A. La Poutré and J. van Leeuwen (1988), Maintenance of Transitive Closures and Transitive Reductions of Graphs. *Proceedings of the International Workshop of Graph-Theoretic Concepts in Computer Science*, Springer, London, 106–120.

See Also

relation(),
reflexive_reduction(), transitive_reduction(), reduction(),
relation_condensation(), relation_component_representation().

Examples

```
R <- as.relation(1 : 5)
relation_incidence(R)

## transitive closure/reduction
RR <- transitive_reduction(R)
relation_incidence(RR)
R == transitive_closure(RR)

## same
require("sets") # closure() and reduction() etc.
R == closure(reduction(R))

## reflexive closure/reduction

RR <- reflexive_reduction(R)
relation_incidence(RR)
R == reflexive_closure(RR)

## same:
R == closure(reduction(R, "reflexive"), "reflexive")

## transitive reduction of a cyclic relation:
## (example from La Poutre and van Leeuwen)

require("sets") # set(), pair() etc.
if(require("Rgraphviz")) {
  G <- set(pair(1L, 2L), pair(2L, 1L), pair(1L, 3L), pair(3L, 1L),
           pair(3L, 7L), pair(2L, 5L), pair(2L, 6L), pair(6L, 5L),
           pair(5L, 7L), pair(4L, 6L), pair(5L, 4L), pair(4L, 7L))
  R <- endorelation(graph = G)
  plot(relation_ensemble(R, R), type = c("raw", "simplified"), main =
           c("original graph", "transitive reduction"))
  }
```

`relation` *Relations*

Description

Creation and manipulation of relations.

Usage

```
relation(domain = NULL, incidence = NULL, graph = NULL,
         charfun = NULL)
endorelation(domain = NULL, incidence = NULL, graph = NULL,
             charfun = NULL)
homorelation(domain = NULL, incidence = NULL, graph = NULL,
             charfun = NULL)
as.relation(x, ...)
is.relation(x)
```

Arguments

`domain`	List (or tuple) of (possibly named) sets (or vectors) used as the domain, recycled as needed to fit the arity of the relation. If `domain` is not a list or tuple, it is converted to a list.
`incidence`	A numeric array with values in the unit interval, or a logical array. Note that one-dimensional incidences are also accepted. The `names`/`dimnames` attribute of `incidence` is used as domain if this is not explicitly given using the `domain` argument.
`graph`	Either a set of equally sized tuples, or a list of (possibly, generic) vectors of same length where each component specifies one relation element, or a data frame where each row specifies one relation element. For the latter, the columns correspond to the domain sets, and the colnames are used as their labels if specified.
`charfun`	A characteristic function of the relation, i.e., a predicate function taking k arguments, with k equal to the arity of the relation.
`x`	an R object.
`...`	Further arguments passed to `as.relation()` methods (currently not used for those defined in the relations package).

Details

Given k sets of objects X_1, ..., X_k, a k-ary relation R on $D(R) = (X_1, \ldots, X_k)$ is a (possibly fuzzy) subset $G(R)$ of the Cartesian product $X_1 \times \cdots \times X_k$. We refer to $D(R)$ and $G(R)$ as the *domain* and the *graph* of the relation, respectively (alternative notions are that of *ground* and *figure*, respectively). We also refer to $s = (s_1, \ldots, s_k)$, where each s_i gives the cardinality of X_i, as the *size* of the relation.

Strictly speaking, the relation is the *pair* $(D(R), G(R))$; often, relations are identified with their graph. If $G(R)$ is a crisp subset of $D(R)$, R is a *crisp relation*. In this case, we say that a k-tuple t is *contained* in the relation R iff it is an element of $G(R)$.

The *characteristic function* f_R of a relation R is the membership function of $G(R)$, giving for each k-tuple t in $D(R)$ the membership (amount of belongingness) of t to $G(R)$. In the crisp case, f_R is also referred to as the indicator function of the relation, and is a binary (0/1) function such that $f_R(t)$ is one iff t is in $G(R)$.

