Good Relations with R

David Meyer and Kurt Hornik

2016-07-01

Given k sets of objects $X_1, \ldots, X_k$, a k-ary relation R on $D(R) = (X_1, \ldots, X_k)$ is a subset $G(R)$ of the Cartesian product $X_1 \times \cdots \times X_k$. I.e., $D(R)$ is a k-tuple of sets and $G(R)$ is a set of k-tuples. We refer to $D(R)$ and $G(R)$ as the *domain* and the *graph* of the relation R, respectively (alternative notions are that of *ground* and *figure*, respectively).

Relations are a very fundamental mathematical concept: well-known examples include the linear order defined on the set of integers, the equivalence relation, notions of preference relations used in economics and political sciences, etc. Package **relations** provides data structures along with common basic operations for relations and also relation ensembles (collections of relations with the same domain), as well as various algorithms for finding suitable consensus relations for given relation ensembles.

1 Relations and Relation Ensembles

1.1 Relations

For a k-ary relation R with domain $D(R) = (X_1, \ldots, X_k)$, we refer to $s = (s_1, \ldots, s_k)$, where each s_i gives the cardinality of X_i, as the *size* of the relation. Note that often, relations are identified with their graph; strictly speaking, the relation is the *pair* $(D(R), G(R))$. We say that a k-tuple t is *contained* in the relation R iff it is an element of $G(R)$. The *incidence* (array) $I(R)$ of R is a k-dimensional $0/1$ array of size s whose elements indicate whether the corresponding k-tuples are contained in R or not.

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 `relation()`, 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 (object by attribute representation of the relation graph). Note that missing values will be propagated in the coercion.

```
> ## A relation created by specifying the graph:
> R <- relation(graph = data.frame(A = c(1, 1:3), B = c(2:4, 4)))
> ## extract domain
> relation_domain(R)

Relation domain:
A pair (A, B) with elements:
{1, 2, 3}
{2, 3, 4}

> ## extract graph
> relation_graph(R)
```

```
Relation graph:
A set with pairs ("A", "B"):
(1, 2)
(1, 3)
(2, 4)
(3, 4)

> ## both ("a pair of domain and graph" ...)
> as.tuple(R)

(Domain = (A = {1, 2, 3}, B = {2, 3, 4}), Graph = {(1, 2), (1, 3), (2,
 4), (3, 4)})

> ## extract incidence
> relation_incidence(R)

Incidences:
   B
A   2 3 4
  1 1 1 0
  2 0 0 1
  3 0 0 1

> ## (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(1:2, c(1,3), c(2,4), c(3,4)))
> relation_incidence(R)

Incidences:
  2 3 4
1 1 1 0
2 0 0 1
3 0 0 1

> ## 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)

Incidences:
             test <<function>>
test            0            0
<<function>>    1            1

> as.relation(1:3)

A binary relation of size 3 x 3.

> relation_graph(as.relation(c(TRUE, FALSE, TRUE)))

Relation graph:
A set with singletons:
("1")
("3")
```

```
> relation_graph(as.relation(factor(c("A", "B", "A"))))

Relation graph:
A set with pairs:
("1", "1")
("1", "3")
("2", "2")
("3", "1")
("3", "3")
```

Note that while coercion uses the factor values to obtain the graph, it infers the domain from the factor names if available and unique, or from the values if unique:

```
> relation_graph(as.relation(factor(c(X = "A", Y = "B", Z = "A"))))

Relation graph:
A set with pairs:
("X", "X")
("X", "Z")
("Y", "Y")
("Z", "X")
("Z", "Z")

> relation_graph(as.relation(factor(c("A", "B", "C"))))

Relation graph:
A set with pairs:
("A", "A")
("B", "B")
("C", "C")
```

The *characteristic function* f_R (sometimes also referred to as indicator function) of a relation R is the predicate (Boolean-valued) function on the Cartesian product $X_1 \times \cdots \times X_k$ such that $f_R(t)$ is true iff the k-tuple t is in $G(R)$. Characteristic functions can both be recovered from a relation via `relation_charfun()`, and be used in the generator for the creation. In the following, `R` represents "a divides b":

```
> divides <- function(a, b) b %% a == 0
> R <- relation(domain = list(1 : 10, 1 : 10), charfun = divides)
> R

A binary relation of size 10 x 10.

> "%|%" <- relation_charfun(R)
> 2L %|% 6L

[1] TRUE

> 2:4 %|% 6L

[1]  TRUE  TRUE FALSE

> 2L %|% c(2:3, 6L)

[1]  TRUE FALSE  TRUE

> "%|%"(2L, 6L)
```

```
[1] TRUE
```

