`socialranking`: A package for evaluating ordinal power relations in cooperative game theory

Jochen Staudacher, Stefano Moretti, Felix Fritz
(Hochschule Kempten, Université Paris Dauphine)
Contact: jochen.staudacher@hs-kempten.de

2022-04-24

Abstract

This document gives a brief introduction to power relations and social ranking solutions aimed at ranking elements based on their contributions within coalitions. This document accompanies version 0.1.0 of the package `socialranking`.

Keywords: power relation, social ranking solution, cooperative game theory, dominance, cp-majority, Copeland, Kramer-Simpson, ordinal Banzhaf, dominance, lexicographical excellence, relations.

Contents

1 Introduction

In the literature of cooperative games, the notion of *power index* [1–3] has been widely studied to analyze the "influence" of individuals taking into account their ability to force a decision within groups or coalitions. In practical situations, however, the information concerning the strength of coalitions is hardly quantifiable. So, any attempt to numerically represent the influence of groups and individuals clashes with the complex and multi-attribute nature of the problem and it seems more realistic to represent collective decision-making mechanisms using an ordinal coalitional framework based on two main ingredients: a binary relation over groups or coalitions and a ranking over the individuals.

The main objective of the package `socialranking` is to provide answers for the general problem of how to compare the elements of a finite set N given a ranking over the elements of its power-set (the set of all possible subsets of N). To do this, the package `socialranking` implements a portfolio of solutions from the recent literature on *social rankings* [4–10].

1.1 Quick Start

A *power relation* (i.e, a ranking over subsets of a finite set N; see the Section 2 for a formal definition) can be constructed using the `newPowerRelation()` or `newPowerRelationFromString()` functions.

```
library(socialranking)
newPowerRelation(c(1,2), ">", 1, "~", c(), ">", 2)
## Elements: 1 2
## 12 > (1 ~ {}) > 2
```

```
newPowerRelationFromString("ab > a ~ {} > b")
## Elements: a b
## ab > (a ~ {}) > b
```

```
newPowerRelationFromString("12 > 1 ~ {} > 2", asWhat = as.numeric)
## Elements: 1 2
## 12 > (1 ~ {}) > 2
```

Functions used to analyze a given `PowerRelation` object can be grouped into three main categories:

- *Comparison* functions, only comparing two elements;
- *Score* functions, calculating the scores for each element;
- *Ranking* functions, creating `SocialRankingSolution` objects.

Comparison functions	Score functions	Ranking functions
`dominates()`		
`cumulativelyDominates()`	`cumulativeScores()`	
`cpMajorityComparison()` `cpMajorityComparisonScore()`	`copelandScores()` `kramerSimpsonScores()`	`copelandRanking()` `kramerSimpsonRanking()`
	`lexcelScores()`	`lexcelRanking()` `dualLexcelRanking()`
	`ordinalBanzhafScores()`	`ordinalBanzhafRanking()`

Comparison and score functions are often used to evaluate a social ranking solution (see section 2 for a formal definition). Listed below are some of the most prominent functions and solutions introduced in the aforementioned papers.

These functions may be called as follows.

```
pr <- newPowerRelationFromString("ab > ac ~ bc > a ~ c > {} > b")

# a dominates b -> TRUE
dominates(pr, "a", "b")
## [1] TRUE

# b does not dominate a -> FALSE
dominates(pr, "b", "a")
## [1] FALSE

# calculate cumulative scores
scores <- cumulativeScores(pr)
# show score of element a
scores$a
## [1] 1 2 3 3 3

# performing a bunch of rankings
lexcelRanking(pr)
## a > b > c

dualLexcelRanking(pr)
## a > c > b

copelandRanking(pr)
## a > b ~ c

kramerSimpsonRanking(pr)
## a > b ~ c

ordinalBanzhafRanking(pr)
## a ~ c > b
```

Finally an incidence matrix for all given coalitions can be constructed using `powerRelationMatrix(pr)` or `as.relation(pr)` from the `relations` package [11]. The incidence matrix may be displayed using `relations::relation_incidence()`.

```
rel <- relations::as.relation(pr)
rel
## A binary relation of size 7 x 7.
```

```
relations::relation_incidence(rel)
## Incidences:
##    ab ac bc a c {} b
## ab  1  1  1 1 1  1 1
## ac  0  1  1 1 1  1 1
## bc  0  1  1 1 1  1 1
## a   0  0  0 1 1  1 1
## c   0  0  0 1 1  1 1
## {}  0  0  0 0 0  1 1
## b   0  0  0 0 0  0 1
```

2 `PowerRelation` Objects

We first introduce some basic definitions on binary relations. Let X be a set. A set $R \subseteq X \times X$ is said a *binary relation* on X. For two elements $x, y \in X$, xRy refers to their relation, more formally it means that $(x, y) \in R$. A binary relation $(x, y) \in R$ is said to be:

- *reflexive*, if for each $x \in X, xRx$
- *transitive*, if for each $x, y, z \in X, xRy$ and $yRz \Rightarrow xRz$
- *total*, if for each $x, y \in X, x \neq y \Rightarrow xRy$ or yRx
- *symmetric*, if for each $x, y \in X, xRy \Leftrightarrow yRx$
- *asymmetric*, if for each $x, y \in X, (x, y) \in R \Rightarrow (y, x) \notin R$
- *antisymmetric*, if for each $x, y \in X, xRy \cap yRx \Rightarrow x = y$

A *preorder* is defined as a reflexive and transitive relation. If it is total, it is called a *total preorder*. Additionally if it is antisymmetric, it is called a *linear order*.

Let $N = \{1, 2, \dots, n\}$ be a finite set of *elements*, sometimes also called *players*. For some $p \in \{1, \dots, 2^n\}$, let $\mathcal{P} = \{S_1, S_2, \dots, S_p\}$ be a set of *coalitions* such that $S_i \subseteq N$ for all $i \in \{1, \dots, p\}$. Thus $\mathcal{P} \subseteq 2^N$, where 2^N denotes the power set of N (i.e., the set of all subsets or coalitions of N).

$\mathcal{T}(N)$ denotes the set of all total preorders on N, $\mathcal{T}(\mathcal{P})$ the set of all total preorders on $\mathcal{P}$. A single total preorder $\succeq \in \mathcal{T}(\mathcal{P})$ is said a *power relation*.