Relations with arity 2, 3, and 4 are typically referred to as *binary*, *ternary*, and *quaternary* relations, respectively. A *homorelation* on X is a relation with homogeneous domain, i.e. $(X, X, \ldots, X)$.

An *endorelation* on X (or binary relation *over* X) is a binary homorelation. See predicates for the most important types of endorelations.

Relations with the same domain can naturally be ordered according to their graphs. I.e., $R \leq S$ iff $G(R)$ is a subset of $G(S)$, or equivalently, iff $f_R(t) \leq f_S(t)$ for every k-tuple t (in the crisp case, iff every tuple contained in R is also contained in S). This induces a lattice structure, with meet (greatest lower bound) and join (least upper bound) the intersection and union of the graphs, respectively, also known as the *intersection* and *union* of the relations. The least element moves metric on this lattice is the *symmetric difference metric*, i.e., the Manhattan distance between the collections of membership values (incidences). In the crisp case, this gives the cardinality of the symmetric difference of the graphs (the number of tuples in exactly one of the relation graphs).

The *complement* (or *negation*) R^c of a relation R is the relation with domain $D(R)$ whose graph is the complement of $G(R)$ (in the crisp case, containing exactly the tuples not contained in R).

For binary crisp relations R, it is customary to write xRy iff (x, y) is contained in R. For binary crisp relations R_1 and R_2 with domains (X, Y) and (Y, Z), the *composition* $T = R_1 * R_2$ of R_1 and R_2 is defined by taking xSz iff there is an y such that xR_1y and yR_2z. The *transpose* (or *inverse*) R^t of the relation R with domain (X, Y) is the relation with domain (Y, X) such that yR^tx iff xRy. The *codual* (Clark (1990), also known as the 'dual' in the fuzzy preference literature, e.g., Ovchinnikov (1991)) is the complement of the transpose (equivalently, the transpose of the complement).

For binary fuzzy relations R, one often writes $R(x, y)$ for the membership of the pair (x, y) in the relation. The above notions need to take the fuzzy logic employed (as described by the fuzzy t-norm (intersection) T, t-conorm (disjunction) S, and negation N) into account. Let R, R_1 and R_2 be binary relations with appropriate domains. Then the memberships for (x, y) of the complement, transpose and codual of R are $N(R(x, y))$, $R(y, x)$ and $N(R(y, x))$, respectively. The membership of (x, y) for the composition of R_1 and R_2 is $\max_z T(R_1(x, z), R_2(z, y))$.

Package **relations** implements finite relations as an S3 class which allows for a variety of representations (even though currently, typically dense array representations of the incidences are employed). Other than by the generator, relations can be obtained by coercion via the generic function `as.relation()`, which has methods for at least logical and numeric vectors, unordered and ordered factors, arrays including matrices, and data frames. Unordered factors are coerced to equivalence relations; ordered factors and numeric vectors are coerced to order relations. Logical vectors give unary relations (predicates). A (feasible) k-dimensional array is taken as the incidence of a k-ary relation. Finally, a data frame is taken as a relation table. Note that missing values will be propagated in the coercion.

`endorelation()` is a wrapper for `relation()`, trying to guess a suitable domain from its arguments to create an endorelation. If a domain is given, all labels are combined and the result (as a list) recycled as needed.

Basic relation operations are available as group methods: `min()` and `max()` give the meet and join, and `range()` a relation ensemble with these two. Comparison operators implement the natural ordering in the relation lattice. Where applicable, `!` gives the complement (negation), `&` and `|` intersection and union, and `*` composition, respectively. Finally, `t()` gives the transpose and `codual()` the codual.

There is a `plot()` method for certain crisp endorelations provided that package **Rgraphviz** is available.

For crisp endorelations R, `sym()` and `asy()` give the symmetric and asymmetric parts of R, defined as the intersection of R with its transpose, or, respectively, with its codual (the complement of its

transpose).

The `summary()` method applies all predicates available and returns a logical vector with the corresponding results.