Quite a few `relation_is_foo()` predicate functions are available. For example, relations with arity 2, 3, and 4 are typically referred to as *binary*, *ternary*, and *quaternary* relations, respectively—the corresponding functions in package **relations** are `relation_is_binary()`, `relation_is_ternary()`, etc. For binary relations R, it is customary to write xRy iff (x, y) is contained in R. For predicates available on binary relations, see Table 1. An *endorelation* on X (or binary relation *over* X) is a binary relation with domain (X, X). Endorelations may or may not have certain basic properties (such as transitivity, reflexivity, etc.) which can be tested in **relations** using the corresponding predicates (see Table 2 for an overview). Some combinations of these basic properties have special names because of their widespread use (such as linear order or weak order), and can again be tested using the functions provided (see Table 3).

```
> R <- as.relation(1:5)
> relation_is(R, "binary")

[1] TRUE

> relation_is(R, "transitive")

[1] TRUE

> relation_is(R, "partial_order")

[1] TRUE
```

Relations with the same domain can naturally be ordered according to their graphs. I.e., $R_1 \leq R_2$ iff $G(R_1)$ is a subset of $G(R_2)$, or equivalently, if every k-tuple t contained in R_1 is also contained in R_2. 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 cardinality of the symmetric difference of the graphs (the number of tuples in exactly one of the relation graphs). This "symdiff" dissimilarity between (ensembles of) relations can be computed by `relation_dissimilarity()`.

```
> x <- matrix(0, 3L, 3L)
> 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))

Incidences:
  1 2 3
1 1 1 1
2 1 1 1
3 1 1 1

> relation_incidence(min(R1, R2))

Incidences:
  1 2 3
1 1 0 0
2 0 1 0
3 0 0 1

> R3 < R2
```

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.

Table 1: Some properties *foo* of binary relations—the predicates in **relations** are `relation_is_`*`foo`*`()` (with hyphens replaced by underscores).

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.
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 that 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.

Table 2: Some properties *bar* of endorelations—the predicates in **relations** are `relation_is_`*`bar`*`()` (with spaces replaced by underscores).

preorder	reflexive and transitive.
quasiorder	the same as preorder.
equivalence	a symmetric preorder.
weak order	complete and transitive.
preference	the same as weak order.
partial order	an antisymmetric preorder.
strict partial order	irreflexive, transitive and antisymmetric.
linear order	a complete partial order.
strict linear order	a complete strict partial order.
match	strongly complete.
tournament	complete and antisymmetric.
interval order	complete and Ferrers.
semiorder	a semitransitive interval order.

Table 3: Some categories *baz* of endorelations—the predicates in **relations** are `relation_is_`*`baz`*`()` (with spaces replaced by underscores).

```
[1] TRUE

> relation_dissimilarity(min(R1, R2), max(R1, R2))

     [,1]
[1,]    6
```

The *complement* (or negation) R^c of a relation R is the relation with domain $D(R)$ whose graph is the complement of $G(R)$, i.e., which contains exactly the tuples not contained in R. For binary relations R_1 and R_2 with domains (X, Y) and (Y, Z), the *composition* $S = R_1 * R_2$ of R_1 and R_2 is defined by taking xSz iff there is a 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 xR^ty iff yRx.

```
> relation_incidence(! R1)

Incidences:
  1 2 3
1 0 1 1
2 0 0 1
3 0 0 0

> relation_incidence(R1 * R2)

Incidences:
  1 2 3
1 1 1 1
2 1 1 1
3 1 1 1

> relation_incidence(t(R2))

Incidences:
  1 2 3
1 1 0 0
2 1 1 0
3 1 1 1
```

There is a `plot()` method for certain endorelations (currently, only complete or antisymmetric transitive relations are supported) provided that package **Rgraphviz** (Hansen, Gentry, Long, Gentleman, Falcon, Hahne, and Sarkar, 2017) is available, creating a Hasse diagram of the relation. The following code produces the Hasse diagram corresponding to the inclusion relation on the power set of $\{a, b, c\}$ which is a partial order (see Figure 1).

```
> ps <- 2 ^ set("a", "b", "c")
> inc <- set_outer(ps, "<=")
> if (require("Rgraphviz")) plot(relation(incidence = inc))
```