In a given power relation $\succeq \in \mathcal{T}(\mathcal{P})$ on $\mathcal{P} \subseteq 2^N$, its symmetric part is denoted by $\sim$ (i.e., $S \sim T$ if $S \succeq T$ and $T \succeq S$), whereas its asymmetric part is denoted by $\succ$ (i.e., $S \succ T$ if $S \succeq T$ and not $T \succeq S$). In other terms, for $S \sim T$ we say that S is *indifferent* to T, whereas for $S \succ T$ we say that S is *strictly better* than T.

Lastly for a given power relation in the form of $S_1 \succeq S_2 \succeq \dots \succeq S_m$, coalitions that are indifferent to one another can be grouped into *equivalence classes* $\sum_i$ such that we get the *quotient order* $\sum_1 \succ \sum_2 \succ \dots \succ \sum_m$.

Example 1. Let $N = \{1, 2\}$ be two players with its corresponding power set $2^N = \{\{1, 2\}, \{1\}, \{2\}, \emptyset\}$. The following power relation is given: $\succeq =$

$\{(\{1,2\},\{2\}),(\{2\},\emptyset),(\emptyset,\{2\}),(\emptyset,\{1\})\}$. This power relation can be rewritten in a consecutive order as: $\{1,2\} \succ \{2\} \sim \emptyset \succ \{1\}$. Its quotient order is formed by three equivalence classes $\sum_1 = \{\{1,2\}\}, \sum_2 = \{\{2\},\emptyset\}$, and $\sum_3 = \{\{1\}\}$; so the quotient order of $\succeq$ is such that $\{\{1,2\}\} \succ \{\{2\},\emptyset\} \succ \{\{1\}\}$.

A *social ranking solution* (also called *social ranking* or, simply, *solution*) on N, is a function $R : \mathcal{T}(\mathcal{P}) \longrightarrow \mathcal{T}(N)$ associating to each power relation $\succeq \in \mathcal{T}(\mathcal{P})$ a total preorder $R(\succeq)$ (or $R^{\succeq}$) over the elements of N. By this definition, the notion $iR^{\succeq}j$ means that applying the social ranking solution to the power relation $\succeq$ gives the result that i is ranked higher than or equal to j.

2.1 Creating `PowerRelation` Objects

A power relation in the `socialranking` package is defined to be reflexive, transitive and total. In designing the package it was deemed logical to have the coalitions specified in a consecutive order, as seen in Example 1. Each coalition in that order is split either by a `">"` (left side strictly better) or a `"~"` (two coalitions indifferent to one another). The following code chunk shows the power relation from Example 1 and how a correlating `PowerRelation` object can be constructed.

```
library(socialranking)
pr <- newPowerRelation(c(1,2), ">", 2, "~", c(), ">", 1)
pr
## Elements: 1 2
## 12 > (2 ~ {}) > 1
```

```
class(pr)
## [1] "PowerRelation"      "SingleCharElements"
```

Notice how coalitions such as $\{1,2\}$ are written as `12` to improve readability. Similarly the function `newPowerRelationFromString` saves some typing on the user's end by interpreting each character of a coalition as a separate player. Note that spaces in that function are ignored.

```
newPowerRelationFromString("12 > 2~{} > 1", asWhat = as.numeric)
## Elements: 1 2
## 12 > (2 ~ {}) > 1
```

The compact notation is only done in `PowerRelation` objects where every player is one digit or one character long. If this is not the case, curly braces and commas are added where needed.

```
prLong <- newPowerRelation(
  c("Alice", "Bob"), ">", "Bob", "~", c(), ">", "Alice"
)
prLong
## Elements: Alice Bob
## {Alice, Bob} > ({Bob} ~ {}) > {Alice}
```

```
class(prLong)
## [1] "PowerRelation"
```

Attribute	Description	Value in `pr`
`elements`	Sorted vector of elements	`c(1,2)`
`rankingCoalitions`	Coalitions in power relation	`list(set(1,2),set(2),set(),set(1))`
`equivalenceClasses`	List containing lists, each containing coalitions in the same equivalence class	`list(list(set(1,2)), list(set(2), set()), list(set(1)))`

Some may have spotted a `"SingleCharElements"` class missing in `class(prLong)` that has been there in `class(pr)`. `"SingleCharElements"` influences the way coalitions are printed. If it is removed from `class(pr)`, the output will include the same curly braces and commas displayed in `prLong`.

```
class(pr) <- class(pr)[-which(class(pr) == "SingleCharElements")]
pr
## Elements: 1 2
## {1, 2} > ({2} ~ {}) > {1}
```

Internally a `PowerRelation` is a list with four attributes (see table below). Notice that every coalition vector is turned into a `set` object from the `sets` package[12].

Since each coalition vector is turned into a `set`, coalitions such as `c(1,2)`, `c(2,1)` and `c(1,1,2,2)` are equivalent.

```
prAtts <- newPowerRelation(c(2,2,1,1,2), ">", c(1,1,1), "~", c())
prAtts
## Elements: 1 2
## 12 > (1 ~ {})
```

```
prAtts$elements
## [1] 1 2
```

```
prAtts$rankingCoalitions
## [[1]]
## {1, 2}
##
## [[2]]
## {1}
##
## [[3]]
## {}
```

```
prAtts$rankingComparators
## [1] ">" "~"
```

```
prAtts$equivalenceClasses
## [[1]]
## [[1]][[1]]
## {1, 2}
##
##
```

```
## [[2]]
## [[2]][[1]]
## {1}
##
## [[2]][[2]]
## {}
```

`equivalenceClassIndex()` determines at which index i a coalition $S \in \sum_i$.

```
equivalenceClassIndex(prAtts, c(2,1))
## [1] 1
```

```
equivalenceClassIndex(prAtts, 1)
## [1] 2
```

```
equivalenceClassIndex(prAtts, c())
## [1] 2
```

```
# are the given coalitions in the same equivalence class?
coalitionsAreIndifferent(prAtts, 1, c())
## [1] TRUE
```

```
coalitionsAreIndifferent(prAtts, 1, c(1,2))
## [1] FALSE
```