References

S. A. Clark (1990), A folk meta-theorem in the foundations of utility theory. *Mathematical Social Sciences*, **19**/3, 253–267. doi:10.1016/01654896(90)90065F.

S. Ovchinnikov (1991), Similarity relations, fuzzy partitions, and fuzzy orderings. *Fuzzy Sets and Systems*, **40**/1, 107–126. doi:10.1016/01650114(91)90048U.

See Also

`relation_incidence()` for obtaining incidences; `relation_domain()` for determining domain, arity, and size; `relation_graph()` for determining the graph of a relation; `relation_charfun()` for determining the characteristic function; predicates for available predicate functions; and algebra for further operations defined on relations.

Examples

```
require("sets") # set(), tuple() etc.

## A relation created by specifying the graph:
R <- relation(graph = data.frame(A = c(1, 1:3), B = c(2:4, 4)))
relation_incidence(R)
## extract domain
relation_domain(R)
## extract graph
relation_graph(R)
## both ("a pair of domain and graph" ...)
as.tuple(R)

## (Almost) the same using the set specification
## (the domain labels are missing).
R <- relation(graph = set(tuple(1,2), tuple(1,3),
                          tuple(2,4), tuple(3,4)))
## equivalent to:
## relation(graph = list(c(1,2), c(1,3), c(2,4), c(3,4)))
relation_incidence(R)

## Explicitly specifying the domain:
R <- relation(domain = list(A = letters[1:3], B = LETTERS[1:4]),
              graph = set(tuple("a","B"), tuple("a","C"),
                          tuple("b","D"), tuple("c","D")))
relation_incidence(R)

## Domains can be composed of arbitrary R objects:
R <- relation(domain = set(c, "test"),
              graph = set(tuple(c, c), tuple(c, "test")))
relation_incidence(R)
```

```
## Characteristic function ("a divides b"):
R <- relation(domain = list(1 : 10, 1 : 10),
              charfun = function(a, b) b %% a == 0)
relation_incidence(R)
## R is a partial order: plot the Hasse diagram provided that
## Rgraphviz is available:
if(require("Rgraphviz")) plot(R)

## conversions and operators
x <- matrix(0, 3, 3)
R1 <- as.relation(row(x) >= col(x))
R2 <- as.relation(row(x) <= col(x))
R3 <- as.relation(row(x) <  col(x))
relation_incidence(max(R1, R2))
relation_incidence(min(R1, R2))
R3 < R2
relation_incidence(R1 * R2)
relation_incidence(! R1)
relation_incidence(t(R2))

### endorelation
s <- set(pair("a","b"), pair("c","d"))
relation_incidence(relation(graph = s))
relation_incidence(endorelation(graph = s))
```

---

scores *Relation Scores*

Description

Compute scores for the tuples of (ensembles of) endorelations.

Usage

```
## S3 method for class 'relation'
relation_scores(x,
                method = c("ranks", "Barthelemy/Monjardet",
                           "Borda", "Kendall", "Wei",
                           "differential", "Copeland"),
                normalize = FALSE, ...)
## S3 method for class 'relation_ensemble'
relation_scores(x,
                method = c("Borda", "Kendall", "differential",
                           "Copeland"),
                normalize = FALSE,
                weights = 1, ...)
```

Arguments

x	an object inheriting from class relation, representing an endorelation.
method	character string indicating the method (see **Details**).
normalize	logical indicating whether the score vector should be normalized to sum up to 1.
weights	Numeric vector of weights used in incidence aggregation, recycled as needed.
...	further arguments to be passed to methods.