1.2 Relation Ensembles

"Relation ensembles" are 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 (of class relation_ensemble and inheriting from tuple), making it possible to use `lapply()` for computations on the individual relations in the ensemble. Relation ensembles can be created via `relation_ensemble()`, or by coercion via the generic function `as.relation_ensemble()` which has methods for at least data frames (regarding each variable as a separate relation). Available

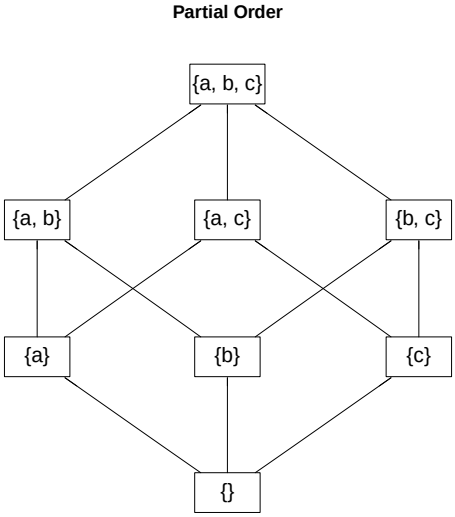

Figure 1: Hasse Diagram of the inclusion relation on the power set of $\{a, b, c\}$.

methods for relation ensembles include those for subscripting, `c()`, `t()`, `rep()`, `print()`, and `plot()`. In addition, there are summary methods defined (`min()`, `max()`, and `range()`). Other operations work element-wise like on tuples due to the inheritance.

The Cetacea data set (Vescia, 1985) is a data frame with 15 variables relating to morphology, osteology, or behavior, with both self-explanatory names and levels, and a common zoological classification (variable `CLASS`) for 36 types of cetacea. We consider each variable an equivalence relation on the objects, excluding 2 variables with missing values, giving a relation ensemble of length 14 (number of complete variables in the data set).

```
> data("Cetacea")
> ind <- vapply(Cetacea, function(s) all(!is.na(s)), TRUE)
> relations <- as.relation_ensemble(Cetacea[, ind])
> print(relations)

An ensemble of 14 relations of size 36 x 36.
```

Available methods for relation ensembles allow to determine duplicated (relation) entries, to replicate and combine, and extract unique elements:

```
> any(duplicated(relations))

[1] FALSE

> thrice <- c(rep(relations, 2L), relations)
> all.equal(unique(thrice), relations)

[1] "names for current but not for target"
```

Note that `unique()` does not preserve attributes, and hence names. In case one wants otherwise, one can subscript by a logical vector indicating the non-duplicated entries:

```
> all.equal(thrice[!duplicated(thrice)], relations)

[1] TRUE
```

Relation (cross-)dissimilarities can be computed for relations and ensembles thereof:

```
> relation_dissimilarity(relations[1 : 2], relations["CLASS"])
```

```
                 CLASS
NECK               584
FORM_OF_THE_HEAD   330
```

To determine which single variable is "closest" to the zoological classification:

```
> d <- relation_dissimilarity(relations)
> sort(as.matrix(d)[, "CLASS"])[-1L]
```

```
                          BLOW_HOLE                           DORSAL_FIN
                                190                                  240
                        SET_OF_TEETH                             FLIPPERS
                                288                                  298
                     FORM_OF_THE_HEAD                             FEEDING
                                330                                  382
                              HABITAT                                BEAK
                                398                                  456
                               COLOR LONGITUDINAL_FURROWS_ON_THE_THROAT
                                494                                  506
                    CERVICAL_VERTEBRAE                     SIZE_OF_THE_HEAD
                                508                                  542
                              NECK
                                584
```

There is also an Ops group method for relation ensembles which works elementwise (in essence, as for tuples):

```
> complement <- !relations
> complement
```

```
An ensemble of 14 relations of size 36 x 36.
```

2 Relational Algebra

In addition to the basic operations defined on relations, the package provides 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. Thus, a "tuple" in this Codd sense is a set of mappings from the attribute names into the union of the attribute domains. The functions defined in **relations**, however, preserve and respect the column ordering.