2.2 Manipulating `PowerRelation` Objects

It is strongly discouraged to directly manipulate `PowerRelation` objects, since modifying one list or vector entry would require updates on all attributes. Instead `newPowerRelation` offers two parameters `rankingCoalitions` and `rankingComparators`, each corresponding to the same named attributes of a `PowerRelation` object.

```
pr
## Elements: 1 2
## {1, 2} > ({2} ~ {}) > {1}
```

```
# reverse power ranking
newPowerRelation(
  rankingCoalitions = rev(pr$rankingCoalitions),
  rankingComparators = pr$rankingComparators
)
## Elements: 1 2
## 1 > ({} ~ 2) > 12
```

Note that `rankingComparators` is optional. By default it assumes `rankingCoalitions` to be a linear order.

```
newPowerRelation(rankingCoalitions = rev(pr$rankingCoalitions))
## Elements: 1 2
## 1 > {} > 2 > 12
```

If the length of the `rankingComparators` vector is smaller or larger than the length of `rankingCoalitions`, the function silently fills in any gaps.

```
# if too short -> comparator values are repeated
newPowerRelation(
  rankingCoalitions = as.list(1:9),
  rankingComparators = "~"
)
## Elements: 1 2 3 4 5 6 7 8 9
## (1 ~ 2 ~ 3 ~ 4 ~ 5 ~ 6 ~ 7 ~ 8 ~ 9)
```

```
newPowerRelation(
    rankingCoalitions = as.list(letters[1:9]),
    rankingComparators = c(">", "~", "~")
)
## Elements: a b c d e f g h i
## a > (b ~ c ~ d) > (e ~ f ~ g) > (h ~ i)
```

```
# if too long -> ignore excessive comparators
newPowerRelation(
  rankingCoalitions = pr$rankingCoalitions,
  rankingComparators = c("~", ">", "~", ">", ">", "~")
)
## Elements: 1 2
## (12 ~ 2) > ({} ~ 1)
```

2.3 Creating Power sets

As the number of elements n increases, the number of possible coalitions increases to $|2^N| = 2^n$. `createPowerset` is a convenient function that not only creates a power set 2^N which can be used to call `newPowerRelation`, but also formats the function call in such a way that makes it easy to rearrange the ordering of the coalitions. RStudio offers shortcuts Alt+Up or Alt+Down (Option+Up or Option+Down on MacOS) to move one or multiple lines of code up or down (see fig. 1).

```
createPowerset(
  c("a", "b", "c"),
  writeLines = TRUE,
  copyToClipboard = FALSE
)
## newPowerRelation(
##   c("a", "b", "c"),
##   ">", c("a", "b"),
##   ">", c("a", "c"),
##   ">", c("b", "c"),
##   ">", c("a"),
##   ">", c("b"),
##   ">", c("c"),
##   ">", c(),
## )
```

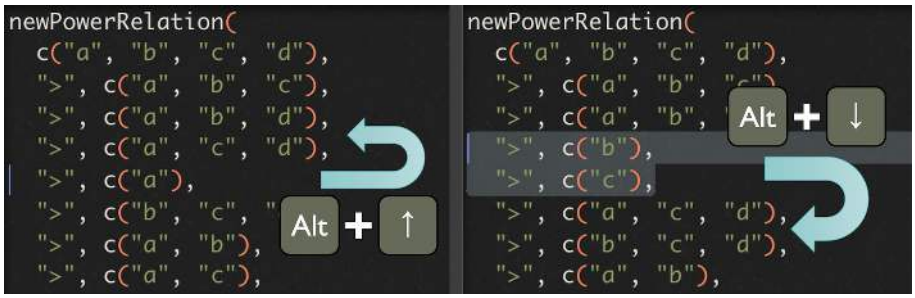

Figure 1: Using Alt+Up or Alt+Down to move one or more lines of code

If `writeLines` and `copyToClipboard` are both set to `FALSE`, the function instead returns a list only containing coalition vectors. This list can be passed directly as `rankingCoalitions` parameter to `newPowerRelation`.

```
ps <- createPowerset(1:2, includeEmptySet = FALSE)
ps
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 2
```

```
newPowerRelation(rankingCoalitions = ps)
## Elements: 1 2
## 12 > 1 > 2
```

```
newPowerRelation(rankingCoalitions = createPowerset(letters[1:4]))
## Elements: a b c d
## abcd > abc > abd > acd > bcd > ab > ac > ad > bc > bd > cd > a > b
> c > d > {}
```

3 `SocialRankingSolution` Objects

The main goal of the `socialranking` package is to rank elements based on a given power ranking. More formally we try to map $R : \mathcal{T}(\mathcal{P}) \to \mathcal{T}(N)$, associating to each power relation $\succeq \in \mathcal{T}(\mathcal{P})$ a total preorder $R(\succeq)$ (or $R^{\succeq}$) over the elements of N.

In this context $iR^{\succeq}j$ tells us that, given a power relation $\succeq$ and applying a social ranking solution $R(\succeq)$, i is ranked higher than or equal to j. From here on out, `>` and `~` also denote the asymmetric and the symmetric part of a social ranking, respectively, i `>` j indicating that i is strictly better than j, whereas in i `~` j, i is indifferent to j.

In section 3.1 we show how a general `SocialRankingSolution` object can be constructed using the `doRanking` function. In the following sections, we will introduce the notion of dominance[4], cumulative dominance[13] and CP-Majority comparison[6] that lets us

compare two elements before diving into the social ranking solutions of the Ordinal Banzhaf Index[5], Copeland-like and Kramer-Simpson-like methods[10], and lastly the Lexicographical Excellence Solution[9] (Lexcel) and the Dual Lexicographical Excellence solution[14] (Dual Lexcel).

Example 2. Let $\{a, b\} \succ (\{a, c\} \sim \{b, c\}) \succ (\{a\} \sim \{c\}) > \emptyset \succ \{b\}$ be a power ranking. Using the following social ranking solutions, we get:

- `a > b > c` for `lexcelRanking`
- `a > c > b` for `dualLexcelRanking`
- `a > b ~ c` for `copelandRanking` and `kramerSimpsonRanking`
- `a ~ c > b` for `ordinalBanzhafRanking`

3.1 Creating `SocialRankingSolution` objects

A `SocialRankingSolution` object represents a total preorder in $\mathcal{T}(N)$ over the elements of N. Internally they are saved as a list of vectors, each containing players that are indifferent to one another. This is somewhat similar to the `equivalenceClasses` attribute in `PowerRelation` objects.