Details

In the following, consider an endorelation R on n objects. Let the *in-degree* $I(x)$ and *out-degree* $O(x)$ of an object x be defined as the numbers of objects y such that yRx and, respectively, xRy, and let $D(x) = I(x) - O(x)$ be the *differential* of x (see Regenwetter and Rykhlevskaia (2004)). Note that I and O are given by the column sums and row sums of the incidence matrix of R. If R is a preference relation with a $\leq$ interpretation, $D(x)$ is the difference between the numbers of objects dominated by x (i.e., $< x$) and dominating x (i.e., $> x$), as "ties" cancel out.

relation_score() is generic with methods for relations and relation ensembles. Available built-in score methods for the relation method are as follows:

"ranks" generalized ranks. A linear transformation of the differential D to the range from 1 to n. An additional argument decreasing can be used to specify the order of the ranks. By default, or if decreasing is true, objects are ranked according to decreasing differential ("from the largest to the smallest" in the $\leq$ preference context) using $(n + 1 - D(x))/2$. Otherwise, if decreasing is false, objects are ranked via $(n + 1 + D(x))/2$ ("from the smallest to the largest"). See Regenwetter and Rykhlevskaia (2004) for more details on generalized ranks.

"Barthelemy/Monjardet" $(M(x) + N(x) - 1)/2$, where $M(x)$ and $N(x)$ are the numbers of objects y such that yRx, and yRx and not xRy, respectively. If R is a $\leq$ preference relation, we get the number of dominated objects plus half the number of the equivalent objects minus 1 (the "usual" average ranks minus one if the relation is complete). See Barthélemy and Monjardet (1981).

"Borda", "Kendall" the out-degrees. See Borda (1770) and Kendall (1955).

"Wei" the eigenvector corresponding to the greatest eigenvalue of the incidence matrix of the complement of R. See Wei (1952).

"differential", "Copeland" the differentials, equivalent to the negative *net flow* of Bouyssou (1992), and also to the Copeland scores.

For relation ensembles, currently only "differential"/"Copeland" and "Borda"/"Kendall" are implemented. They are computed on the aggregated incidences of the ensembles' relations.

Definitions of scores for "preference relations" R are somewhat ambiguous because R can encode $\leq$ or $\geq$ (or strict variants thereof) relationships (and all such variants are used in the literature). Package **relations** generally assumes a $\leq$ encoding, and that scores in the strict sense should increase with preference (the most preferred get the highest scores) whereas ranks decrease with preference (the most preferred get the lowest ranks).

Value

A vector of scores, with names taken from the relation domain labels.

References

J.-P. Barthélemy and B. Monjardet (1981), The median procedure in cluster analysis and social choice theory. *Mathematical Social Sciences*, **1**, 235–267. doi:10.1016/01654896(81)90041X.

J. C. Borda (1781), Mémoire sur les élections au scrutin. Histoire de l'Académie Royale des Sciences.

D. Bouyssou (1992), Ranking methods based on valued preference relations: A characterization of the net flow network. *European Journal of Operational Research*, **60**, 61–67. doi:10.1016/0377-2217(92)903335.

M. Kendall (1955), Further contributions to the theory of paired comparisons. *Biometrics*, **11**, 43–62. doi:10.2307/3001479.

M. Regenwetter and E. Rykhlevskaia (2004), On the (numerical) ranking associated with any finite binary relation. *Journal of Mathematical Psychology*, **48**, 239–246. doi:10.1016/j.jmp.2004.03.003.

T. H. Wei (1952). *The algebraic foundation of ranking theory*. Unpublished thesis, Cambridge University.

Examples

```
## Example taken from Cook and Cress (1992, p.74)
I <- matrix(c(0, 0, 1, 1, 1,
              1, 0, 0, 0, 1,
              0, 1, 0, 0, 1,
              0, 1, 1, 0, 0,
              0, 0, 0, 1, 0),
            ncol = 5,
            byrow = TRUE)
R <- relation(domain = letters[1:5], incidence = I)

## Note that this is a "preference matrix", so take complement:
R <- !R

## Compare Kendall and Wei scores
cbind(
      Kendall = relation_scores(R, method = "Kendall", normalize = TRUE),
      Wei = relation_scores(R, method = "Wei", normalize = TRUE)
     )

## Example taken from Cook and Cress (1992, p.136)
## Note that the results indicated for the Copeland scores have
## (erroneously?) been computed from the *unweighted* votes.
## Also, they report the votes as strict preferences, so we
## create the dual relations.

D <- letters[1:5]
X <- as.relation(ordered(D, levels = c("b", "c", "a", "d", "e")))
Y <- as.relation(ordered(D, levels = c("d", "a", "e", "c", "b")))
Z <- as.relation(ordered(D, levels = c("e", "c", "b", "a", "d")))
E <- relation_ensemble(X, Y, Z)
relation_scores(E, "Copeland")
relation_scores(E, "Borda", weights = c(4, 3, 2))
```

setters *Modify Relations*

Description

Modify relations by (re)setting their domain, graph, or incidences.