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 corresponding function in package **relations** is `relation_projection()`.

```
> ## 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)
```

```
 Name   Age Weight
 Helena 54  54
 Sally  28  64
 George 29  70
 Harry  34  80
 Peter  34  80
```

```
> relation_table(relation_projection(Person, c("Age", "Weight")))
```

```
 Age Weight
 54  54
 28  64
 29  70
 34  80
```

(Note that Harry and Peter have the same age and weight.)

The *selection* of a relation is the relation obtained by taking a subset of the relation graph, defined by some logical expression. The corresponding function in **relations** is `relation_selection()`.

```
> ## selection
> relation_table(R1 <- relation_selection(Person, Age < 29))
```

```
 Name  Age Weight
 Sally 28  64
```

```
> relation_table(R2 <- relation_selection(Person, Age >= 34))
```

```
 Name   Age Weight
 Helena 54  54
 Harry  34  80
 Peter  34  80
```

```
> relation_table(R3 <- relation_selection(Person, Age == Weight))
```

```
 Name   Age Weight
 Helena 54  54
```

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. One can use `-` as a shortcut for `relation_complement()`, and `%U%` or `|` for `relation_union()`. The difference between `%U%` and `|` is that the latter only works for identical domains.

```
> ## union
> relation_table(R1 %U% R2)
```

```
 Name   Age Weight
 Helena 54  54
 Sally  28  64
 Harry  34  80
 Peter  34  80
```

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

```
 Name   Age Weight
 Helena 54  54
 Harry  34  80
 Peter  34  80
```

```
> ## complement
> relation_table(Person - R2)

 Name   Age Weight
 Sally  28  64
 George 29  70
```

The *intersection* (*symmetric difference*) of two relations is the relation with all tuples they have (do not have) in common. One can use `&` instead of `relation_intersection()` in case of identical domains.

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

 Name   Age Weight
 Helena 54  54

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

 Name   Age Weight
 Helena 54  54

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

 Name  Age Weight
 Harry 34  80
 Peter 34  80
```

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. A shortcut for `relation_cartesian()` is `%><%`.

```
> ## 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)

 Name    EmpId DeptName
 George  3401  Finance
 Harry   3415  Finance
 John    3999  N.N.
 Harriet 2202  Sales
 Sally   2241  Sales

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

```
 DeptName   Manager
 Production Charles
 Finance    George
 Sales      Harriet
```

```
> relation_table(Employee %><% Dept)
```

```
 Name    EmpId DeptName DeptName   Manager
 George  3401  Finance  Production Charles
 Harry   3415  Finance  Production Charles
 John    3999  N.N.     Production Charles
 Harriet 2202  Sales    Production Charles
 Sally   2241  Sales    Production Charles
 George  3401  Finance  Finance    George
 Harry   3415  Finance  Finance    George
 John    3999  N.N.     Finance    George
 Harriet 2202  Sales    Finance    George
 Sally   2241  Sales    Finance    George
 George  3401  Finance  Sales      Harriet
 Harry   3415  Finance  Sales      Harriet
 John    3999  N.N.     Sales      Harriet
 Harriet 2202  Sales    Sales      Harriet
 Sally   2241  Sales    Sales      Harriet
```

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 shortcuts for `relation_division()` and `relation_remainder()` are `%/%` and `%%`, respectively.

```
> ## 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)
```

```
 Student Task
 Eugene  Compiler1
 Fred    Compiler1
 Eugene  Database1
 Fred    Database1
 Sara    Database1
 Fred    Database2
 Sara    Database2
```

```
> DBProject <- data.frame(Task = c("Database1", "Database2"),
+                         stringsAsFactors = FALSE)
> DBProject <- as.relation(DBProject)
> relation_table(DBProject)
```

```
 Task
 Database1
 Database2
```

```
> relation_table(Completed %/% DBProject)
```

```
 Student
 Fred
 Sara
```

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

```
 Student Task
 Eugene  Compiler1
 Fred    Compiler1
 Eugene  Database1
```

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 of `relation_join()` 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`. Alternatively, one can use the operators `%|><|%`, `%=><%`, `%><=%`, and `%=><=%` for the natural join, left join, right join, and full outer join, respectively.