The function `doRanking` offers a generic way of creating `SocialRankingSolution` objects. Given a `PowerRelation` object and a sortable vector or list of scores it determines the power relation between all players. Note that `length(scores) == length(powerRelation$elements)` must be `TRUE`. Additionally each index in `scores` corresponds to the index in the sorted vector `powerRelation$elements`.

```
pr <- newPowerRelationFromString("abc > ab ~ ac > bc")

# pr$elements == c("a", "b", "c")
# we define some arbitrary score vector where "a" scores highest
# "b" and "c" both score 1, thus they are indifferent
scores <- c(100, 1, 1)
doRanking(pr, scores)
## a > b ~ c
```

```
# we can also tell doRanking to punish higher scores
doRanking(pr, scores, decreasing = FALSE)
## b ~ c > a
```

By default elements are assumed to be indifferent to one another if their scores are equal. Sometimes however other factors come into play that make it non-obvious how to compare two scores. In those cases a function comparing two scores can be passed that should return `TRUE` if the two scores are considered equal.

```
scores <- c(0, 20, 21)
# b and c are considered to be indifferent,
# because their score difference is less than 2
doRanking(pr, scores, isIndifferent = function(a,b) abs(a-b) < 2)
## b ~ c > a
```

3.2 Comparison Functions

Comparison functions only compare two elements in a given power relation. They do not offer a social ranking solution. However in cases such as CP-Majority comparison, those comparison functions may be used to construct a social ranking solution in some particular cases.

3.2.1 Dominance

Definition 1. (Dominance [4]) Given a power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and two elements $i, j \in N$, i *dominates* j in $\succeq$ if $S \cup \{i\} \succeq S \cup \{j\}$ for each $S \in 2^{N \setminus \{i,j\}}$. i also *strictly* dominates j if there exists $S \in 2^{N \setminus \{i,j\}}$ such that $S \cup \{i\} \succ S \cup \{j\}$.

The implication is that for every coalition i and j can join, i has at least the same positive impact as j.

The function `dominates(pr, e1, e2)` only returns a logical value `TRUE` if `e1` dominates `e2`, else `FALSE`. Note that `e1` not dominating `e2` does *not* indicate that `e2` dominates `e1`, nor does it imply that `e1` is indifferent to `e2`.

```
pr <- newPowerRelationFromString(
  "3 > 1 > 2 > 12 > 13 > 23",
  asWhat = as.numeric
)

# 1 clearly dominates 2
dominates(pr, 1, 2)
## [1] TRUE
```

```
dominates(pr, 2, 1)
## [1] FALSE
```

```
# 3 does not dominate 1, nor does 1 dominate 3, because
# {}u3 > {}u1, but 2u1 > 2u3
dominates(pr, 1, 3)
## [1] FALSE
```

```
dominates(pr, 3, 1)
## [1] FALSE
```

```
# an element i dominates itself, but it does not strictly dominate itself
# because there is no Sui > Sui
dominates(pr, 1, 1)
## [1] TRUE
```

```
dominates(pr, 1, 1, strictly = TRUE)
## [1] FALSE
```

For any $S \in 2^{N \setminus \{i,j\}}$, we can only compare $S \cup \{i\} \succeq S \cup \{j\}$ if both $S \cup \{i\}$ and $S \cup \{j\}$ take part in the power relation.

Additionally, for $S = \emptyset$, we also want to compare $\{i\} \succeq \{j\}$. In some situations however a comparison between singletons is not desired. For this reason the parameter `includeEmptySet` can be set to `FALSE` such that $\emptyset \cup \{i\} \succeq \emptyset \cup \{j\}$ is not considered in the CP-Majority comparison.

```
pr <- newPowerRelationFromString("ac > bc ~ b > a ~ abc > ab")

# FALSE because ac > bc, whereas b > a
dominates(pr, "a", "b")
## [1] FALSE
```

```
# TRUE because ac > bc, ignoring b > a comparison
dominates(pr, "a", "b", includeEmptySet = FALSE)
## [1] TRUE
```

3.2.2 Cumulative Dominance

When comparing two players $i, j \in N$, instead of looking at particular coalitions $S \in 2^{N \setminus \{i,j\}}$ they can join, we look at how many stronger coalitions they can form at each point. This property was originally introduced in [13] as a regular dominance axiom.

For a given power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and its corresponding quotient order $\sum_1 \succ \cdots \succ \sum_m$, the power of a player i is given by a vector $\text{Score}_{\text{Cumul}}(i) \in \mathbb{N}^m$ where we cumulatively sum the amount of times i appears in $\sum_k$ for each index k.

Definition 2. (Cumulative Dominance Score) Given a power relation $\succeq \in \mathcal{T}(\mathcal{P})$ and its quotient order $\sum_1 \succ \cdots \succ \sum_m$, the cumulative score vector $\text{Score}_{\text{Cumul}}(i) \in \mathbb{N}^m$ of an element $i \in N$ is given by:

$$\text{Score}_{\text{Cumul}}(i) = \Big(\sum_{t=1}^{k} |\{S \in \textstyle\sum_t : i \in S\}| \Big)_{k \in \{1,\dots,m\}} \tag{1}$$

Definition 3. (Cumulative Dominance) Given two elements $i, j \in N$, i *cumulatively dominates* j in $\succeq$, if $\text{Score}_{\text{Cumul}}(i)_k \geq \text{Score}_{\text{Cumul}}(j)_k$ for each $k \in \{1, \dots, m\}$. i also *strictly* cumulatively dominates j if there exists a k such that $\text{Score}_{\text{Cumul}}(i)_k > \text{Score}_{\text{Cumul}}(j)_k$.