Usage

```
relation_domain(x) <- value
relation_domain_names(x) <- value
relation_graph(x) <- value
relation_incidence(x) <- value
```

Arguments

x	an R object inheriting from class relation.
value	for setting the domain, a tuple (or list) as long as the arity of the relation x, with sets of cardinality (for lists: numbers of elements) identical to the size of x. For setting the graph, either a set of tuples of equal lengths (arity of the relation) or a data frame or something coercible to this, with the values of the components of the given tuples (rows) always elements of the corresponding elements of the domain of x. For setting incidences, a numeric array with values in the unit interval or a logical array, with dimension the size of the relation x. For setting the domain names, a character vector as long as the arity of the relation x.

See Also

relation_domain() for getting the domain of a relation; relation_domain_names() for getting the domain names; relation_graph() for getting the graph; relation_incidence() for getting the incidences; relation() for basic information.

Examples

```
R <- as.relation(1 : 3)
print(R)

relation_domain(R)
## tuple format:
require("sets") # set(), pair() etc.
relation_domain(R) <- pair(X = set("a","b","c"), Y = set("A","B","C"))
relation_domain(R)
## the same in list format:
relation_domain(R) <- list(X = letters[1:3], Y = LETTERS[1:3])
relation_domain(R)
```

```
relation_domain_names(R) <- c("XX","YY")
relation_domain_names(R)

relation_incidence(R)
relation_incidence(R) <- diag(1, 3, 3)
relation_incidence(R)

relation_graph(R)
## set format:
relation_graph(R) <- set(pair("a","B"), pair("a","C"), pair("b","C"))
relation_graph(R)
## the same in data frame format:
relation_graph(R) <-
    data.frame(c("a", "a", "b"), c("B", "C", "C"),
               stringsAsFactors = FALSE)
relation_graph(R)
```

SVMBench *SVM Benchmarking Data and Consensus Relations*

Description

SVM_Benchmarking_Classification and SVM_Benchmarking_Regression represent the results of a benchmark study (Meyer, Leisch and Hornik, 2003) comparing Support Vector Machines to other predictive methods on real and artificial data sets involving classification and regression methods, respectively. In addition, SVM_Benchmarking_Classification_Consensus and SVM_Benchmarking_Regression_Consensus provide consensus rankings derived from these data.

Usage

```
data("SVM_Benchmarking_Classification")
data("SVM_Benchmarking_Regression")
data("SVM_Benchmarking_Classification_Consensus")
data("SVM_Benchmarking_Regression_Consensus")
```

Format

SVM_Benchmarking_Classification (SVM_Benchmarking_Regression) is an ensemble of 21 (12) relations representing pairwise comparisons of 17 classification (10 regression) methods on 21 (12) data sets. Each relation of the ensemble summarizes the results for a particular data set. The relations are reflexive endorelations on the set of methods employed, with a pair (a, b) of distinct methods contained in a relation iff both delivered results on the corresponding data set and a did not perform significantly better than b at the 5% level. Since some methods failed on some data sets, the relations are not guaranteed to be complete or transitive. See Meyer et al. (2003) for details on the experimental design of the benchmark study, and Hornik and Meyer (2007) for the pairwise comparisons.

The corresponding consensus objects are lists of ensembles of consensus relations fitted to the benchmark results. For each of the following three endorelation families: `SD/L` ("linear orders"), `SD/O` ("partial orders"), and `SD/W` ("weak orders"), *all* possible consensus relations have been computed (see `relation_consensus`). For both classification and regression, the three relation ensembles obtained are provided as a named list of length 3. See Hornik & Meyer (2007) for details on the meta-analysis.