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

```
 Name    EmpId DeptName Manager
 George  3401  Finance  George
 Harry   3415  Finance  George
 Harriet 2202  Sales    Harriet
 Sally   2241  Sales    Harriet
```

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

```
 Name    EmpId DeptName Manager
 George  3401  Finance  George
 Harry   3415  Finance  George
 Harriet 2202  Sales    Harriet
 Sally   2241  Sales    Harriet
 John    3999  N.N.     NA
```

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

```
 Name    EmpId DeptName   Manager
 NA        NA  Production Charles
 George  3401  Finance    George
 Harry   3415  Finance    George
 Harriet 2202  Sales      Harriet
 Sally   2241  Sales      Harriet
```

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

```
 Name    EmpId DeptName   Manager
 NA        NA  Production Charles
 George  3401  Finance    George
 Harry   3415  Finance    George
 Harriet 2202  Sales      Harriet
 Sally   2241  Sales      Harriet
 John    3999  N.N.       NA
```

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. Shortcuts for `relation_semijoin()` are `%|><%` and `%><|%` for left and right semijoin, respectively.

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

 Name    EmpId DeptName
 George  3401  Finance
 Harry   3415  Finance
 Harriet 2202  Sales
 Sally   2241  Sales

> relation_table(Employee %><|% Dept)

 DeptName Manager
 Finance  George
 Sales    Harriet
```

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. Shortcuts for `relation_antijoin()` are `%|>%` and `%<|%` for left and right antijoin, respectively.

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

 Name EmpId DeptName
 John 3999  N.N.

> relation_table(Employee %<|% Dept)

 DeptName   Manager
 Production Charles
```

3 Consensus Relations

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, 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).

Consensus relations can be computed by `relation_consensus()`, which has the following built-in methods. 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 ("wins" in case of orderings). These are then aggregated across relations by weighted averaging. Finally, objects are ordered according to their aggregated scores.

`"Copeland"` the consensus method proposed by Copeland (1951) is similar to the Borda method, except that the Copeland scores are the number of objects y such that yR_bx, minus the number of objects y such that xR_by ("defeats" in case of orderings).

`"Condorcet"` the consensus method proposed by Condorcet (1785), in fact minimizing the criterion function Φ with d as symmetric difference distance over all possible 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").

`"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.

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

These consensus relations are determined by reformulating the consensus problem as an integer 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** (Theussl and Hornik, 2017, default), **lpSolve** (Buttrey, 2005), **Rsymphony** (Harter, Hornik, and Theussl, 2017), or **Rcplex** (Bravo and Theussl, 2016), 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`.

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

"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.

"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.

"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.

For the Condorcet, CS, symdiff, CKS and PC methods, one can obtain a relation ensemble with *all* consensus relations by setting the control parameter `all` to `TRUE`.

In the following, we first show an example of computing a consensus equivalence (i.e., a consensus partition) of 30 felines repeating the classical analysis of Marcotorchino and Michaud (1982). The data comprises 10 morphological and 4 behavioral variables, taken here as different classifications of the same 30 animals:

```
> data("Felines")
> relations <- as.relation_ensemble(Felines)
```

Now fit an equivalence relation to this, and look at the classes:

```
> E <- relation_consensus(relations, "symdiff/E")
> ids <- relation_class_ids(E)
> split(rownames(Felines), ids)

$`1`
 [1] "LION"     "TIGRE"    "JAGUAR"   "LEOPARD"  "ONCE"     "GUEPARD"
 [7] "PUMA"     "NEBUL"    "YAGUARUN" "CHAUS"    "DORE"     "MERGUAY"
[13] "MARGERIT" "CHINE"    "BENGALE"  "BORNEO"   "NIGRIPES" "MANUL"
[19] "TIGRIN"   "TEMMINCK" "ANDES"

$`2`
[1] "SERVAL"

$`3`
[1] "OCELOT"   "LYNX"     "VIVERRIN" "CAFER"    "ROUILLEU" "MALAIS"

$`4`
[1] "CARACAL" "MARBRE"
```

Next, we demonstrate the computation of consensus preferences, using an example from Cook and Kress (1992, pp. 48ff). The input data is a "preference matrix" of paired comparisons where entry (i, j) is one iff object x_i is preferred to object x_j ($x_i \succ x_j$). We set up the corresponding '$\prec$' relation.

```
> 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 = 5L,
+              byrow = TRUE,
+              dimnames = list(letters[1:5], letters[1:5]))
> R <- as.relation(t(pm))
> relation_incidence(R)

Incidences:
  a b c d e
a 0 0 1 0 0
b 1 0 1 0 0
c 0 0 0 1 1
d 1 1 0 0 1
e 1 1 0 0 0

> relation_is(R, "tournament")

[1] TRUE
```