For a given `PowerRelation` object `pr` and two elements `e1` and `e2`, `cumulativeScores(pr)` returns the vectors described in definition 2 for each element, `cumulativelyDominates(pr, e1, e2)` returns `TRUE` or `FALSE` based on definition 3.

```
pr <- newPowerRelationFromString("ab > (ac ~ bc) > (a ~ c) > {} > b")
cumulativeScores(pr)
## $a
## [1] 1 2 3 3 3
##
## $b
## [1] 1 2 2 2 3
```

```
##
## $c
## [1] 0 2 3 3 3
##
## attr(,"class")
## [1] "CumulativeScores"
```

```
# for each index k, $a[k] >= $b[k]
cumulativelyDominates(pr, "a", "b")
```
```
## [1] TRUE
```

```
# $a[3] > $b[3], therefore a also strictly dominates b
cumulativelyDominates(pr, "a", "b", strictly = TRUE)
```
```
## [1] TRUE
```

```
# $b[1] > $c[1], but $c[3] > $b[3]
# therefore neither b nor c dominate each other
cumulativelyDominates(pr, "b", "c")
```
```
## [1] FALSE
```

```
cumulativelyDominates(pr, "c", "b")
```
```
## [1] FALSE
```

Similar to the dominance property from our previous section, two elements not dominating one or the other does not indicate that they are indifferent.

3.2.3 CP-Majority comparison

The Ceteris Paribus Majority (CP-Majority) relation is a somewhat relaxed version of the dominance property. Instead of checking if $S \cup \{i\} \succeq S \cup \{j\}$ for all $S \in 2^{N \setminus \{i,j\}}$, the CP-Majority relation $iR^{\succeq}_{\text{CP}}j$ holds if the number of times $S \cup \{i\} \succeq S \cup \{j\}$ is greater than or equal to the number of times $S \cup \{j\} \succeq S \cup \{i\}$.

Definition 4. (CP-Majority [6]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Ceteris Paribus majority* relation is the binary relation $R^{\succeq}_{\text{CP}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{CP}}j \Leftrightarrow d_{ij}(\succeq) \geq d_{ji}(\succeq) \tag{2}$$

where $d_{ij}(\succeq)$ represents the cardinality of the set $D_{ij}(\succeq)$, the set of all coalitions $S \in 2^{N \setminus \{i,j\}}$ for which $S \cup \{i\} \succeq S \cup \{j\}$.

`cpMajorityComparisonScore(pr, e1, e2)` calculates the two scores $d_{ij}(\succeq)$ and $-d_{ji}(\succeq)$. Notice the minus sign - that way we can use the sum of both values to determine the relation between `e1` and `e2`.

```
pr <- newPowerRelationFromString("ab > (ac ~ bc) > (a ~ c) > {} > b")
cpMajorityComparisonScore(pr, "a", "b")
```
```
## [1]  2 -1
```

```
cpMajorityComparisonScore(pr, "b", "a")
## [1]  1 -2
```

```
if(sum(cpMajorityComparisonScore(pr, "a", "b")) >= 0) {
  print("a >= b")
} else {
  print("b > a")
}
## [1] "a >= b"
```

As a slight variation the logical parameter `strictly` calculates $d_{ij}(\succ)$ and $-d_{ji}(\succ)$, the number of coalitions $S \in 2^{N\setminus\{i,j\}}$ where $S \cup \{i\} \succ S \cup \{j\}$.

```
# Now (ac ~ bc) is not counted
cpMajorityComparisonScore(pr, "a", "b", strictly = TRUE)
## [1] 1 0
```

```
# Notice that the sum is still the same
sum(cpMajorityComparisonScore(pr, "a", "b", strictly = FALSE)) ==
  sum(cpMajorityComparisonScore(pr, "a", "b", strictly = TRUE))
## [1] TRUE
```

Coincidentally, `cpMajorityComparisonScore` with `strictly = TRUE` can be used to determine if `e1` (strictly) dominates `e2`.

`cpMajorityComparisonScore` should be used for simple and quick calculations. The more comprehensive function `cpMajorityComparison(pr, e1, e2)` does the same calculations, but in the process retains more information about all the comparisons that might be interesting to a user, i.e., the set $D_{ij}(\succeq)$ and $D_{ji}(\succeq)$ as well as the relation $iR^{\succeq}_{\text{CP}}j$. See the documentation for a full list of available data.

```
# extract more information in cpMajorityComparison
cpMajorityComparison(pr, "a", "b")
## a > b
## D_ab = {c, {}}
## D_ba = {c}
## Score of a = 2
## Score of b = 1
```

```
# with strictly set to TRUE, coalition c does
# neither appear in D_ab nor in D_ba
cpMajorityComparison(pr, "a", "b", strictly = TRUE)
## a > b
## D_ab = {{}}
## D_ba = {}
## Score of a = 1
## Score of b = 0
```

The CP-Majority relation can generate cycles, which is the reason that it is not offered as a social ranking solution. Instead we will introduce the Copeland-like method and Kramer-Simpson-like method in chapters 3.3.2 and 3.3.3 that make use of the CP-Majority functions to determine a power relation between elements. For further readings on CP-Majority, see [7] and [10].

3.3 Social Ranking Solutions

3.3.1 Ordinal Banzhaf

The Ordinal Banzhaf Score is a vector defined by the principle of marginal contributions. Intuitively speaking, if a player joining a coalition causes it to move up in the ranking, it can be interpreted as a positive contribution. On the contrary a negative contribution means that participating causes the coalition to go down in the ranking.

Definition 5. (Ordinal marginal contribution [5]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. For a given element $i \in N$, its *ordinal marginal contribution* $m_i^S(\succeq)$ with right to a coalition $S \in \mathcal{P}$ is defined as:

$$m_i^S(\succeq) = \begin{cases} 1 & \text{if } S \cup \{i\} \succ S \\ -1 & \text{if } S \succ S \cup \{i\} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Definition 6. (Ordinal Banzhaf relation) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Ordinal Banzhaf relation* is the binary relation $R_{\text{Banz}}^{\succeq} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR_{\text{Banz}}^{\succeq}j \Leftrightarrow \text{Score}_{\text{Banz}}(i) \geq \text{Score}_{\text{Banz}}(j), \tag{4}$$

where $\text{Score}_{\text{Banz}}(i) = \sum_S m_i^S(\succeq)$ for all $S \in N \quad \{i\}$.

Note that if $S \cup \{i\} \notin \mathcal{P}$, $m_i^S(\succeq) = 0$.