Source

D. Meyer, F. Leisch, and K. Hornik (2003), The support vector machine under test. *Neurocomputing*, **55**, 169–186. doi:10.1016/S09252312(03)004314.

K. Hornik and D. Meyer (2007), Deriving consensus rankings from benchmarking experiments. In R. Decker and H.-J. Lenz, *Advances in Data Analysis*. Studies in Classification, Data Analysis, and Knowledge Organization. Springer-Verlag: Heidelberg, 163–170.

Examples

```
data("SVM_Benchmarking_Classification")

## 21 data sets
names(SVM_Benchmarking_Classification)

## 17 methods
relation_domain(SVM_Benchmarking_Classification)

## select weak orders
weak_orders <-
    Filter(relation_is_weak_order, SVM_Benchmarking_Classification)

## only the artifical data sets yield weak orders
names(weak_orders)

## visualize them using Hasse diagrams
if(require("Rgraphviz")) plot(weak_orders)

## Same for regression:
data("SVM_Benchmarking_Regression")

## 12 data sets
names(SVM_Benchmarking_Regression)

## 10 methods
relation_domain(SVM_Benchmarking_Regression)

## select weak orders
weak_orders <-
    Filter(relation_is_weak_order, SVM_Benchmarking_Regression)

## only two of the artifical data sets yield weak orders
names(weak_orders)
```

```
## visualize them using Hasse diagrams
if(require("Rgraphviz")) plot(weak_orders)

## Consensus solutions:

data("SVM_Benchmarking_Classification_Consensus")
data("SVM_Benchmarking_Regression_Consensus")

## The solutions for the three families are not unique
print(SVM_Benchmarking_Classification_Consensus)
print(SVM_Benchmarking_Regression_Consensus)

## visualize the consensus weak orders
classW <- SVM_Benchmarking_Classification_Consensus$W
regrW <- SVM_Benchmarking_Regression_Consensus$W
if(require("Rgraphviz")) {
    plot(classW)
    plot(regrW)
}

## in tabular style:
ranking <- function(x) rev(names(sort(relation_class_ids(x))))
sapply(classW, ranking)
sapply(regrW, ranking)

## (prettier and more informative:)
relation_classes(classW[[1L]])
```

`table` *Relation Table*

Description

Returns a tabular representation of a relation like a "view" of a relational database table.

Usage

```
relation_table(x, memberships = TRUE)
```

Arguments

`x`	an object inheriting from class `relation`.
`memberships`	logical; should membership vector (if any) be added to the table?

Value

An object of class `relation_table`, inheriting from class `data.frame`.

See Also

relation_join()

Examples

```
R <- data.frame(Name = c("David", "John"),
                Age = c(33, 66),
                stringsAsFactors = FALSE)
R <- as.relation(R)
relation_table(R)
```

---

trace *Traces of Endorelations*

Description

Compute the left or right trace of an endorelation.

Usage

```
relation_trace(x, which)
```

Arguments

x	an endorelation.
which	one of "left" or "right", or a unique abbreviation thereof.

Details

Let R be a crisp endorelation. The left and right trace of R contain all pairs x, y for which zRx implies zRy for all z (left trace) or yRz implies xRz for all z (right trace), respectively. These are the largest (in the natural ordering of relations with the same domain) relations such that $R * S \leq R$ or $S * R \leq R$, respectively (where $*$ denotes composition). In the fuzzy case, the memberships of the traces can be defined as the infima over the corresponding fuzzy membership implications. See Chapter 2.3 in Fodor and Roubens (1994) for more information.

References

J. Fodor and M. Roubens (1994), *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer Academic Publishers: Dordrecht.

`transform` *Transform incidences*

Description

Carry out transformations between incidence matrices from endorelations and other codings.