Next, we seek a linear consensus order:

```
> L <- relation_consensus(R, "symdiff/L")
> relation_incidence(L)

Incidences:
  a b c d e
a 1 0 1 0 0
b 1 1 1 0 0
c 0 0 1 0 0
d 1 1 1 1 1
e 1 1 1 0 1
```

or perhaps more conveniently, the class ids sorted according to increasing preference:

```
> relation_class_ids(L)

a b c d e
4 3 5 1 2
```

Note, however, that this linear order is not unique; we can compute *all* consensus linear orders, and also produce a comparing plot (see Figure 2):

```
> L <- relation_consensus(R, "symdiff/L", control = list(all = TRUE))
> print(L)

An ensemble of 2 relations of size 5 x 5.

> if(require("Rgraphviz")) plot(L)
```

Quite annoyingly, object c comes out first and last, respectively:

```
> lapply(L, relation_class_ids)

[[1]]
a b c d e
4 3 5 1 2

[[2]]
a b c d e
5 4 1 2 3
```

Finally, we compute the closest weak order with at most 3 indifference classes:

```
> W3 <- relation_consensus(R, "symdiff/W", control = list(k = 3))
> relation_incidence(W3)

Incidences:
  a b c d e
a 1 0 0 0 0
b 1 1 0 0 1
c 1 1 1 1 1
d 1 1 1 1 1
e 1 1 0 0 1

> relation_class_ids(W3)

a b c d e
3 2 1 1 2
```

(Note again that this is not unique; there are 6 consensus weak orders with $k = 3$ classes, which can be computed as above by adding `all = TRUE` to the `control` list.)

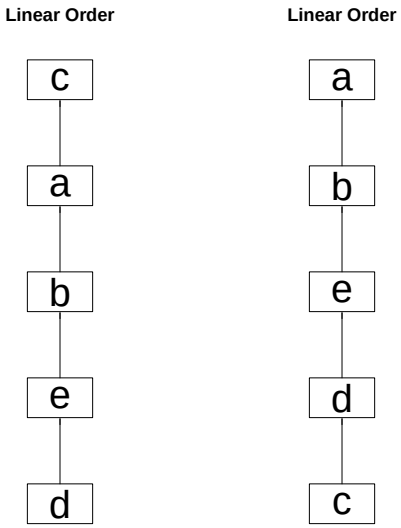

Figure 2: Hasse Diagram of all consensus relations (linear orders) for an example provided by Cook and Kress.

References

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

H. C. Bravo and S. Theussl. **Rcplex***: R Interface to CPLEX*, 2016. URL `https://CRAN.R-project.org/package=Rcplex`. R package version 0.3-3.

S. E. Buttrey. Calling the `lp_solve` linear program software from R, S-PLUS and Excel. *Journal of Statistical Software*, 14(4), 2005. ISSN 1548-7660. doi: 10.18637/jss.v014.i04.

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

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

W. D. Cook and M. Kress. *Ordinal information and preference structures: decision models and applications*. Prentice-Hall, New York, 1992.

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

A. Copeland. A reasonable social welfare function. *mimeo*, 1951. University of Michigan.

K. D. Hansen, J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, and D. Sarkar. **Rgraphviz***: Provides plotting capabilities for R graph objects*, 2017. R package version 2.20.0.

R. Harter, K. Hornik, and S. Theussl. **Rsymphony***: SYMPHONY in R*, 2017. URL `https://CRAN.R-project.org/package=Rsymphony`. R package version 0.1-26.

K. Hornik and D. Meyer. Deriving consensus rankings from benchmarking experiments. In R. Decker and H.-J. Lenz, editors, *Advances in Data Analysis (Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 163–170. Springer-Verlag, 2007.

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

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

S. Theussl and K. Hornik. **Rglpk***: R/GNU Linear Programming Kit Interface*, 2017. URL `https://CRAN.R-project.org/package=Rglpk`. R package version 0.6-3.

G. Vescia. Descriptive classification of cetacea: Whales, porpoises, and dolphins. In J. F. Marcotorchino, J. M. Proth, and J. Janssen, editors, *Data analysis in real life environment: ins and outs of solving problems*. Elsevier Science Publishers B.V., 1985.

Index