The function `ordinalBanzhafScores` returns two numbers for each element: the number of coalitions S where a player's contribution has a positive impact, and the number of coalitions S where a player's contribution has a negative impact. These two numbers are added and elements are ranked highest to lowest.

```
pr <- newPowerRelation(
  c(1,2),
  ">", c(1),
  ">", c(2)
)

# both players 1 and 2 have an Ordinal Banzhaf Score of 1
# therefore they are indifferent to one another
ordinalBanzhafScores(pr)
```

```
## $`1`
## [1] 1 0
##
## $`2`
## [1] 1 0
##
## attr(,"class")
## [1] "OrdinalBanzhafScores"
```

```
ordinalBanzhafRanking(pr)
```

```
## 1 ~ 2
```

```
pr <- newPowerRelationFromString("ab > a > {} > b")

# player b has a negative impact on the empty set
# -> player b's score is 1 - 1 = 0
# -> player a's score is 2 - 0 = 2
sapply(ordinalBanzhafScores(pr), function(score) sum(score))
## a b
## 2 0
```

```
ordinalBanzhafRanking(pr)
## a > b
```

3.3.2 Copeland-like method

The Copeland-like method of ranking elements based on the CP-Majority rule is strongly inspired by the Copeland score from social choice theory[15]. The score of an element $i \in N$ is determined by the amount of the pairwise CP-Majority winning comparisons $iR^{\succeq}_{\text{CP}}j$, minus the number of all losing comparisons $jR^{\succeq}_{\text{CP}}i$ against all other elements $j \in N \setminus \{i\}$.

Definition 7. (Copeland-like relation [10]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Copeland-like* relation is the binary relation $R^{\succeq}_{\text{Cop}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{Cop}}j \Leftrightarrow \text{Score}_{\text{Cop}}(i) \geq \text{Score}_{\text{Cop}}(j), \tag{5}$$

where $\text{Score}_{\text{Cop}}(i) = |\{j \in N \setminus \{i\} : d_{ij}(\succeq) \geq d_{ji}(\succeq)\}| - |\{j \in N \setminus \{i\} : d_{ij}(\succeq) \leq d_{ji}(\succeq)\}|$

`copelandScores(pr)` returns two numerical values for each element, a positive number for the winning comparisons (shown in $\text{Score}_{\text{Cop}}(i)$ on the left) and a negative number for the losing comparisons (in $\text{Score}_{\text{Cop}}(i)$ on the right).

```
pr <- newPowerRelationFromString("(abc ~ ab ~ c ~ a) > (b ~ bc) > ac")
scores <- copelandScores(pr)

# Based on CP-Majority, a>=b and a>=c (+2), but b>=a (-1)
scores$a
## [1]  2 -1
```

```
sapply(copelandScores(pr), sum)
##  a  b  c
##  1  0 -1
```

```
copelandRanking(pr)
## a > b > c
```

3.3.3 Kramer-Simpson-like method

Strongly inspired by the Kramer-Simpson method of social choice theory[16, 17], elements are ranked inversely to their greatest pairwise defeat over all possible CP-Majority comparisons.

Definition 8. (Kramer-Simpson-like relation [10]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Kramer-Simpson-like* relation is the binary relation $R_{\mathrm{KS}}^{\succeq} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR_{\mathrm{KS}}^{\succeq}j \Leftrightarrow \mathrm{Score}_{\mathrm{KS}}(i) \leq \mathrm{Score}_{\mathrm{KS}}(j), \tag{6}$$

where $\mathrm{Score}_{\mathrm{KS}}(i) = \max_j d_{ji}(\succeq)$ for all $j \in N \quad \{i\}$.

`kramerSimpsonScores(pr)` returns a single numerical value for each element, which sorted lowest to highest gives us the ranking solution.

```
pr <- newPowerRelationFromString("(abc ~ ab ~ c ~ a) > (b ~ bc) > ac")
unlist(kramerSimpsonScores(pr))
```

```
## a b c
## 0 0 1
```

```
kramerSimpsonRanking(pr)
```

```
## a ~ b > c
```

There is a small caveat to Definition 8. By default this function does not compare $d_{ii}(\succeq)$. In other terms, the score of every element is the maximum CP-Majority comparison score against all *other* elements.

This is slightly different from the definition found in [10], where the CP-Majority comparison $d_{ii}(\succeq)$ is also considered. Since by definition $d_{ii}(\succeq) = 0$, the Kramer-Simpson scores in those cases will never be negative, possibly discarding valuable information.

To still account for the original definition in [10], the functions `kramerSimpsonScores` and `kramerSimpsonRanking` offer a `compIvsI` parameter that can be set to `TRUE` if one wishes for $d_{ii}(\succeq)$ to be included in the comparisons.

```
pr <- newPowerRelationFromString(
  "b > (a ~ c) > ab > (ac ~ bc) > {} > abc"
)
kramerSimpsonRanking(pr)
```

```
## b > a > c
```

```
# notice how b's score is negative
unlist(kramerSimpsonScores(pr))
```

```
##  a  b  c
##  1 -1  2
```

```
kramerSimpsonScores(pr, elements = "b", compIvsI = TRUE)
```

```
## $b
## [1] 0
##
## attr(,"class")
## [1] "KramerSimpsonScores"
```

3.3.4 Lexicographical Excellence Solution

The idea behind the lexicographical excellence solution (Lexcel) is to reward elements appearing more frequently in higher ranked equivalence classes.

For a given power relation $\succeq$ and its quotient order $\sum_1 \succ \cdots \succ \sum_m$, we denote by i_k the number of coalitions in $\sum_k$ containing i:

$$i_k = |\{S \in \sum_k : i \in S\}| \tag{7}$$

for $k \in \{1, \ldots, m\}$. Now, let $\text{Score}_{\text{Lex}}(i)$ be the m-dimensional vector $\text{Score}_{\text{Lex}}(i) = (i_1, \ldots, i_m)$ associated to $\succeq$. Consider the lexicographic order $\geq_{\text{Lex}}$ among vectors $\mathbf{i}$ and $\mathbf{j}$: $\mathbf{i} \geq_{\text{Lex}} \mathbf{j}$ if either $\mathbf{i} = \mathbf{j}$ or there exists $t : i_r = j_r, r \in \{1, \ldots, t-1\}$, and $i_t > j_t$.