Usage

```
transform_incidences(x, from = c("PO", "SO", "01", "-1+1"),
                        to = c("PO", "SO", "01", "-1+1"))
```

Arguments

`x`	An incidence matrix from an endorelation.
`from`, `to`	The coding scheme (see **Details**).

Details

In the following, we consider an incidence matrix X with cells x_{jk} of a relation R with tuples (a_j, b_k).

For the `"PO"` ("Preference Order") coding, X is a 0/1 matrix, and $a_j R b_k$ iff $x_{jk} = 1$. It follows in particular that if both x_{jk} and x_{kj} are 0, the corresponding pair (a_j, b_k) is not contained in R, i.e., a_j and b_k are unrelated.

For the `"SO"` (""Strict Order"") coding, X is a 0/1 matrix with possible `NA` values. As for `"PO"`, $a_j R b_k$ iff $x_{jk} = 1$, but at most one of x_{jk} and x_{kj} can be 1. If both are missing (`NA`), a_j and b_k are unrelated.

For the `"01"` coding, X is a matrix with values 0, 1, or 0.5. The coding is similar to `"SO"`, except that `NA` is represented by 0.5.

For the `"-1+1"` coding, X is a matrix with values -1, 0, or 1. The coding is similar to `"SO"`, except that `NA` is represented by 0, and $x_{jk} = -1$ if *not* $a_j R b_k$.

See Also

`relation_incidence()`.

Examples

```
require("sets") # set(), pair() etc.
x <- relation(domain = c(1,2,3,4),
              graph = set(pair(1,2), pair(4,2), pair(1,3), pair(1,4),
                          pair(3,2), pair(2,1)))
inc <- relation_incidence(x)
print(inc)

transform_incidences(inc, to = "SO")
transform_incidences(inc, to = "01")
```

```
transform_incidences(inc, to = "-1+1")

## transformations should be loss-free:
inc2 <- transform_incidences(inc, from = "PO", to = "-1+1")
inc2 <- transform_incidences(inc2, from = "-1+1", to = "SO")
inc2 <- transform_incidences(inc2, from = "SO", to = "01")
inc2 <- transform_incidences(inc2, from = "01", to = "PO")
stopifnot(identical(inc, inc2))
```

`violations` *Violations of Relation Properties*

Description

Computes a measure of remoteness of a relation from a specified property.

Usage

```
relation_violations(x,
                    property =
                    c("complete", "match",
                      "reflexive", "irreflexive", "coreflexive",
                      "symmetric", "antisymmetric", "asymmetric",
                      "transitive", "negatively_transitive",
                      "Ferrers", "semitransitive",
                      "trichotomous",
                      "Euclidean"),
                    tuples = FALSE,
                    na.rm = FALSE)
```

Arguments

`x`	an endorelation.
`property`	a character string specifying one of the properties for which the number of violations can be computed.
`tuples`	a logical indicating whether to return the amount of violations (default), or the tuples for which the property is violated.
`na.rm`	a logical indicating whether to remove tuples for which the property information is not available (due to missing memberships).

Value

If `tuples` is false (default), the amount of violations for the specified property: for crisp relations, the minimum number of object tuples involved in the definition of the property (e.g., singletons for reflexivity, pairs for antisymmetry, and triples for transitivity) that must be modified/added/removed to make the relation satisfy the property.

If `tuples` is true, a set of tuples of objects for which the respective property is violated.

See Also

predicates for the definitions of the properties.

Examples

```
## partial order:
R <- as.relation(1:3)
relation_incidence(R)
## R clearly is transitive, but not symmetric:
relation_violations(R, "transitive")
relation_violations(R, "symmetric")
## Pairs for which symmetry is violated:
relation_violations(R, "symmetric", TRUE)

## create a simple relation:
require("sets") # set(), pair() etc.
R <- relation(domain = letters[1:2],
              graph = set(pair("a","b"), pair("b","a")))
relation_incidence(R)
## R is clearly symmetric, but not antisymmetric:
relation_violations(R, "symmetric")
relation_violations(R, "antisymmetric")
```

Index