Definition 9. (Lexicographic-Excellence relation [8]) Let $\succeq \in \mathcal{T}(\mathcal{P})$ with its corresponding quotient order $\sum_1 \succ \cdots \succ \sum_m$. The *Lexicographic-Excellence* relation is the binary relation $R^{\succeq}_{\text{Lex}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{Lex}}j \Leftrightarrow \text{Score}_{\text{Lex}}(i) \geq_{\text{Lex}} \text{Score}_{\text{Lex}}(j) \tag{8}$$

```
pr <- newPowerRelationFromString(
  "12 > (123 ~ 23 ~ 3) > (1 ~ 2) > 13",
  asWhat = as.numeric
)

# show the number of times an element appears in each equivalence class
# e.g. 3 appears 3 times in [[2]] and 1 time in [[4]]
lapply(pr$equivalenceClasses, unlist)
```

```
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 1 2 3 2 3 3
##
## [[3]]
## [1] 1 2
##
## [[4]]
## [1] 1 3
```

```
lexScores <- lexcelScores(pr)
for(i in names(lexScores))
  paste0("Lexcel score of element ", i, ": ", lexScores[i])

# at index 1, element 2 ranks higher than 3
lexScores['2'] > lexScores['3']
```

```
## [1] TRUE
```

```
# at index 2, element 2 ranks higher than 1
lexScores['2'] > lexScores['1']
## [1] TRUE
```

```
lexcelRanking(pr)
## 2 > 1 > 3
```

For some generalizations of the Lexcel solution see also [9].

Lexcel score vectors are very similar to the cumulative score vectors (3.2.2) in that the number of times an element appears in a given equivalence class is of interest. In fact, applying the base function `cumsum` on an element's lexcel score gives us its cumulative score.

```
lexcelCumulated <- lapply(lexScores, cumsum)
cumulScores <- cumulativeScores(pr)

paste0(names(lexcelCumulated), ": ", lexcelCumulated, collapse = ', ')
## [1] "1: 1:4, 2: c(1, 3, 4, 4), 3: c(0, 3, 3, 4)"
```

```
paste0(names(cumulScores), ": ", cumulScores, collapse = ', ')
## [1] "1: 1:4, 2: c(1, 3, 4, 4), 3: c(0, 3, 3, 4)"
```

3.3.5 Dual Lexicographical Excellence Solution

Similar to the Lexcel ranking, the Dual Lexcel also uses the Lexcel score vectors from definition 9 to establish a ranking. However, instead of rewarding higher frequencies in high ranking coalitions, it punishes higher frequencies in lower ranking coalitions, or, it punishes mediocrity[14].

Take the values i_k for $k \in \{1, \dots, m\}$ and the Lexcel score vector $\text{Score}_{\text{Lex}}(i)$ from section 3.3.4. Consider the dual lexicographical order $\geq_{\text{DualLex}}$ among vectors $\mathbf{i}$ and $\mathbf{j}$: $\mathbf{i} \geq_{\text{DualLex}} \mathbf{j}$ if either $\mathbf{i} = \mathbf{j}$ or there exists $t : i_t < j_t$ and $i_r = j_r, r \in \{t+1, \dots, m\}$.

Definition 10. (Dual Lexicographical-Excellence relation [14]) Let $\succeq \in \mathcal{T}(\mathcal{P})$. The *Dual Lexicographic-Excellence* relation is the binary relation $R^{\succeq}_{\text{DualLex}} \subseteq N \times N$ such that for all $i, j \in N$:

$$iR^{\succeq}_{\text{DualLex}}j \Leftrightarrow \text{Score}_{\text{Lex}}(i) \geq_{\text{DualLex}} \text{Score}_{\text{Lex}}(j) \tag{9}$$

The S3 class `LexcelScores` does not account for Dual Lexcel comparisons. Instead `-rev(x)` is called on a Lexcel score vector `x` such that the resulting comparisons produces a Dual Lexcel ranking solution.

```
pr <- newPowerRelationFromString(
  "12 > (123 ~ 23 ~ 3) > (1 ~ 2) > 13",
  asWhat = as.numeric
)

lexScores <- lexcelScores(pr)
```

```
# in regular Lexcel, 1 scores higher than 3
lexScores['1'] > lexScores['3']
## [1] TRUE
```

```
# turn Lexcel score into Dual Lexcel score
dualLexScores <- structure(
  lapply(lexcelScores(pr), function(r) -rev(r)),
  class = 'LexcelScores'
)

# now 1 scores lower than 3
dualLexScores['1'] > dualLexScores['3']
## [1] FALSE
```

```
# element 2 comes out at the top in both Lexcel and Dual Lexcel
lexcelRanking(pr)
## 2 > 1 > 3
```

```
dualLexcelRanking(pr)
## 2 > 3 > 1
```

4 Relations

4.1 Incidence Matrix

In our vignette we focused more on the intuitive aspects of power relations and social ranking solutions. To reiterate, a power relation is a total preorder, or a reflexive and transitive relation $\succeq \in \mathcal{P} \times \mathcal{P}$, where $\sim$ denotes the symmetric part and $\succ$ its asymmetric part.

A power relation can be viewed as an incidence matrix $(b_{ij}) = B \in \{0,1\}^{|\mathcal{P}|\times|\mathcal{P}|}$. Given two coalitions $i, j \in \mathcal{P}$, if iRj then $b_{ij} = 1$, else 0.

With help of the `relations` package, the functions `relations::as.relation(pr)` and `powerRelationMatrix(pr)` turn a `PowerRelation` object into a `relation` object. `relations` then offers ways to display the `relation` object as an incidence matrix with `relation_incidence(rel)` and to test basic properties such `relation_is_linear_order(rel)`, `relation_is_acyclic(rel)` and `relation_is_antisymmetric(rel)` (see `relations` package for more [11]).

```
pr <- newPowerRelationFromString("ab > a > {} > b")
rel <- relations::as.relation(pr)

relations::relation_incidence(rel)
## Incidences:
##    ab a {} b
## ab  1 1  1 1
## a   0 1  1 1
## {}  0 0  1 1
## b   0 0  0 1
```

```
c(
  relations::relation_is_acyclic(rel),
  relations::relation_is_antisymmetric(rel),
  relations::relation_is_linear_order(rel),
  relations::relation_is_complete(rel),
  relations::relation_is_reflexive(rel),
  relations::relation_is_transitive(rel)
)
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Note that the columns and rows are sorted by their names in `relation_domain(rel)`, hence why each name is preceded by the ordering number.

```
# a power relation where coalitions {1} and {2} are indifferent
pr <- newPowerRelationFromString("12 > (1 ~ 2)", asWhat = as.numeric)
rel <- relations::as.relation(pr)

# we have both binary relations {1}R{2} as well as {2}R{1}
relations::relation_incidence(rel)
## Incidences:
##    12 1 2
## 12  1 1 1
## 1   0 1 1
## 2   0 1 1
```

```
# FALSE
c(
  relations::relation_is_acyclic(rel),
  relations::relation_is_antisymmetric(rel),
  relations::relation_is_linear_order(rel),
  relations::relation_is_complete(rel),
  relations::relation_is_reflexive(rel),
  relations::relation_is_transitive(rel)
)
## [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

4.2 Cycles and Transitive Closure

A cycle in a power relation exists, if there is one coalition $S \in 2^N$ that appears twice. For example, in $\{1,2\} \succ (\{1\} \sim \emptyset) \succ \{1,2\}$, the coalition $\{1,2\}$ appears at the beginning and at the end of the power relation.

Properly handling power relations and calculating social ranking solutions with cycles is somewhat ill-defined, hence a warning message is shown as soon as one is created.

```
newPowerRelation(c(1,2), ">", 2, ">", 1, "~", 2, ">", c(1,2))
#! Warning in newPowerRelation(c(1, 2), ">", 2, ">", 1, "~", 2, ">",
c(1, 2)): Found the following duplicates. Did you mean to introduce
cycles?
#!  {2}
#!  {1, 2}
```

```
## Elements: 1 2
## 12 > 2 > (1 ~ 2) > 12
```

Recall that a power relation is transitive, meaning for three coalitions $x, y, z \in 2^N$, if xRy and yRz, then xRz. If we introduce cycles, we pretty much introduce symmetry. Assume we have the power relation $x \succ y \succ x$. Then, even though xRy and yRx are defined as the asymmetric part of the power relation $\succeq$, together they form the symmetric power relation $x \sim y$.

`transitiveClosure(pr)` is a function that turns a power relation with cycles into one without one. In the process of removing duplicate coalitions, it turns all asymmectric relations within a cycle into symmetric relations.

```
pr <- suppressWarnings(newPowerRelation(1, '>', 2, '>', 1))
pr
```

```
## Elements: 1 2
## 1 > 2 > 1
```

```
transitiveClosure(pr)
```

```
## Elements: 1 2
## (1 ~ 2)
```

```
# two cycles, (1>3>1) and (2>23>2)
pr <- suppressWarnings(
  newPowerRelationFromString(
    "1 > 3 > 1 > 2 > 23 > 2",
    asWhat = as.numeric
  )
)

transitiveClosure(pr)
```

```
## Elements: 1 2 3
## (1 ~ 3) > (2 ~ 23)
```

```
# overlapping cycles
pr <- suppressWarnings(
  newPowerRelationFromString("c > ac > b > ac > (a ~ b) > abc")
)

transitiveClosure(pr)
```

```
## Elements: a b c
## c > (ac ~ b ~ a) > abc
```

Bibliography

1. Shapley LS, Shubik M. A method for evaluating the distribution of power in a committee system. American political science review. 1954;48:787–92.
2. Banzhaf III JF. Weighted voting doesn't work: A mathematical analysis. Rutgers L Rev. 1964;19:317.
3. Holler MJ, Nurmi H. Power, voting, and voting power: 30 years after. Springer; 2013.

4. Moretti S, Öztürk M. Some axiomatic and algorithmic perspectives on the social ranking problem. In: International conference on algorithmic DecisionTheory. Springer; 2017. p. 166–81.
5. Khani H, Moretti S, Öztürk M. An ordinal banzhaf index for social ranking. In: 28th international joint conference on artificial intelligence (IJCAI 2019). 2019. p. 378–84.
6. Haret A, Khani H, Moretti S, Öztürk M. Ceteris paribus majority for social ranking. In: 27th international joint conference on artificial intelligence (IJCAI-ECAI-18). 2018. p. 303–9.
7. Fayard N, Öztürk-Escoffier M. Ordinal social ranking: Simulation for CP-majority rule. In: DA2PL'2018 (from multiple criteria decision aid to preference learning). 2018.
8. Bernardi G, Lucchetti R, Moretti S. Ranking objects from a preference relation over their subsets. Social Choice and Welfare. 2019;52:589–606.
9. Algaba E, Moretti S, Rémila E, Solal P. Lexicographic solutions for coalitional rankings. Social Choice and Welfare. 2021;1–33.
10. Allouche T, Escoffier B, Moretti S, Öztürk M. Social ranking manipulability for the CP-majority, banzhaf and lexicographic excellence solutions. In: Bessiere C, editor. Proceedings of the twenty-ninth international joint conference on artificial intelligence, IJCAI-20. International Joint Conferences on Artificial Intelligence Organization; 2020. p. 17–23. doi:10.24963/ijcai.2020/3.
11. Meyer D, Hornik K. Relations: Data structures and algorithms for relations. 2022. https://CRAN.R-project.org/package=relations.
12. Meyer D, Hornik K. Generalized and customizable sets in R. Journal of Statistical Software. 2009;31:1–27.
13. Moretti S. An axiomatic approach to social ranking under coalitional power relations. Homo Oeconomicus. 2015;32:183–208.
14. Serramia M, López-Sánchez M, Moretti S, Rodríguez-Aguilar JA. On the dominant set selection problem and its application to value alignment. Autonomous Agents and Multi-Agent Systems. 2021;35:1–38.
15. Copeland AH. A reasonable social welfare function. mimeo, 1951. University of Michigan; 1951.
16. Simpson PB. On defining areas of voter choice: Professor tullock on stable voting. The Quarterly Journal of Economics. 1969;83:478–90.
17. Kramer GH. A dynamical model of political equilibrium. Journal of Economic Theory. 1975;16:310–34.